

Vertica® Analytic Database 5.0

# SQL Reference Manual

Copyright© 2006-2011 Vertica, An HP Company

Date of Publication: June 20, 2011



CONFIDENTIAL

# Contents

<b>Technical Support</b>	<b>1</b>
--------------------------	----------

---

<b>About the Documentation</b>	<b>2</b>
--------------------------------	----------

---

Where to Find the Vertica Documentation .....	2
Reading the Online Documentation .....	2
Printing Full Books .....	4
Suggested Reading Paths .....	4
Where to Find Additional Information .....	6
Typographical Conventions.....	7

<b>Preface</b>	<b>9</b>
----------------	----------

---

<b>SQL Overview</b>	<b>10</b>
---------------------	-----------

---

<b>System Limits</b>	<b>11</b>
----------------------	-----------

---

<b>SQL Language Elements</b>	<b>12</b>
------------------------------	-----------

---

Keywords and Reserved Words .....	12
Keywords .....	12
Reserved Words .....	14
Identifiers .....	15
Literals .....	17
Number-type Literals .....	17
String Literals .....	19
Date/Time Literals .....	28
Operators .....	35
Binary Operators .....	35
Boolean Operators .....	37
Comparison Operators .....	37
Data Type Coercion Operators (CAST).....	38
Date/Time Operators .....	39
Mathematical Operators .....	40
NULL Operators .....	41
String Concatenation Operators .....	42
Expressions .....	43
Aggregate Expressions.....	44
CASE Expressions.....	45
Column References .....	46
Comments .....	47
Date/Time Expressions .....	48
NULL Value .....	50
Numeric Expressions.....	50

Predicates .....	51
BETWEEN-predicate.....	51
Boolean-predicate .....	52
column-value-predicate .....	53
IN-predicate .....	54
INTERPOLATE.....	54
join-predicate .....	58
LIKE-predicate.....	59
NULL-predicate .....	63

---

**SQL Data Types** **64**

---

Binary Data Types.....	65
Boolean Data Type.....	69
Character Data Types.....	70
Date/Time Data Types.....	72
DATE.....	73
DATETIME.....	74
INTERVAL .....	74
SMALLDATETIME.....	88
TIME.....	88
TIMESTAMP .....	90
Numeric Data Types .....	95
DOUBLE PRECISION (FLOAT) .....	97
INTEGER .....	100
NUMERIC .....	100
Data Type Coercion .....	104
Data Type Coercion Chart .....	107

---

**SQL Functions** **109**

---

Aggregate Functions.....	110
AVG [Aggregate] .....	110
CORR.....	111
COUNT [Aggregate].....	111
COVAR_POP.....	115
COVAR_SAMP.....	115
MAX [Aggregate].....	116
MIN [Aggregate] .....	117
REGR_AVGX .....	117
REGR_AVGY .....	118
REGR_COUNT .....	118
REGR_INTERCEPT.....	119
REGR_R2 .....	119
REGR_SLOPE.....	120
REGR_SXX.....	120
REGR_SXY.....	121
REGR_SYY.....	121
STDDEV [Aggregate].....	122
STDDEV_POP [Aggregate] .....	123
STDDEV_SAMP [Aggregate] .....	123
SUM [Aggregate] .....	124
SUM_FLOAT [Aggregate].....	125

VAR_POP [Aggregate] .....	126
VAR_SAMP [Aggregate] .....	127
VARIANCE [Aggregate] .....	127
Analytic Functions .....	128
window_partition_clause .....	130
window_order_clause .....	131
window_frame_clause .....	133
named_windows .....	135
AVG [Analytic] .....	136
CONDITIONAL_CHANGE_EVENT [Analytic] .....	137
CONDITIONAL_TRUE_EVENT [Analytic] .....	138
COUNT [Analytic] .....	139
CUME_DIST [Analytic] .....	140
DENSE_RANK [Analytic] .....	142
EXPONENTIAL_MOVING_AVERAGE [Analytic] .....	143
FIRST_VALUE [Analytic] .....	146
LAG [Analytic] .....	149
LAST_VALUE [Analytic] .....	152
LEAD [Analytic] .....	153
MAX [Analytic] .....	155
MEDIAN [Analytic] .....	156
MIN [Analytic] .....	158
NTILE [Analytic] .....	159
PERCENT_RANK [Analytic] .....	160
PERCENTILE_CONT [Analytic] .....	162
PERCENTILE_DISC [Analytic] .....	164
RANK [Analytic] .....	166
ROW_NUMBER [Analytic] .....	168
STDDEV [Analytic] .....	169
STDDEV_POP [Analytic] .....	170
STDDEV_SAMP [Analytic] .....	171
SUM [Analytic] .....	173
VAR_POP [Analytic] .....	174
VAR_SAMP [Analytic] .....	175
VARIANCE [Analytic] .....	176
Performance Optimization for Analytic Sort Computation .....	178
Boolean Functions .....	181
BIT_AND .....	181
BIT_OR .....	182
BIT_XOR .....	183
Date/Time Functions .....	184
ADD_MONTHS .....	185
AGE_IN_MONTHS .....	187
AGE_IN_YEARS .....	188
CLOCK_TIMESTAMP .....	189
CURRENT_DATE .....	190
CURRENT_TIME .....	190
CURRENT_TIMESTAMP .....	191
DATE_PART .....	191
DATE .....	195
DATE_TRUNC .....	196
DATEDIFF .....	197
DAY .....	202
DAYOFMONTH .....	203

DA YOFWEEK.....	204
DA YOFWEEK_ISO.....	204
DA YOFYEAR.....	205
DA YS.....	206
EXTRACT.....	207
GETDATE.....	210
GETUTCDATE.....	211
HOUR.....	211
ISFINITE.....	212
JULIAN_DAY.....	212
LAST_DAY.....	213
LOCALTIME.....	213
LOCALTIMESTAMP.....	214
MICROSECOND.....	215
MIDNIGHT_SECONDS.....	215
MINUTE.....	216
MONTH.....	216
MONTHS_BETWEEN.....	217
NEXT_DAY.....	219
NOW [Date/Time].....	220
OVERLAPS.....	220
QUARTER.....	221
ROUND [Date/Time].....	222
SECOND.....	223
STATEMENT_TIMESTAMP.....	223
SYSDATE.....	224
TIME_SLICE.....	225
TIMEOFDAY.....	230
TIMESTAMPADD.....	230
TIMESTAMPDIFF.....	232
TIMESTAMP_ROUND.....	233
TIMESTAMP_TRUNC.....	234
TRANSACTION_TIMESTAMP.....	235
TRUNC [Date/Time].....	236
WEEK.....	237
WEEK_ISO.....	238
YEAR.....	239
YEAR_ISO.....	240
Formatting Functions.....	241
TO_BITSTRING.....	241
TO_CHAR.....	242
TO_DATE.....	243
TO_HEX.....	244
TO_TIMESTAMP.....	245
TO_NUMBER.....	247
Template Patterns for Date/Time Formatting.....	248
Template Patterns for Numeric Formatting.....	250
Geospatial Package SQL Functions.....	251
Geospatial SQL Functions.....	252
BB_WITHIN.....	254
BEARING.....	255
CHORD_TO_ARC.....	255
DWITHIN.....	256
ECEP_CHORD.....	257

ECEF_x .....	257
ECEF_y .....	258
ECEF_z .....	258
ISLEFT .....	259
KM2MILES .....	260
LAT_WITHIN .....	260
LL_WITHIN .....	261
LLD_WITHIN .....	262
LON_WITHIN .....	262
MILES2KM .....	263
RADIUS_LON .....	264
RADIUS_M .....	264
RADIUS_N .....	265
radius_r .....	265
RADIUS_Ra .....	266
RADIUS_Rc .....	266
RADIUS_Rv .....	267
RADIUS_SI .....	268
RAYCROSSING .....	268
WGS84_a .....	269
WGS84_b .....	269
WGS84_e2 .....	270
WGS84_f .....	270
WGS84_if .....	271
WGS84_r1 .....	271
IP Conversion Functions .....	271
INET_ATON .....	272
INET_NTOA .....	273
V6_ATON .....	273
V6_NTOA .....	275
V6_SUBNETA .....	275
V6_SUBNETN .....	276
V6_TYPE .....	277
Mathematical Functions .....	279
ABS .....	279
ACOS .....	279
ASIN .....	280
ATAN .....	280
ATAN2 .....	281
CBRT .....	281
CEILING (CEIL) .....	282
COS .....	282
COT .....	283
DEGREES .....	283
DISTANCE .....	284
DISTANCEV .....	285
EXP .....	285
FLOOR .....	286
HASH .....	287
LN .....	288
LOG .....	288
MOD .....	289
MODULARHASH .....	290
PI .....	291

POWER .....	291
RADIANS .....	292
RANDOM .....	292
RANDOMINT .....	293
ROUND .....	293
SIGN .....	295
SIN .....	295
SQRT .....	296
TAN.....	296
TRUNC.....	297
WIDTH_BUCKET .....	298
NULL-handling Functions.....	299
COALESCE.....	299
IFNULL .....	300
ISNULL .....	301
NULLIF .....	303
NULLIFZERO .....	304
NVL.....	304
NVL2 .....	306
ZEROIFNULL .....	307
Sequence Functions.....	308
NEXTVAL.....	308
CURRVAL .....	310
LAST_INSERT_ID .....	311
String Functions.....	314
ASCII .....	314
BIT_LENGTH .....	315
BITCOUNT .....	316
BITSTRING_TO_BINARY .....	316
BTRIM.....	317
CHARACTER_LENGTH.....	318
CHR.....	318
CONCAT .....	319
DECODE.....	320
GREATEST .....	321
GREATESTB .....	322
HEX_TO_BINARY .....	324
INET_ATON .....	324
INET_NTOA .....	325
INITCAP .....	326
INITCAPB .....	327
INSERT .....	328
INSTR.....	328
INSTRB .....	330
ISUTF8.....	331
LEAST .....	332
LEASTB .....	333
LEFT .....	334
LENGTH.....	335
LOWER.....	336
LOWERB .....	336
LPAD.....	337
LTRIM.....	338
MD5 .....	338

OCTET_LENGTH .....	339
OVERLAY .....	340
OVERLAYB .....	341
POSITION .....	342
POSITIONB .....	343
QUOTE_IDENT .....	344
QUOTE_LITERAL .....	345
REPEAT .....	345
REPLACE .....	346
RIGHT .....	347
RPAD .....	348
RTRIM .....	348
SPACE .....	349
SPLIT_PART .....	350
SPLIT_PARTB .....	351
STRPOS .....	352
STRPOSB .....	352
SUBSTR .....	353
SUBSTRB .....	354
SUBSTRING .....	355
TO_BITSTRING .....	356
TO_HEX .....	357
TRANSLATE .....	358
TRIM .....	358
UPPER .....	359
UPPERB .....	360
V6_ATON .....	361
V6_NTOA .....	362
V6_SUBNETA .....	363
V6_SUBNETN .....	364
V6_TYPE .....	364
System Information Functions .....	367
CURRENT_DATABASE .....	367
CURRENT_SCHEMA .....	367
CURRENT_USER .....	368
HAS_TABLE_PRIVILEGE .....	368
SESSION_USER .....	369
USER .....	370
VERSION .....	371
URI Encode/Decode Functions .....	371
URI_PERCENT_DECODE .....	371
URI_PERCENT_ENCODE .....	372
Vertica Functions .....	374
Alphabetical List of Vertica Functions .....	374
Catalog Management Functions .....	473
Cluster Scaling Functions .....	479
Constraint Management Functions .....	480
Data Collector Functions .....	490
Database Management Functions .....	492
Epoch Management Functions .....	499
License Management Functions .....	505
Partition Management Functions .....	511
Pattern Matching Functions .....	519
Projection Management Functions .....	523

Purge Functions.....	530
Regular Expression Functions.....	533
Session Management Functions.....	546
Statistic Management Functions.....	556
Storage Management Functions.....	563
Timeseries Functions.....	569
Tuple Mover Functions.....	572
Workload Management Functions.....	573

---

**SQL Statements** **578**

---

ALTER FUNCTION.....	578
ALTER PROJECTION RENAME.....	580
ALTER PROFILE.....	580
ALTER PROFILE RENAME.....	582
ALTER RESOURCE POOL.....	582
ALTER ROLE RENAME.....	585
ALTER SCHEMA.....	585
ALTER SEQUENCE.....	586
ALTER TABLE.....	589
table-constraint.....	593
ALTER USER.....	594
COMMENT ON COLUMN.....	595
COMMENT ON CONSTRAINT.....	596
COMMENT ON FUNCTION.....	597
COMMENT ON LIBRARY.....	598
COMMENT ON NODE.....	599
COMMENT ON PROJECTION.....	600
COMMENT ON SCHEMA.....	601
COMMENT ON SEQUENCE.....	601
COMMENT ON TABLE.....	602
COMMENT ON VIEW.....	603
COMMIT.....	605
CONNECT.....	605
COPY.....	607
Parameters.....	608
COPY Formats.....	618
Notes.....	619
Examples.....	620
See Also.....	626
COPY FROM VERTICA.....	626
CREATE FUNCTION (SQL Macros).....	629
CREATE FUNCTION (UDF).....	631
CREATE LIBRARY.....	634
CREATE PROCEDURE.....	635
CREATE PROFILE.....	637
CREATE PROJECTION.....	640
encoding-type.....	644
hash-segmentation-clause.....	646
range-segmentation-clause.....	647
CREATE RESOURCE POOL.....	649
Built-in Pools.....	652
Built-in Pool Configuration.....	653

CREATE ROLE .....	657
CREATE SCHEMA .....	657
CREATE SEQUENCE.....	658
CREATE TABLE.....	664
column-definition (table) .....	670
column-name-list (table).....	671
column-constraint .....	674
table-constraint.....	678
hash-segmentation-clause (table).....	679
range-segmentation-clause (table) .....	680
CREATE TEMPORARY TABLE .....	682
column-definition (temp table) .....	687
column-name-list (temp table).....	688
hash-segmentation-clause (temp table) .....	690
range-segmentation-clause (temp table) .....	691

CREATE TRANSFORM FUNCTION .....	693
CREATE USER.....	695
CREATE VIEW .....	697
DELETE.....	699
DISCONNECT .....	700
DROP FUNCTION.....	700
DROP LIBRARY.....	701
DROP PROCEDURE.....	702
DROP PROFILE.....	703
DROP PROJECTION.....	704
DROP RESOURCE POOL .....	705
DROP ROLE.....	705
DROP SCHEMA .....	706
DROP TRANSFORM FUNCTION.....	707
DROP SEQUENCE .....	708
DROP TABLE.....	709
DROP USER.....	711
DROP VIEW .....	711
EXPLAIN .....	713
EXPORT .....	718
GRANT (Database).....	720
GRANT (Function).....	721
GRANT (Procedure).....	721
GRANT (Resource Pool).....	722
GRANT (Role).....	723
GRANT (Schema).....	725
GRANT (Sequence).....	725
GRANT (Table).....	727
GRANT (Transform Function).....	728
GRANT (View).....	728
INSERT.....	730
LCOPY.....	731
PROFILE.....	732
RELEASE SAVEPOINT .....	733
REVOKE (Database).....	734
REVOKE (Function).....	735
REVOKE (Procedure).....	736
REVOKE (Resource Pool).....	736
REVOKE (Role).....	737
REVOKE (Schema).....	739
REVOKE (Sequence).....	739
REVOKE (Table).....	741
REVOKE (Transform Function).....	742
REVOKE (View).....	742
ROLLBACK .....	744
ROLLBACK TO SAVEPOINT .....	744
SAVEPOINT.....	745
SELECT .....	747
INTO Clause.....	748
FROM Clause.....	750
WHERE Clause.....	752
GROUP BY Clause.....	754
HAVING Clause.....	756
ORDER BY Clause.....	757

LIMIT Clause .....	759
OFFSET Clause .....	760
TIMESERIES Clause.....	761
MATCH Clause .....	763
SET DATESTYLE.....	767
SET ESCAPE_STRING_WARNING.....	768
SET INTERVALSTYLE .....	768
SET LOCALE.....	769
SET ROLE.....	772
SET SEARCH_PATH.....	774
SET SESSION CHARACTERISTICS.....	776
SET SESSION MEMORYCAP.....	777
SET SESSION RESOURCE POOL .....	778
SET SESSION RUNTIMECAP .....	779
SET SESSION TEMPSPACECAP.....	780
SET STANDARD_CONFORMING_STRINGS .....	781
SET TIME_ZONE .....	782
Time Zone Names for Setting TIME_ZONE.....	783
SHOW .....	785
TRUNCATE TABLE .....	787
UNION .....	788
UPDATE.....	792

**SQL System Tables (Monitoring APIs)****794**

V_CATALOG Schema .....	795
COLUMNS .....	795
COMMENTS .....	797
DATABASES .....	798
DUAL .....	798
FOREIGN_KEYS.....	799
GRANTS .....	800
LICENSE_AUDITS .....	801
NODES .....	802
PASSWORDS .....	803
PRIMARY_KEYS.....	803
PROFILE_PARAMETERS .....	804
PROFILES .....	804
PROJECTION_COLUMNS .....	805
PROJECTION_DELETE_CONCERNS .....	811
PROJECTIONS.....	811
RESOURCE_POOLS .....	814
ROLES.....	815
SEQUENCES .....	815
SYSTEM_TABLES .....	817
TABLE_CONSTRAINTS .....	818
TABLES .....	820
TYPES .....	821
USER_AUDITS.....	821
USER_FUNCTIONS.....	822
USER_PROCEDURES .....	823
USER_TRANSFORMS .....	824
USERS.....	824

VIEW_COLUMNS .....	825
VIEWS.....	827
V_MONITOR Schema .....	828
ACTIVE_EVENTS .....	828
COLUMN_STORAGE.....	830
CONFIGURATION_PARAMETERS .....	832
CRITICAL_NODES .....	833
CURRENT_SESSION.....	834
DATA_COLLECTOR.....	837
DATABASE_CONNECTIONS .....	841
DATABASE_SNAPSHOTS .....	842
DELETE_VECTORS .....	843
DISK_RESOURCE_REJECTIONS .....	843
DISK_STORAGE.....	844
EVENT_CONFIGURATIONS .....	848
EXECUTION_ENGINE_PROFILES.....	849
HOST_RESOURCES .....	854
LOAD_STREAMS.....	856
LOCKS .....	858
NODE_RESOURCES .....	861
PARTITIONS.....	862
PROJECTION_REFRESHES .....	863
PROJECTION_STORAGE .....	865
QUERY_METRICS .....	868
QUERY_PROFILES.....	870
RESOURCE_ACQUISITIONS .....	872
RESOURCE_ACQUISITIONS_HISTORY.....	874
RESOURCE_POOL_STATUS .....	877
RESOURCE_QUEUES.....	882
RESOURCE_REJECTIONS .....	882
RESOURCE_USAGE .....	884
SESSION_PROFILES .....	886
SESSIONS .....	888
STORAGE_CONTAINERS.....	890
STRATA .....	893
STRATA_STRUCTURES.....	896
SYSTEM.....	898
TUNING_RECOMMENDATIONS .....	899
TUPLE_MOVER_OPERATIONS.....	901
USER_LIBRARIES .....	902
USER_LIBRARY_MANIFEST .....	903
WOS_CONTAINER_STORAGE .....	903

---

**Appendix: Compatibility with Other RDBMS** **907**

Data Type Mappings Between Vertica and Oracle .....	907
---	-----

---

**Index** **911**

---

**Copyright Notice** **921**

---



# Technical Support

---

To submit problem reports, questions, comments, and suggestions, use the Technical Support page on the Vertica Web site.

---

## Notes:

- You must be a registered user in order to access the ***MyVertica Portal*** ***<http://myvertica.vertica.com/v-zone/overview>***.
  - If you are not a registered user, you can request access at the ***Technical Support Web page*** ***<http://www.vertica.com/support>***.
- 

Before you report a problem, run the Diagnostics Utility described in the Troubleshooting Guide and attach the resulting `.zip` file to your ticket.

# About the Documentation

---

This section describes how to access and print Vertica documentation. It also includes *suggested reading paths* (page 4).

## Where to Find the Vertica Documentation

You can read or download the Vertica documentation for the current release of Vertica® Analytic Database from the **Product Documentation Page** [http://www.vertica.com/v-zone/product\\_documentation](http://www.vertica.com/v-zone/product_documentation). You must be a registered user to access this page.

The documentation is available as a compressed tarball (.tar) or a zip archive (.zip) file. When you extract the file on the database server system or locally on the client, contents are placed in a /vertica50\_doc/ directory.

---

### Notes:

- The documentation on the Vertica Web site is updated each time a new release is issued.
  - A more recent version of the product documentation might be available online. To check for critical product or document information added after the product release, see the Vertica Product Documentation downloads site. You can download the PDF version or browse books online
  - If you are using an older version of the software, refer to the documentation on your database server or client systems.
- 

See Installing Vertica Documentation in the Installation Guide.

## Reading the Online Documentation

### Reading the HTML documentation files

The Vertica documentation files are provided in HTML browser format for platform independence. The HTML files require only a browser that displays frames properly with JavaScript enabled. The HTML files do not require a Web (HTTP) server.

The Vertica documentation is supported on the following browsers:

- Mozilla FireFox
- Internet Explorer
- Apple Safari
- Opera
- Google Chrome (server-side installations only)

The instructions that follow assume you have installed the documentation on a client or server machine.

### Mozilla Firefox

- 1 Open a browser window.
- 2 Choose one of the following methods to access the documentation:
  - Select **File > Open File**, navigate to `..\HTML-WEBHELP\index.htm`, and click **Open**.
  - OR drag and drop `index.htm` into a browser window.
  - OR press **CTRL+O**, navigate to `index.htm`, and click **Open**.

### Internet Explorer

Use one of the following methods:

- 1 Open a browser window.
- 2 Choose one of the following methods to access the documentation:
  - Select **File > Open > Browse**, navigate to `..\HTML-WEBHELP\index.htm`, click **Open**, and click **OK**.
  - OR drag and drop `index.htm` into the browser window.
  - OR press **CTRL+O**, Browse to the file, click **Open**, and click **OK**.

**Note:** If a message warns you that Internet Explorer has restricted the web page from running scripts or ActiveX controls, right-click anywhere within the message and select **Allow Blocked Content**.

### Apple Safari

- 1 Open a browser window.
- 2 Choose one of the following methods to access the documentation:
  - Select **File > Open File**, navigate to `..\HTML-WEBHELP\index.htm`, and click **Open**.
  - OR drag and drop `index.htm` into the browser window.
  - OR press **CTRL+O**, navigate to `index.htm`, and click **Open**.

### Opera

- 1 Open a browser window.
- 2 Position your cursor in the title bar and right click > **Customize > Appearance**, click the **Toolbar** tab and select **Main Bar**.
- 3 Choose one of the following methods to access the documentation:
  - Open a browser window and click **Open**, navigate to `..\HTML-WEBHELP\index.htm`, and click **Open**.
  - OR drag and drop `index.htm` into the browser window.
  - OR press **CTRL+O**, navigate to `index.htm`, and click **Open**.

### Google Chrome

Google does not support access to client-side installations of the documentation. You'll have to point to the documentation installed on a server system.

- 1 Open a browser window.
- 2 Choose one of the following methods to access the documentation:
  - In the address bar, type the location of the `index.htm` file on the server. For example:  
<file://<servername>//vertica50 doc//HTML/Master/index.htm>
  - OR drag and drop `index.htm` into the browser window.
  - OR press **CTRL+O**, navigate to `index.htm`, and click **Open**.

## Notes

The `.tar` or `.zip` file you download contains a complete documentation set.

The documentation page of the **Downloads Web site** [http://www.vertica.com/v-zone/download\\_vertica](http://www.vertica.com/v-zone/download_vertica) is updated as new versions of Vertica are released. When the version you download is no longer the most recent release, refer only to the documentation included in your RPM.

The Vertica documentation contains links to Web sites of other companies or organizations that Vertica does not own or control. If you find broken links, please let us know.

Report any script, image rendering, or text formatting problems to **Technical Support** (on page 1).

## Printing Full Books

Vertica also publishes books as Adobe Acrobat™ PDF. The books are designed to be printed on standard 8½ x 11 paper using full duplex (two-sided) printing.

**Note:** Vertica manuals are topic driven and not meant to be read in a linear fashion. Therefore, the PDFs do not resemble the format of typical books.

Open and print the PDF documents using Acrobat Acrobat Reader. You can download the latest version of the free Reader from the **Adobe Web site** (<http://www.adobe.com/products/acrobat/readstep2.html>).

The following list provides links to the PDFs.

- Concepts Guide
- Installation Guide
- Getting Started Guide
- Administrator's Guide
- Programmer's Guide
- SQL Reference Manual
- Troubleshooting Guide

## Suggested Reading Paths

This section provides a suggested reading path for various users. Vertica recommends that you read the manuals listed under All Users first.

## **All Users**

- **New Features** — Release-specific information, including new features and behavior changes to the product and documentation
- **Concepts Guide** — Basic concepts critical to understanding Vertica
- **Getting Started Guide** — A tutorial that takes you through the process of configuring a Vertica database and running example queries
- **Troubleshooting Guide** — General troubleshooting information

## **System Administrators**

- **New Features** — Release-specific information, including new features and behavior changes to the product and documentation
- **Installation Guide** — Platform configuration and software installation

## **Database Administrators**

- **Installation Guide** — Platform configuration and software installation
- **Administrator's Guide** — Database configuration, loading, security, and maintenance

## **Application Developers**

- **Programmer's Guide** — Connecting to a database, queries, transactions, and so on
- **SQL Reference Manual** — SQL and Vertica-specific language information

## Where to Find Additional Information

Visit the *Vertica Web site* (<http://www.vertica.com>) to keep up to date with:

- Downloads
- Frequently Asked Questions (FAQs)
- Discussion forums
- News, tips, and techniques
- Training

## Typographical Conventions

The following are the typographical and syntax conventions used in the Vertica documentation.

Typographical Convention	Description
<b>Bold</b>	Indicates areas of emphasis, such as a special menu command.
<b>Button</b>	Indicates the word is a button on the window or screen.
Code	SQL and program code displays in a monospaced (fixed-width) font.
Database objects	Names of database objects, such as tables, are shown in san-serif type.
<i>Emphasis</i>	Indicates emphasis and the titles of other documents or system files.
monospace	Indicates literal interactive or programmatic input/output.
<i>monospace italics</i>	Indicates user-supplied information in interactive or programmatic input/output.
UPPERCASE	Indicates the name of a SQL command or keyword. SQL keywords are case insensitive; <code>SELECT</code> is the same as <code>Select</code> , which is the same as <code>select</code> .
<b>User input</b>	Text entered by the user is shown in bold san serif type.
↵	indicates the Return/Enter key; implicit on all user input that includes text
Right-angle bracket >	Indicates a flow of events, usually from a drop-down menu.
Click	Indicates that the reader clicks options, such as menu command buttons, radio buttons, and mouse selections; for example, "Click OK to proceed."
Press	Indicates that the reader perform some action on the keyboard; for example, "Press Enter."
Syntax Convention	Description
Text without brackets/braces	Indicates content you type as shown.
< <i>Text inside angle brackets</i> >	Placeholder for which you must supply a value. The variable is usually shown in italics. See Placeholders below.
[ Text inside brackets ]	Indicates optional items; for example, <code>CREATE TABLE [schema_name.]table_name</code> The brackets indicate that the <code>schema_name</code> is optional. Do not type the square brackets.
{ Text inside braces }	Indicates a set of options from which you choose one; for example: <code>QUOTES { ON   OFF }</code> indicates that exactly one of ON or OFF must

	be provided. You do not type the braces: QUOTES ON
Backslash \	Continuation character used to indicate text that is too long to fit on a single line.
Ellipses . . .	Indicate a repetition of the previous parameter. For example, <code>option[. . .]</code> means that you can enter multiple, comma-separated options.  Note: Showing an ellipses in code examples might also mean that part of the text has been omitted for readability, such as in multi-row result sets.
Indentation	Is an attempt to maximize readability; SQL is a free-form language.
<i>Placeholders</i>	Items that must be replaced with appropriate identifiers or expressions are shown in italics.
Vertical bar	Is a separator for mutually exclusive items. For example: <code>[ASC   DESC]</code>  Choose one or neither. You do not type the square brackets.

# Preface

---

This guide provides a reference description of the Vertica SQL database language.

## **Audience**

This document is intended for anyone who uses Vertica. It assumes that you are familiar with the basic concepts and terminology of the SQL language and relational database management systems.

# SQL Overview

---

An abbreviation for Structured Query Language, SQL is a widely-used, industry standard data definition and data manipulation language for relational databases.

**Note:** In Vertica, use a semicolon to end a statement or to combine multiple statements on one line.

## Vertica Support for ANSI SQL Standards

Vertica SQL supports a subset of ANSI SQL-99.

See *BNF Grammar for SQL-99* (<http://savage.net.au/SQL/sql-99.bnf.html>)

## Support for Historical Queries

Unlike most databases, the **DELETE** (page 699) command in Vertica does not delete data; it marks records as deleted. The **UPDATE** (page 792) command performs an INSERT and a DELETE. This behavior is necessary for historical queries. See Historical (Snapshot) Queries in the Programmer's Guide.

## Joins

Vertica supports typical data warehousing query joins. For details, see Joins in the Programmer's Guide.

Vertica also provides the **INTERPOLATE** (page 54) predicate, which allows for a special type of join. The event series join is a Vertica SQL extension that lets you analyze two event series when their measurement intervals don't align precisely—such as when timestamps don't match. These joins provide a natural and efficient way to query misaligned event data directly, rather than having to normalize the series to the same measurement interval. See Event Series Joins in the Programmer's Guide for details.

## Transactions

Session-scoped isolation levels determine transaction characteristics for transactions within a specific user session. You set them through the **SET SESSION CHARACTERISTICS** (page 776) command. Specifically, they determine what data a transaction can access when other transactions are running concurrently. See Transactions in the Concepts Guide.

# System Limits

---

This section describes system limits on the size and number of objects in a Vertica database. In most cases, computer memory and disk drive are the limiting factors.

Item	Limit
Database size	Approximates the number of files times the file size on a platform, depending on the maximum disk configuration.
Table size	2 <sup>64</sup> rows per node, or 2 <sup>63</sup> bytes per column, whichever is smaller.
Row size	8MB. The row size is approximately the sum of its maximum column sizes, where, for example a varchar(80) has a maximum size of 80 bytes.
Key size	1600 x 4000
Number of tables/projections per database	Limited by physical RAM, as the catalog must fit in memory.
Number of concurrent connections per node	Default of 50, limited by physical RAM (or threads per process), typically 1024.
Number of concurrent connections per cluster	Limited by physical RAM of a single node (or threads per process), typically 1024.
Number of columns per table	1600.
Number of rows per load	2 <sup>63</sup> .
Number of partitions	256. <b>Note:</b> The maximum number of partitions varies with the number of columns in the table, as well as system RAM. Vertica recommends a maximum of 20 partitions. Ideally, create no more than 12.
Length for a fixed-length column	65000 bytes.
Length for a variable-length column	65000 bytes.
Length of basic names	128 bytes. Basic names include table names, column names, etc.
Query length	No limit.
Depth of nesting subqueries	Unlimited in FROM or WHERE or HAVING clause.

# SQL Language Elements

---

This chapter presents detailed descriptions of the language elements and conventions of Vertica SQL.

## Keywords and Reserved Words

Keywords are words that have a specific meaning in the SQL language. Although SQL is not case-sensitive with respect to keywords, they are generally shown in uppercase letters throughout this documentation for readability purposes.

Some SQL keywords are also reserved words that cannot be used in an identifier unless enclosed in double quote (") characters.

## Keywords

Keyword are words that are specially handled by the grammar. Every SQL statement contains one or more keywords.

Begins with	Keyword
A	ABORT, ABSOLUTE, ACCESS, ACCESRANK, ACCOUNT, ACTION, ADD, ADMIN, AFTER, AGGREGATE, ALL, ALSO, ALTER, ANALYSE, ANALYZE, AND, ANY, ARRAY, AS, ASC, ASSERTION, ASSIGNMENT, AT, AUTHORIZATION, AUTO, AUTO_INCREMENT, AVAILABLE
B	BACKWARD, BEFORE, BEGIN, BETWEEN, BIGINT, BINARY, BIT, BLOCK_DICT, BLOCKDICT_COMP, BOOLEAN, BOTH, BY, BYTEA, BZIP
C	CACHE, CALLED, CASCADE, CASE, CAST, CATALOGPATH, CHAIN, CHAR, CHAR_LENGTH, CHARACTER, CHARACTER_LENGTH, CHARACTERISTICS, CHARACTERS, CHECK, CHECKPOINT, CLASS, CLOSE, CLUSTER, COLLATE, COLUMN, COLUMNS_COUNT, COMMENT, COMMIT, COMMITTED, COMMONDELTA_COMP, CONNECT, CONSTRAINT, CONSTRAINTS, COPY, CORRELATION, CREATE, CREATEDB, CREATEUSER, CROSS, CSV, CURRENT, CURRENT_DATABASE, CURRENT_DATE, CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_USER, CURSOR, CYCLE
D	DATA, DATABASE, DATAPATH, DATE, DATEDIFF, DATETIME, DAY, DEALLOCATE, DEC, DECIMAL, DECLARE, DECODE, DEFAULT, DEFAULTS, DEFERRABLE, DEFERRED, DEFINE, DEFINER, DELETE, DELIMITER, DELIMITERS, DELTARANGE_COMP, DELTARANGE_COMP_SP, DELTAVAL, DESC, DETERMINES, DIRECT, DIRECTCOLS, DIRECTGROUPED, DIRECTPROJ, DISABLE, DISCONNECT, DISTINCT, DISTVALINDEX, DO, DOMAIN, DOUBLE, DROP, DURABLE
E	EACH, ELSE, ENABLE, ENABLED, ENCLOSED, ENCODED, ENCODING, ENCRYPTED, END, ENFORCELENGTH, EPHEMERAL, EPOCH, ERROR, ESCAPE, EVENT, EVENTS, EXCEPT, EXCEPTIONS, EXCLUDE, EXCLUDING, EXCLUSIVE, EXECUTE, EXISTS, EXPIRE, EXPLAIN, EXPORT, EXTERNAL, EXTRACT

F	FAILED_LOGIN_ATTEMPTS, FALSE, FETCH, FILLER, FIRST, FLOAT, FOLLOWING, FOR, FORCE, FOREIGN, FORMAT, FORWARD, FREEZE, FROM, FULL, FUNCTION
G	GCDDELTA, GLOBAL, GRANT, GROUP, GROUPE D, GZIP
H	HANDLER, HASH, HAVING, HOLD, HOSTNAME, HOUR, HOURS
I	IDENTIFIED, IDENTITY, IF, IGNORE, ILIKE, ILIKEB, IMMEDIATE, IMMUTABLE, IMPLICIT, IN, INCLUDING, INCREMENT, INDEX, INHERITS, INITIALLY, INNER, INOUT, INPUT, INSENSITIVE, INSERT, INSTEAD, INT, INTEGER, INTERPOLATE, INTERSECT, INTERVAL, INTERVALYM, INTO, INVOKER, IS, ISNULL, ISOLATION
J	JOIN
K	KEY, KSAFE
L	LANCOMPILER, LANGUAGE, LARGE, LAST, LATEST, LEADING, LEFT, LESS, LEVEL, LIBRARY, LIKE, LIKEB, LIMIT, LISTEN, LOAD, LOCAL, LOCALTIME, LOCALTIMESTAMP, LOCATION, LOCK
M	MANAGED, MATCH, MAXCONCURRENCY, MAXMEMORYSIZE, MAXVALUE, MEMORYCAP, MEMORYSIZE, MERGEOUT, MICROSECONDS, MILLISECONDS, MINUTE, MINUTES, MINVALUE, MODE, MONEY, MONTH, MOVE, MOVEOUT
N	NAME, NATIONAL, NATIVE, NATURAL, NCHAR, NEW, NEXT, NO, NOCREATEDB, NOCREATEUSER, NODE, NODES, NONE, NOT, NOTHING, NOTIFY, NOTNULL, NOWAIT, NULL, NULLCOLS, NULLS, NULLSEQUAL, NULLIF, NUMBER, NUMERIC
O	OBJECT, OCTETS, OF, OFF, OFFSET, OIDS, OLD, ON, ONLY, OPERATOR, OPTION, OR, ORDER, OTHERS, OUT, OUTER, OVER, OVERLAPS, OVERLAY, OWNER
P	PARTIAL, PARTITION, PASSWORD, PASSWORD_GRACE_TIME, PASSWORD_LIFE_TIME, PASSWORD_LOCK_TIME, PASSWORD_MAX_LENGTH, PASSWORD_MIN_DIGITS, PASSWORD_MIN_LENGTH, PASSWORD_MIN_LETTERS, PASSWORD_MIN_LOWERCASE_LETTERS, PASSWORD_MIN_SYMBOLS, PASSWORD_MIN_UPPERCASE_LETTERS, PASSWORD_REUSE_MAX, PASSWORD_REUSE_TIME, PATTERN, PERCENT, PERMANENT, PINNED, PLACING, PLANNEDCONCURRENCY, POOL, POSITION, PRECEDING, PRECISION, PREPARE, PRESERVE, PREVIOUS, PRIMARY, PRIOR, PRIORITY, PRIVILEGES, PROCEDURAL, PROCEDURE, PROFILE, PROJECTION
Q	QUEUE_TIMEOUT, QUOTE
R	RANGE, RAW, READ, REAL, RECHECK, RECORD, RECOVER, REFERENCES, REFRESH, REINDEX, REJECTED, REJECTMAX, RELATIVE, RELEASE, RENAME, REPEATABLE, REPLACE, RESET, RESOURCE, RESTART, RESTRICT, RESULTS, RETURN, RETURNREJECTED, REVOKE, RIGHT, RLE, ROLE, ROLES, ROLLBACK, ROW, ROWS, RULE, RUNTIMECAP
S	SAMPLE, SAVEPOINT, SCHEMA, SCROLL, SECOND, SECONDS, SECURITY, SEGMENTED, SELECT, SEQUENCE, SERIALIZABLE, SESSION, SESSION_USER, SET, SETOF, SHARE, SHOW, SIMILAR, SIMPLE, SINGLEINITIATOR, SITE, SITES, SKIP, SMALLDATE TIME, SMALLINT, SOME,

	SPLIT, STABLE, START, STATEMENT, STATISTICS, STDERR, STDIN, STDOUT, STORAGE, STREAM, STRICT, SUBSTRING, SYSDATE, SYSID
T	TABLE, TABLESPACE, TEMP, TEMPLATE, TEMPORARY, TEMPSPACECAP, TERMINATOR, THAN, THEN, TIES, TIME, TIMESERIES, TIMESTAMP, TIMESTAMPADD, TIMESTAMPDIFF, TIMESTAMPTZ, TIMETZ, TIMEZONE, TINYINT, TO, TOAST, TRAILING, TRANSACTION, TRANSFORM, TREAT, TRICKLE, TRIGGER, TRIM, TRUE, TRUNCATE, TRUSTED, TUNING, TYPE
U	UNBOUNDED, UNCOMMITTED, UNCOMPRESSED, UNENCRYPTED, UNION, UNIQUE, UNKNOWN, UNLIMITED, UNLISTEN, UNLOCK, UNSEGMENTED, UNTIL, UPDATE, USAGE, USER, USING
V	VACUUM, VALIDATOR, VALINDEX, VALUE, VALUES, VARBINARY, VARCHAR, VARCHAR2, VARYING, VERBOSE, VERTICA, VIEW, VOLATILE
W	WAIT, WHEN, WHERE, WINDOW, WITH, WITHIN, WITHOUT, WORK, WRITE
Y	YEAR
Z	ZONE

## Reserved Words

Many SQL keywords are also reserved words, but a reserved word is not necessarily a keyword. For example, a reserved word might be reserved for other/future use. In Vertica, reserved words can be used anywhere an identifier is used, as long as you double-quote it.

Begins with	Reserved Word
A	ALL, ANALYSE, ANALYZE, AND, ANY, ARRAY, AS, ASC
B	BINARY, BOTH
C	CASE, CAST, CHECK, COLUMN, CONSTRAINT, CORRELATION, CREATE, CURRENT_DATABASE, CURRENT_DATE, CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_USER
D	DEFAULT, DEFERRABLE, DESC, DISTINCT, DO
E	ELSE, ENCODED, END, EXCEPT
F	FALSE, FOR, FOREIGN, FROM
G	GRANT, GROUP, GROUPE D
H	HAVING
I	IN, INITIALLY, INTERSECT, INTERVAL, INTERVALYM, INTO
J	JOIN
K	KSAFE
L	LEADING, LIMIT, LOCALTIME, LOCALTIMESTAMP
M	MATCH
N	NEW, NOT, NULL, NULLSEQUAL

O	OFF, OFFSET, OLD, ON, ONLY, OR, ORDER
P	PINNED, PLACING, PRIMARY, PROJECTION
R	REFERENCES
S	SCHEMA, SEGMENTED, SELECT, SESSION_USER, SOME, SYSDATE
T	TABLE, THEN, TIMESERIES, TO, TRAILING, TRUE
U	UNBOUNDED, UNION, UNIQUE, UNSEGMENTED, USER, USING
W	WHEN, WHERE, WINDOW, WITH, WITHIN

## Identifiers

Identifiers (names) of objects such as schema, table, projection, column names, and so on, can be up to 128 bytes in length.

### Unquoted Identifiers

Unquoted SQL identifiers must begin with one of the following:

- An alphabetic character (A-Z or a-z, including letters with diacritical marks and non-Latin letters)
- Underscore (\_)

Subsequent characters in an identifier can be:

- Alphabetic
- Digits(0-9)
- Dollar sign (\$). Dollar sign is not allowed in identifiers according to the SQL standard and could cause application portability problems.

### Quoted Identifiers

Identifiers enclosed in double quote (") characters can contain any character. If you want to include a double quote, you need a pair of them; for example """". You can use names that would otherwise be invalid, such as names that include only numeric characters ("123") or contain space characters, punctuation marks, keywords, and so on; for example, `CREATE SEQUENCE "my sequence!";`

Double quotes are required for non-alphanumerics and SQL keywords such as "1time", "Next week" and "Select".

**Note:** Identifiers are not case-sensitive. Thus, identifiers "ABC", "ABc", and "aBc" are synonymous, as are ABC, ABc, and aBc .

## Identifiers Are Stored As Created

SQL identifiers, such as table and column names, are no longer converted to lowercase. They are stored as created, and references to them are resolved using case-insensitive compares. It is not necessary to double quote mixed-case identifiers. For example, The following statement creates table ALLCAPS.

```
=> CREATE TABLE ALLCAPS(c1 varchar(30));
=> INSERT INTO ALLCAPS values('upper case');
```

The following statements are variations of the same query and all return identical results:

```
=> SELECT * FROM ALLCAPS;
=> SELECT * FROM allcaps;
=> SELECT * FROM "allcaps";
```

All three commands return the same result:

```
      c1
-----
upper case
(1 row)
```

Note that the system returns an error if you try to create table AllCaps:

```
=> CREATE TABLE allcaps(c1 varchar(30));
      ROLLBACK: table "AllCaps" already exists
```

See QUOTE\_IDENT (page 344) for additional information.

### Special note about Case-sensitive System Tables

The `V_CATALOG.TABLES` (page 820) `TABLE_SCHEMA` and `TABLE_NAME` columns are case sensitive when used with an equality (=) predicate in queries. For example, given the following sample schema, if you execute a query using the = predicate, Vertica returns 0 rows:

```
=> CREATE SCHEMA SS;
=> CREATE TABLE SS.TT (c1 int);
=> INSERT INTO ss.tt VALUES (1);
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema
='ss';
```

```
table_schema | table_name
-----+-----
(0 rows)
```

**TIP:** Use the case-insensitive **ILIKE** predicate to return the expected results.

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema
ILIKE 'ss';
```

```
table_schema | table_name
-----+-----
SS           | TT
(1 row)
```

## Literals

Literals are numbers or strings used in SQL as constants. Literals are included in the select-list, along with expressions and built-in functions and can also be constants.

Vertica provides support for number-type literals (integers, numerics, and floating points) string literals, and date/time literals. The various string literal formats are discussed in this section.

### Number-type Literals

There are three types of numbers in Vertica: Integers, numerics, and floats.

- **Integers** (page 100) are whole numbers less than  $2^{63}$  and must be digits.
- **Numerics** (page 100) are very large integers or include a decimal point with a precision and a scale.

**Note:** Whole numbers that are larger than  $2^{63}$  are treated as numerics. Numbers with a decimal point but no exponent are treated as numerics with default precision and scale.

- **Floating point** (page 97) literals are like numerics with the addition of an exponent.

Numeric-type values can also be generated using casts from character strings. This is a more general syntax. See the Examples section below, as well as **Data Type Coercion Operators (CAST)** (page 38).

## Syntax

```
digits
digits.[digits] | [digits].digits
digits e[+-]digits | [digits].digits e[+-]digits | digits.[digits] e[+-]digits
```

## Parameters

*digits* represents one or more numeric characters (0 through 9).

## Notes

- At least one digit must follow the exponent marker (*e*), if *e* is present.
- There cannot be any spaces or other characters embedded in the constant.
- Leading plus (+) or minus (-) signs are not considered part of the constant; they are unary operators applied to the constant.
- A numeric constant that contains neither a decimal point nor an exponent is initially presumed to be type `INTEGER` if its value fits; otherwise it is presumed to be `NUMERIC`.
- In most cases a numeric-type constant is automatically coerced to the most appropriate type depending on context. When necessary, you can force a numeric value to be interpreted as a specific data type by casting it as described in ***Data Type Coercion Operators (CAST)*** (page 38).
- Vertica follows the IEEE specification for floating point, including NaN (not a number) and Infinity (Inf).
- A NaN is not greater than and at the same time not less than anything, even itself. In other words, comparisons always return false whenever a NaN is involved. See ***Numeric Expressions*** (page 50) for examples.

## Examples

The following are examples of number-type literals:

```
42
3.5
4.
.001
5e2
1.925e-3
```

### Scientific notation :

```
=> SELECT NUMERIC '1e10';
   ?column?
-----
10000000000
(1 row)
```

### BINARY scaling:

```
=> SELECT NUMERIC '1p10';
   ?column?
-----
1024
(1 row)
```

```
=> SELECT FLOAT 'Infinity';
?column?
-----
Infinity
(1 row)
```

## See Also

**Data Type Coercion** (page 104)

## String Literals

String literals are string values surrounded by single or double quotes. Double-quoted strings are subject to the backslash, but single-quoted strings do not require a backslash, except for `'` and `\\`.

You can embed single quotes and backslashes into single-quoted strings.

To include other backslash (escape) sequences, such as `\t` (tab), you must use the double-quoted form.

Single quoted strings require a preceding space between them and the preceding word because single quotes are allowed in identifiers.

### Standard Conforming Strings and Escape Characters

When interpreting commands, such as those entered in `vsq` or in queries passed via JDBC or ODBC, Vertica uses standard conforming strings as specified in the SQL standard. In standard conforming strings, backslashes are treated as string literals (ordinary characters), not escape characters.

**Note:** Text read in from files or streams (such as the data inserted using the **COPY** (page 607) statement) are not treated as literal strings. The **COPY** command defines its own escape characters for the data it reads. See the **COPY** (page 607) statement documentation for details.

In Vertica databases prior to 4.0, the standard conforming strings was not on by default, and backslashes were considered escape sequences. After 4.0, escape sequences, including Windows path names, did not work as they had previously. For example, the TAB character `'\t'` is two characters: `'\'` and `'t'`.

`E'...'` is the **Extended character string literal** (page 23) format, so to treat backslashes as escape characters, use `E'\t'`.

The following options are available, but Vertica recommends that you migrate your application to use standard conforming strings at your earliest convenience, after warnings have been addressed.

- To revert to pre 4.0 behavior, set the `StandardConformingStrings` parameter to `'0'`, as described in Configuration Parameters in the Administrator's Guide.
- To enable standard conforming strings permanently, set the `StandardConformingStrings` parameter to `'1'`, as described in the procedure in the section, "Identifying Strings that are not Standard Conforming," below.

- To enable standard conforming strings per session, use **SET STANDARD\_CONFORMING\_STRING\_TO\_ON** (page 781), which treats backslashes as escape characters for the current session only.

The two sections that follow help you identify issues between Vertica 3.5 and 4.0.

### Identifying Strings that are not Standard Conforming

The following procedure can be used to identify non-standard conforming strings in your application so that you can convert them into standard conforming strings:

- 1 Be sure the StandardConformingStrings parameter is off, as described in Internationalization Parameters in the Administrator's Guide.

```
=> SELECT SET_CONFIG_PARAMETER ('StandardConformingStrings' , '0');
```

**Note:** Vertica recommends that you migrate your application to use Standard Conforming Strings at your earliest convenience.

- 2 Turn on the EscapeStringWarning parameter. (ON is the default in <DBMS\_SHORT 4.0.)

```
=> SELECT SET_CONFIG_PARAMETER ('EscapeStringWarning', '1');
```

Vertica now returns a warning each time it encounters an escape string within a string literal. For example, Vertica interprets the `\n` in the following example as a new line:

```
=> SELECT 'a\nb';
      WARNING:  nonstandard use of escape in a string literal at character
      8
      HINT:    Use the escape string syntax for escapes, e.g., E'\r\n'.
      ?column?
-----
      a
      b
      (1 row)
```

When StandardConformingStrings is ON, the string is interpreted as four characters: a \ n b.

Modify each string that Vertica flags by extending it as in the following example:

```
E'a\nb'
```

Or if the string has quoted single quotes, double them; for example, 'one' ' double'.

- 3 Turn on the StandardConformingStrings parameter for all sessions:

```
SELECT SET_CONFIG_PARAMETER ('StandardConformingStrings' , '1');
```

### Doubled Single Quotes

This section discusses vsql inputs that are not passed on to the server.

Vertica recognizes two consecutive single quotes within a string literal as one single quote character. For example, the following inputs, 'You''re here!' ignored the second consecutive quote and returns the following:

```
=> SELECT 'You''re here!';
      ?column?
-----
      You're here!
      (1 row)
```

This is the SQL standard representation and is preferred over the form, 'You\'re here!', because backslashes are not parsed as before. You need to escape the backslash:

```
=> SELECT (E'You\'re here!');
      ?column?
-----
You're here!
(1 row)
```

This behavior change introduces a potential incompatibility in the use of the `vsq` `\set` command, which automatically concatenates its arguments. For example, the following works in both Vertica 3.5 and 4.0:

```
\set file '\'' `pwd` '/file.txt' '\''
\echo :file
```

`vsq` takes the four arguments and outputs the following:

```
'/home/vertica/file.txt'
```

In Vertica 3.5 the above `\set file` command could be written all with the arguments run together, but in 4.0 the adjacent single quotes are now parsed differently:

```
\set file '\''`pwd`'/file.txt'\''
\echo :file
'/home/vertica/file.txt''
```

Note the extra single quote at the end. This is due to the pair of adjacent single quotes together with the backslash-quoted single quote.

The extra quote can be resolved either as in the first example above, or by combining the literals as follows:

```
\set file '\''`pwd`'/file.txt''
\echo :file
'/home/vertica/file.txt'
```

In either case the backslash-quoted single quotes should be changed to doubled single quotes as follows:

```
\set file '''' `pwd` '/file.txt''''
```

## See Also

**STANDARD\_CONFORMING\_STRINGS** (page 781)

**ESCAPE\_STRING\_WARNING** (page 768)

Internationalization Parameters and Implement Locales for International Data Sets in the Administrator's Guide

## String Literals (Character)

Character string literals are a sequence of characters from a predefined character set and are enclosed by single quotes. If the single quote is part of the sequence, it must be doubled as `''`.

## Syntax

```
'characters'
```

## Parameters

*characters* is an arbitrary sequence of characters bounded by single quotes (').

### Single Quotes in a String

The SQL standard way of writing a single-quote character within a string literal is to write two adjacent single quotes. For example:

```
=> SELECT 'Chester''s gorilla';
      ?column?
-----
Chester's gorilla
(1 row)
```

### Standard Conforming Strings and Escape Characters

Vertica uses standard conforming strings as specified in the SQL standard, which means that backslashes are treated as string literals, not escape characters.

**Note:** Earlier versions of Vertica did not use standard conforming strings, and backslashes were always considered escape sequences. To revert to this older behavior, set the `StandardConformingStrings` parameter to '0', as described in Configuration Parameters in the Administrator's Guide.

### Examples

```
=> SELECT 'This is a string';
      ?column?
-----
This is a string
(1 row)
```

```
=> SELECT 'This \is a string';
WARNING: nonstandard use of escape in a string literal at character 8
HINT: Use the escape string syntax for escapes, e.g., E'\r\n'.
      ?column?
-----
This is a string
(1 row)
```

```
vmartdb=> SELECT E'This \is a string';
      ?column?
-----
This is a string
=> SELECT E'This is a \n new line';
      ?column?
-----
This is a
new line
(1 row)
```

```
=> SELECT 'String''s characters';
      ?column?
-----
```

```
String's characters
(1 row)
```

### See Also

**STANDARD\_CONFORMING\_STRINGS** (page 781) and **ESCAPE\_STRING\_WARNING** (page 768) in the SQL Reference Manual

Internationalization Parameters and Implement Locales for International Data Sets in the Administrator's Guide

## Extended String Literals

### Syntax

```
E'characters'
```

### Parameters

*characters* is an arbitrary sequence of characters bounded by single quotes (').

You can use C-style backslash sequence in extended string literals, which are an extension to the SQL standard. You specify an extended string literal by writing the letter E as a prefix (before the opening single quote); for example:

```
E'extended character string\n'
```

Within an extended string, the backslash character (\) starts a C-style backslash sequence, in which the combination of backslash and following character or numbers represent a special byte value, as shown in the following list. Any other character following a backslash is taken literally; for example, to include a backslash character, write two backslashes (\\).

- \\ is a backslash
- \b is a backspace
- \f is a form feed
- \n is a newline
- \r is a carriage return
- \t is a tab
- \x## is a tab where ## is a 1 or 2-digit hexadecimal number
- \###, where ### is a 1, 2, or 3-digit octal number representing a byte with the corresponding code.

When an extended string literal is concatenated across lines, write only E before the first opening quote:

```
=> SELECT E'first part o'
->      'f a long line';
      ?column?
-----
first part of a long line
(1 row)
```

Two adjacent single quotes are used as one single quote:

```
=> SELECT 'Aren't string literals fun?';
```

```
?column?
```

```
-----  
Aren't string literals fun?  
(1 row)
```

## Standard Conforming Strings and Escape Characters

When interpreting commands, such as those entered in `vsq`l or in queries passed via JDBC or ODBC, Vertica uses standard conforming strings as specified in the SQL standard. In standard conforming strings, backslashes are treated as string literals (ordinary characters), not escape characters.

**Note:** Text read in from files or streams (such as the data inserted using the **COPY** (page 607) statement) are not treated as literal strings. The **COPY** command defines its own escape characters for the data it reads. See the **COPY** (page 607) statement documentation for details.

In Vertica databases prior to 4.0, the standard conforming strings was not on by default, and backslashes were considered escape sequences. After 4.0, escape sequences, including Windows path names, did not work as they had previously. For example, the TAB character `'\t'` is two characters: `'\'` and `'t'`.

`E'...'`  is the **Extended character string literal** (page 23) format, so to treat backslashes as escape characters, use `E'\t'`.

The following options are available, but Vertica recommends that you migrate your application to use standard conforming strings at your earliest convenience, after warnings have been addressed.

- To revert to pre 4.0 behavior, set the `StandardConformingStrings` parameter to `'0'`, as described in Configuration Parameters in the Administrator's Guide.
- To enable standard conforming strings permanently, set the `StandardConformingStrings` parameter to `'1'`, as described in the procedure in the section, "Identifying Strings that are not Standard Conforming," below.
- To enable standard conforming strings per session, use **SET STANDARD\_CONFORMING\_STRING TO ON** (page 781), which treats backslashes as escape characters for the current session only.

The two sections that follow help you identify issues between Vertica 3.5 and 4.0.

## Identifying Strings that are not Standard Conforming

The following procedure can be used to identify non-standard conforming strings in your application so that you can convert them into standard conforming strings:

- 1 Be sure the `StandardConformingStrings` parameter is off, as described in Internationalization Parameters in the Administrator's Guide.

```
=> SELECT SET_CONFIG_PARAMETER ('StandardConformingStrings' , '0');
```

**Note:** Vertica recommends that you migrate your application to use Standard Conforming Strings at your earliest convenience.

- 2 Turn on the `EscapeStringWarning` parameter. (ON is the default in `<DBMS_SHORT 4.0.>`)

```
=> SELECT SET_CONFIG_PARAMETER ('EscapeStringWarning', '1');
```

Vertica now returns a warning each time it encounters an escape string within a string literal. For example, Vertica interprets the `\n` in the following example as a new line:

```
=> SELECT 'a\nb';
      WARNING:  nonstandard use of escape in a string literal at character
      8
      HINT:  Use the escape string syntax for escapes, e.g., E'\r\n'.
      ?column?
-----
      a
      b
      (1 row)
```

When `StandardConformingStrings` is ON, the string is interpreted as four characters: `a \ n b`.

Modify each string that Vertica flags by extending it as in the following example:

```
E'a\nb'
```

Or if the string has quoted single quotes, double them; for example, `'one' ' double'`.

### 3 Turn on the `StandardConformingStrings` parameter for all sessions:

```
SELECT SET_CONFIG_PARAMETER ('StandardConformingStrings' , '1');
```

## Doubled Single Quotes

This section discusses `vsq` inputs that are not passed on to the server.

Vertica recognizes two consecutive single quotes within a string literal as one single quote character. For example, the following inputs, `'You''re here!'` ignored the second consecutive quote and returns the following:

```
=> SELECT 'You''re here!';
      ?column?
-----
      You're here!
      (1 row)
```

This is the SQL standard representation and is preferred over the form, `'You\'re here!'`, because backslashes are not parsed as before. You need to escape the backslash:

```
=> SELECT (E'You\'re here!');
      ?column?
-----
      You're here!
      (1 row)
```

This behavior change introduces a potential incompatibility in the use of the `vsq \set` command, which automatically concatenates its arguments. For example, the following works in both Vertica 3.5 and 4.0:

```
\set file '\'' `pwd` '/file.txt' '\''
\echo :file
```

`vsq` takes the four arguments and outputs the following:

```
'/home/vertica/file.txt'
```

In Vertica 3.5 the above `\set file` command could be written all with the arguments run together, but in 4.0 the adjacent single quotes are now parsed differently:

```
\set file '\''`pwd`'/file.txt'\''
\echo :file
'/home/vertica/file.txt''
```

Note the extra single quote at the end. This is due to the pair of adjacent single quotes together with the backslash-quoted single quote.

The extra quote can be resolved either as in the first example above, or by combining the literals as follows:

```
\set file '\''`pwd`'/file.txt''
\echo :file
'/home/vertica/file.txt'
```

In either case the backslash-quoted single quotes should be changed to doubled single quotes as follows:

```
\set file "" `pwd` '/file.txt''
```

### Additional Examples

```
=> SELECT 'This \is a string';
      ?column?
-----
This \is a string
(1 row)

=> SELECT E'This \is a string';
      ?column?
-----
This is a string
=> SELECT E'This is a \n new line';
      ?column?
-----
This is a
new line
(1 row)

=> SELECT 'String''s characters';
      ?column?
-----
String's characters
(1 row)
```

## Unicode String Literals

### Syntax

```
U&'characters' [ UESCAPE '<Unicode escape character>' ]
```

### Parameters

*characters* is an arbitrary sequence of UTF-8 characters bounded by single quotes (').

*Unicode escape character* is a single character from the source language character set other than a hexit, plus sign (+), quote ('), double quote ("), or white space.

When StandardConformingStrings is enabled, Vertica supports SQL standard Unicode character string literals (the character set is UTF-8 only).

Before entering a Unicode character string literal, enable standard conforming strings in one of the following ways.

- To enable for all sessions, update the StandardConformingStrings configuration parameter. See Configuration Parameters in the Administrator's Guide.
- To treat backslashes as escape characters for the current session, use the **SET STANDARD\_CONFORMING\_STRINGS** (page 781) statement.

## Examples

To enter a Unicode character in hexadecimal, use the following syntax:

```
=> SET STANDARD_CONFORMING_STRINGS TO ON;
```

To enter, for example, the Russian phrase for "thank you":

```
=> SELECT U&'\0441\043F\0430\0441\0438\0431\043E' as 'thank you';
      thank you
-----
      спасибо
(1 row)
```

To enter in hexadecimal, for example, the German word 'müde' (where u is really u-umlaut):

```
=> SELECT U&'m\00fcde';
?column?
-----
      müde
(1 row)
```

```
=> SELECT 'ü';
?column?
-----
      ü
(1 row)
```

## See Also

**STANDARD\_CONFORMING\_STRINGS** (page 781) and **ESCAPE\_STRING\_WARNING** (page 768) in the SQL Reference Manual

Internationalization Parameters and Implement Locales for International Data Sets in the Administrator's Guide

## String Literals (Dollar-Quoted)

Dollar-quoted string literals are rarely used but are provided here for your convenience.

The standard syntax for specifying string literals can be difficult to understand. To allow more readable queries in such situations, Vertica SQL provides "dollar quoting." Dollar quoting is not part of the SQL standard, but it is often a more convenient way to write complicated string literals than the standard-compliant single quote syntax.

## Syntax

`$$characters$$`

## Parameters

*characters* is an arbitrary sequence of characters bounded by paired dollar signs (\$\$).

Dollar-quoted string content is treated as a literal. Single quote, backslash, and dollar sign characters have no special meaning within a dollar-quoted string.

## Notes

A dollar-quoted string that follows a keyword or identifier must be separated from the preceding word by whitespace; otherwise the dollar quoting delimiter would be taken as part of the preceding identifier.

## Examples

```
=> SELECT $$Fred's\n car$$;  
      ?column?
```

```
-----  
Fred's\n car  
(1 row)
```

```
=> SELECT 'SELECT 'fact''';  
ERROR:  syntax error at or near "';'" at character 21  
LINE 1: SELECT 'SELECT 'fact''';
```

```
=> SELECT 'SELECT $$fact'$$;  
      ?column?
```

```
-----  
SELECT $$fact  
(1 row)
```

```
=> SELECT 'SELECT ''fact''';  
      ?column?
```

```
-----  
SELECT 'fact';  
(1 row)
```

## Date/Time Literals

Date or time literal input must be enclosed in single quotes. Input is accepted in almost any reasonable format, including ISO 8601, SQL-compatible, traditional POSTGRES, and others.

Vertica is more flexible in handling date/time input than the SQL standard requires. The exact parsing rules of date/time input and for the recognized text fields including months, days of the week, and time zones are described in *Date/Time Expressions* (page 48).

## Time Zone Values

Vertica attempts to be compatible with the SQL standard definitions for time zones. However, the SQL standard has an odd mix of date and time types and capabilities. Obvious problems are:

- Although the `DATE` (page 73) type does not have an associated time zone, the `TIME` (page 88) type can. Time zones in the real world have little meaning unless associated with a date as well as a time, since the offset can vary through the year with daylight-saving time boundaries.
- Vertica assumes your local time zone for any data type containing only date or time.
- The default time zone is specified as a constant numeric offset from UTC. It is therefore not possible to adapt to daylight-saving time when doing date/time arithmetic across DST boundaries.

To address these difficulties, Vertica recommends using Date/Time types that contain both date and time when you use time zones. Vertica recommends that you do *not* use the type `TIME WITH TIME ZONE`, even though it is supported for legacy applications and for compliance with the SQL standard.

Time zones and time-zone conventions are influenced by political decisions, not just earth geometry. Time zones around the world became somewhat standardized during the 1900's, but continue to be prone to arbitrary changes, particularly with respect to daylight-savings rules.

Vertica currently supports daylight-savings rules over the time period 1902 through 2038, corresponding to the full range of conventional UNIX system time. Times outside that range are taken to be in "standard time" for the selected time zone, no matter what part of the year in which they occur.

Example	Description
PST	Pacific Standard Time
-8:00	ISO-8601 offset for PST
-800	ISO-8601 offset for PST
-8	ISO-8601 offset for PST
zulu	Military abbreviation for UTC
z	Short form of <code>zulu</code>

## Day of the Week Names

The following tokens are recognized as names of days of the week:

Day	Abbreviations
SUNDAY	SUN
MONDAY	MON
TUESDAY	TUE, TUES
WEDNESDAY	WED, WEDS
THURSDAY	THU, THUR, THURS
FRIDAY	FRI
SATURDAY	SAT

## Month Names

The following tokens are recognized as names of months:

Month	Abbreviations
JANUARY	JAN
FEBRUARY	FEB
MARCH	MAR
APRIL	APR
MAY	MAY
JUNE	JUN
JULY	JUL
AUGUST	AUG
SEPTEMBER	SEP, SEPT
OCTOBER	OCT
NOVEMBER	NOV
DECEMBER	DEC

## Interval Values

An interval value represents the duration between two points in time.

### Syntax

```
[ @ ] quantity unit [ quantity unit... ] [ AGO ]
```

### Parameters

@	(at sign) is optional and ignored
quantity	Is an integer <b>numeric constant</b> (page 17)
unit	Is one of the following units or abbreviations or plurals of the following units: MILLISECOND      DAY      DECADE SECOND            WEEK      CENTURY MINUTE            MONTH     MILLENNIUM HOUR              YEAR
AGO	[Optional] specifies a negative interval value (an interval going back in time). 'AGO' is a synonym for '-'.

The amounts of different units are implicitly added up with appropriate sign accounting.

## Notes

- Quantities of days, hours, minutes, and seconds can be specified without explicit unit markings. For example:  
'1 12:59:10' is read the same as '1 day 12 hours 59 min 10 sec'
- The boundaries of an interval constant are:
  - '9223372036854775807 usec' to '9223372036854775807 usec ago'
  - 296533 years 3 mons 21 days 04:00:54.775807 to -296533 years -3 mons -21 days -04:00:54.775807
- The range of an interval constant is  $\pm 2^{63} - 1$  (plus or minus two to the sixty-third minus one) microseconds.
- In Vertica, the interval fields are additive and accept large floating point numbers.

## Examples

```

SELECT INTERVAL '1 12:59:10';
?column?
-----
1 12:59:10
(1 row)
SELECT INTERVAL '9223372036854775807 usec';
?column?
-----
106751991 04:00:54.775807
(1 row)
SELECT INTERVAL '-9223372036854775807 usec';
?column?
-----
-106751991 04:00:54.775807
(1 row)
SELECT INTERVAL '-1 day 48.5 hours';
?column?
-----
-3 00:30
(1 row)
SELECT TIMESTAMP 'Apr 1, 07' - TIMESTAMP 'Mar 1, 07';
?column?
-----
31
(1 row)
SELECT TIMESTAMP 'Mar 1, 07' - TIMESTAMP 'Feb 1, 07';
?column?
-----
28
(1 row)
SELECT TIMESTAMP 'Feb 1, 07' + INTERVAL '29 days';
?column?
-----
03/02/2007 00:00:00
(1 row)

```

```

SELECT TIMESTAMP WITHOUT TIME ZONE '1999-10-01' + INTERVAL '1 month - 1
second'
AS "Oct 31";
      Oct 31
-----
10/30/1999 23:59:59
(1 row)

```

**interval-literal**

The following table lists the units allowed for an `interval-literal` parameter.

Unit	Description
a	Julian year, 365.25 days exactly
ago	Indicates negative time offset
c, cent, century	Century
centuries	Centuries
d, day	Day
days	Days
dec, decade	Decade
decades, decs	Decades
h, hour, hr	Hour
hours, hrs	Hours
ka	Julian kilo-year, 365250 days exactly
m	Minute or month for year/month, depending on context. See Notes below this table.
microsecond	Microsecond
microseconds	Microseconds
mil, millennium	Millennium
millennia, mils	Millennia
millisecond	Millisecond
milliseconds	Milliseconds
min, minute, mm	Minute
mins, minutes	Minutes
mon, month	Month
mons, months	Months
ms, msec, millisecond	Millisecond
mseconds, msecs	Milliseconds

q, qtr, quarter	Quarter
qtrs, quarters	Quarters
s, sec, second	Second
seconds, secs	Seconds
us, usec	Microsecond
microseconds, useconds, usecs	Microseconds
w, week	Week
weeks	Weeks
y, year, yr	Year
years, yrs	Years

## Notes

The input unit 'm' can represent either 'months' or 'minutes,' depending on context. To illustrate, the following command creates a one-column table with some interval vales:

```
=> CREATE TABLE int_test(i INTERVAL YEAR TO MONTH);
```

In the first INSERT statement, the values are inserted as 1 year, six months:

```
=> INSERT INTO int_test VALUES('1 year 6 months');
```

In the second INSERT statement, the minutes value is ignored, as the DAY TO SECOND part is truncated:

```
=> INSERT INTO int_test VALUES('1 year 6 minutes');
```

In the third INSERT statement, the 'm' counts as minutes value is ignored, as the DAY TO SECOND part is truncated:

```
=> INSERT INTO int_test VALUES('1 year 6 m'); -- the m counts as months
Query the table and you will notice that the second row does not contain the minutes input:
```

```
=> SELECT * FROM int_test;
   i
-----
 1-6
 1-0
 1-6
(3 rows)
```

In the following command, the the 'm' counts as minutes, because the DAY TO SECOND interval-qualifier extracts day/time values from the input:

```
=> SELECT INTERVAL '1y6m' DAY TO SECOND;
?column?
-----
 365 00:06
(1 row)
```

**interval-qualifier**

The following table lists the optional interval qualifiers. Values in `INTERVAL` fields, other than `SECOND`, are integers with a default precision of 2 when they are not the first field.

Interval Type	Units	Valid interval-literal entries
Day/time intervals	DAY	Unconstrained.
	DAY TO HOUR	An interval that represents a span of days and hours.
	DAY TO MINUTE	An interval that represents a span of days and minutes.
	DAY TO SECOND	(Default) interval that represents a span of days, hours, minutes, seconds, and fractions of a second if subtype unspecified.
	HOUR	Hours within days.
	HOUR TO MINUTE	An interval that represents a span of hours and minutes.
	HOUR TO SECOND	An interval that represents a span of hours and seconds.
	MINUTE	Minutes within hours.
	MINUTE TO SECOND	An interval that represents a span of minutes and seconds.
	SECOND	Seconds within minutes. <b>Note:</b> The <code>SECOND</code> field can have an interval fractional seconds precision, which indicates the number of decimal digits maintained following the decimal point in the <code>SECONDS</code> value. When <code>SECOND</code> is not the first field, it has a precision of 2 places before the decimal point.
Year/month intervals	MONTH	Months within year.
	YEAR	Unconstrained.
	YEAR TO MONTH	An interval that represents a span of years and months.

**Notes**

You cannot combine day/time and year/month qualifiers. For example, the following intervals are not allowed:

- DAY TO YEAR

- HOUR TO MONTH

## Operators

Operators are logical, mathematical, and equality symbols used in SQL to evaluate, compare, or calculate values.

### Binary Operators

Each of the functions in the following table works with binary and varbinary data types.

Operator	Function	Description
'='	binary_eq	Equal to
'<>'	binary_ne	Not equal to
'<'	binary_lt	Less than
'<='	binary_le	Less than or equal to
'>'	binary_gt	Greater than
'>='	binary_ge	Greater than or equal to
'&'	binary_and	And
'~'	binary_not	Not
' '	binary_or	Or
'#'	binary_xor	Either or
'  '	binary_cat	Concatenate

### Notes

If the arguments vary in length binary operators treat the values as though they are all equal in length by right-extending the smaller values with the zero byte to the full width of the column (except when using the `binary_cat` function). For example, given the values `'ff'` and `'f'`, the value `'f'` is treated as `'f0'`.

Operators are strict with respect to nulls. The result is null if any argument is null. For example, `null <> 'a'::binary` returns null.

To apply the OR (`'|'`) operator to a varbinary type, explicitly cast the arguments; for example:

```
=> SELECT '1'::VARBINARY | '2'::VARBINARY;
?column?
-----
3
(1 row)
```

Similarly, to apply the **LENGTH** (page 335), **REPEAT** (page 345), **TO\_HEX** (page 244), and **SUBSTRING** (page 355) functions to a binary type, explicitly cast the argument; for example:

```
=> SELECT LENGTH('\001\002\003\004'::varbinary(4));
```

```

LENGTH
-----
         4
(1 row)

```

When applying an operator or function to a column, the operator's or function's argument type is derived from the column type.

### Examples

In the following example, the zero byte is not removed from column cat1 when values are concatenated:

```

=> SELECT 'ab'::BINARY(3) || 'cd'::BINARY(2) AS cat1, 'ab'::VARBINARY(3) ||
      'cd'::VARBINARY(2) AS cat2;
   cat1   | cat2
-----+-----
 ab\000cd | abcd
(1 row)

```

When the binary value `'ab'::binary(3)` is translated to varbinary, the result is equivalent to `'ab\\000'::varbinary(3)`; for example:

```

=> SELECT 'ab'::binary(3);
   binary
-----
 ab\000
(1 row)

```

The following example performs a bitwise AND operation on the two input values (see also **BIT\_AND** (page 181)):

```

=> SELECT '10001' & '011' as AND;
   AND
-----
     1
(1 row)

```

The following example performs a bitwise OR operation on the two input values (see also **BIT\_OR** (page 182)):

```

=> SELECT '10001' | '011' as OR;
   OR
-----
 10011
(1 row)

```

The following example concatenates the two input values:

```

=> SELECT '10001' || '011' as CAT;
   CAT
-----
10001011
(1 row)

```

## Boolean Operators

### Syntax

[ AND | OR | NOT ]

### Parameters

SQL uses a three-valued Boolean logic where the null value represents "unknown."

a	b	a AND b	a OR b
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	NULL	NULL	NULL

a	NOT a
TRUE	FALSE
FALSE	TRUE
NULL	NULL

### Notes

- The operators `AND` and `OR` are commutative, that is, you can switch the left and right operand without affecting the result. However, the order of evaluation of subexpressions is not defined. When it is essential to force evaluation order, use a **CASE** (page 45) construct.
- Do not confuse Boolean operators with the **Boolean-predicate** (page 52) or the **Boolean** (page 69) data type, which can have only two values: true and false.

## Comparison Operators

Comparison operators are available for all data types where comparison makes sense. All comparison operators are binary operators that return values of True, False, or NULL.

### Syntax and Parameters

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

= or <=>	equal
<> or !=	not equal

### Notes

- The != operator is converted to <> in the parser stage. It is not possible to implement != and <> operators that do different things.
- The comparison operators return NULL (signifying "unknown") when either operand is null.
- The <=> operator performs an equality comparison like the = operator, but it returns true, instead of NULL, if both operands are NULL, and false, instead of NULL, if one operand is NULL.

## Data Type Coercion Operators (CAST)

Data type coercion (casting) passes an expression value to an input conversion routine for a specified data type, resulting in a constant of the indicated type.

### Syntax

```
CAST ( expression AS data-type )
      expression::data-type
      data-type 'string'
```

### Parameters

<i>expression</i>	Is an expression of any type
<i>data-type</i>	Converts the value of <i>expression</i> to one of the following data types: <b>BINARY</b> (page 65) <b>BOOLEAN</b> (page 69) <b>CHARACTER</b> (page 70) <b>DATE/TIME</b> (page 72) <b>NUMERIC</b> (page 95)

### Notes

- In Vertica, data type coercion (casting) can be invoked only by an explicit cast request. It must use, for example, one of the following constructs:  
CAST(x AS data-type-name)  
or  
x::data-type-name
- Type coercion format of *data-type* '*string*' can be used only to specify the data type of a quoted string constant.
- The explicit type cast can be omitted if there is no ambiguity as to the type the constant must be. For example, when a constant is assigned directly to a column, it is automatically coerced to the column's data type.
- If a binary value is cast (implicitly or explicitly) to a binary type with a smaller length, the value is silently truncated. For example:

```
=> SELECT 'abcd'::BINARY(2);
   binary
-----
    ab
(1 row)
```

- No casts other than BINARY to and from VARBINARY and resize operations are currently supported.
- On binary data that contains a value with fewer bytes than the target column, values are right-extended with the zero byte '\0' to the full width of the column. Trailing zeros on variable length binary values are not right-extended:

```
=> SELECT 'ab'::BINARY(4), 'ab'::VARBINARY(4);
   binary | varbinary
-----+-----
 ab\000\000 | ab
(1 row)
```

## Examples

```
=> SELECT CAST((2 + 2) AS VARCHAR);
   varchar
-----
    4
(1 row)
=> SELECT (2 + 2)::VARCHAR;
   varchar
-----
    4
(1 row)
=> SELECT '2.2' + 2;
ERROR:  invalid input syntax for integer: "2.2"
=> SELECT FLOAT '2.2' + 2;
?column?
-----
    4.2
(1 row)
```

## See Also

**Data Type Coercion** (page 104)

## Date/Time Operators

### Syntax

```
[ + | - | * | / ]
```

### Parameters

```
+ Addition
- Subtraction
* Multiplication
/ Division
```

## Notes

- The operators described below that take `TIME` or `TIMESTAMP` inputs actually come in two variants: one that takes `TIME WITH TIME ZONE` or `TIMESTAMP WITH TIME ZONE`, and one that takes `TIME WITHOUT TIME ZONE` or `TIMESTAMP WITHOUT TIME ZONE`. For brevity, these variants are not shown separately.
- The `+` and `*` operators come in commutative pairs (for example both `DATE + INTEGER` and `INTEGER + DATE`); only one of each such pair is shown.

Example	Result Type	Result
<code>DATE '2001-09-28' + INTEGER '7'</code>	DATE	'2001-10-05'
<code>DATE '2001-09-28' + INTERVAL '1 HOUR'</code>	TIMESTAMP	'2001-09-28 01:00:00'
<code>DATE '2001-09-28' + TIME '03:00'</code>	TIMESTAMP	'2001-09-28 03:00:00'
<code>INTERVAL '1 DAY' + INTERVAL '1 HOUR'</code>	INTERVAL	'1 DAY 01:00:00'
<code>TIMESTAMP '2001-09-28 01:00' + INTERVAL '23 HOURS'</code>	TIMESTAMP	'2001-09-29 00:00:00'
<code>TIME '01:00' + INTERVAL '3 HOURS'</code>	TIME	'04:00:00'
<code>- INTERVAL '23 HOURS'</code>	INTERVAL	'-23:00:00'
<code>DATE '2001-10-01' - DATE '2001-09-28'</code>	INTEGER	'3'
<code>DATE '2001-10-01' - INTEGER '7'</code>	DATE	'2001-09-24'
<code>DATE '2001-09-28' - INTERVAL '1 HOUR'</code>	TIMESTAMP	'2001-09-27 23:00:00'
<code>TIME '05:00' - TIME '03:00'</code>	INTERVAL	'02:00:00'
<code>TIME '05:00' - INTERVAL '2 HOURS'</code>	TIME	'03:00:00'
<code>TIMESTAMP '2001-09-28 23:00' - INTERVAL '23 HOURS'</code>	TIMESTAMP	'2001-09-28 00:00:00'
<code>INTERVAL '1 DAY' - INTERVAL '1 HOUR'</code>	INTERVAL	'1 DAY -01:00:00'
<code>TIMESTAMP '2001-09-29 03:00' - TIMESTAMP '2001-09-27 12:00'</code>	INTERVAL	'1 DAY 15:00:00'
<code>900 * INTERVAL '1 SECOND'</code>	INTERVAL	'00:15:00'
<code>21 * INTERVAL '1 DAY'</code>	INTERVAL	'21 DAYS'
<code>DOUBLE PRECISION '3.5' * INTERVAL '1 HOUR'</code>	INTERVAL	'03:30:00'
<code>INTERVAL '1 HOUR' / DOUBLE PRECISION '1.5'</code>	INTERVAL	'00:40:00'

## Mathematical Operators

Mathematical operators are provided for many data types.

Operator	Description	Example	Result
!	Factorial	<code>5 !</code>	120
+	Addition	<code>2 + 3</code>	5
-	Subtraction	<code>2 - 3</code>	-1
*	Multiplication	<code>2 * 3</code>	6
/	Division (integer division truncates results)	<code>4 / 2</code>	2
%	Modulo (remainder)	<code>5 % 4</code>	1
^	Exponentiation	<code>2.0 ^ 3.0</code>	8
/	Square root	<code> / 25.0</code>	5
/	Cube root	<code>  / 27.0</code>	3

!!	Factorial (prefix operator)	!! 5	120
@	Absolute value	@ -5.0	5
&	Bitwise AND	91 & 15	11
	Bitwise OR	32   3	35
#	Bitwise XOR	17 # 5	20
~	Bitwise NOT	~1	-2
<<	Bitwise shift left	1 << 4	16
>>	Bitwise shift right	8 >> 2	2

## Notes

- The bitwise operators work only on integer data types, whereas the others are available for all numeric data types.
- Vertica supports the use of the factorial operators on positive and negative floating point (`DOUBLE PRECISION` (page 97)) numbers as well as integers. For example:

```
=> SELECT 4.98!;
      ?column?
-----
 115.978600750905
(1 row)
```
- Factorial is defined in term of the gamma function, where  $(-1) = \text{Infinity}$  and the other negative integers are undefined. For example

```
(-4)! = NaN
-4! = -(4!) = -24.
```
- Factorial is defined as  $z! = \text{gamma}(z+1)$  for all complex numbers  $z$ . See the *Handbook of Mathematical Functions* <http://www.math.sfu.ca/~cbm/aands/> (1964) Section 6.1.5.
- See `MOD()` (page 289) for details about the behavior of `%`.

## NULL Operators

To check whether a value is or is not NULL, use the constructs:

```
expression IS NULL expression IS NOT NULL
```

Alternatively, use equivalent, but nonstandard, constructs:

```
expression ISNULL expression NOTNULL
```

Do not write `expression = NULL` because NULL is not "equal to" NULL. (The null value represents an unknown value, and it is not known whether two unknown values are equal.) This behavior conforms to the SQL standard.

**Note:** Some applications might expect that `expression = NULL` returns true if `expression` evaluates to the null value. Vertica strongly recommends that these applications be modified to comply with the SQL standard.

## String Concatenation Operators

To concatenate two strings on a single line, use the concatenation operator (two consecutive vertical bars).

### Syntax

```
string || string
```

### Parameters

<i>string</i>	Is an expression of type CHAR or VARCHAR
---------------	--

### Notes

- || is used to concatenate expressions and constants. The expressions are cast to VARCHAR if possible, otherwise to VARBINARY, and must both be one or the other.
- Two consecutive strings within a single SQL statement on separate lines are automatically concatenated

### Examples

The following example is a single string written on two lines:

```
=> SELECT E'xx'
-> '\\';
?column?
-----
xx\
(1 row)
```

This example shows two strings concatenated:

```
=> SELECT E'xx' ||
-> '\\';
?column?
-----
xx\\
(1 row)

=> SELECT 'auto' || 'mobile';
?column?
-----
automobile
(1 row)
=> SELECT 'auto'
-> 'mobile';
?column?
-----
automobile
(1 row)
=> SELECT 1 || 2;
?column?
-----
12
```

```

(1 row)
=> SELECT '1' || '2';
   ?column?
-----
    12
(1 row)
=> SELECT '1'
   ?column?
-----
    12
(1 row)

```

## Expressions

SQL expressions are the components of a query that compare a value or values against other values. They can also perform calculations. Expressions found inside any SQL command are usually in the form of a conditional statement.

### Operator Precedence

The following table shows operator precedence in decreasing (high to low) order.

**Note:** When an expression includes more than one operator, Vertica recommends that you specify the order of operation using parentheses, rather than relying on operator precedence.

Operator/Element	Associativity	Description
.	left	table/column name separator
::	left	typecast
[ ]	left	array element selection
-	right	unary minus
^	left	exponentiation
* / %	left	multiplication, division, modulo
+ -	left	addition, subtraction
IS		IS TRUE, IS FALSE, IS UNKNOWN, IS NULL
IN		set membership
BETWEEN		range containment
OVERLAPS		time interval overlap
LIKE		string pattern matching
< >		less than, greater than
=	right	equality, assignment
NOT	right	logical negation

AND	left	logical conjunction
OR	left	logical disjunction

### Expression Evaluation Rules

The order of evaluation of subexpressions is not defined. In particular, the inputs of an operator or function are not necessarily evaluated left-to-right or in any other fixed order. To force evaluation in a specific order, use a `CASE` (page 45) construct. For example, this is an untrustworthy way of trying to avoid division by zero in a `WHERE` clause:

```
=> SELECT x, y WHERE x <> 0 AND y/x > 1.5;
```

But this is safe:

```
=> SELECT x, y
WHERE
CASE
  WHEN x <> 0 THEN y/x > 1.5
  ELSE false
END;
```

A `CASE` construct used in this fashion defeats optimization attempts, so use it only when necessary. (In this particular example, it would be best to avoid the issue by writing `y > 1.5*x` instead.)

### Aggregate Expressions

An aggregate expression represents the application of an **aggregate function** (page 110) across the rows or groups of rows selected by a query.

Using `AVG()` as an example, the syntax of an aggregate expression is one of the following:

- Invokes the aggregate across all input rows for which the given expression yields a non-null value:  
`AVG (expression)`
- Is the same as `AVG (expression)`, because `ALL` is the default:  
`AVG (ALL expression)`
- Invokes the `AVG()` function across all input rows for all distinct, non-null values of the expression, where `expression` is any value expression that does not itself contain an aggregate expression.  
`AVG (DISTINCT expression)`

An aggregate expression only can appear in the select list or `HAVING` clause of a `SELECT` statement. It is forbidden in other clauses, such as `WHERE`, because those clauses are evaluated before the results of aggregates are formed.

## CASE Expressions

The CASE expression is a generic conditional expression that can be used wherever an expression is valid. It is similar to case and if/then/else statements in other languages.

### Syntax (form 1)

```
CASE
  WHEN condition THEN result
  [ WHEN condition THEN result ]...
  [ ELSE result ]
END
```

### Parameters

<i>condition</i>	Is an expression that returns a boolean (true/false) result. If the result is false, subsequent WHEN clauses are evaluated in the same manner.
<i>result</i>	Specifies the value to return when the associated <i>condition</i> is true.
ELSE <i>result</i>	If no <i>condition</i> is true then the value of the CASE expression is the result in the ELSE clause. If the ELSE clause is omitted and no condition matches, the result is null.

### Syntax (form 2)

```
CASE expression
  WHEN value THEN result
  [ WHEN value THEN result ]...
  [ ELSE result ]
END
```

### Parameters

<i>expression</i>	Is an expression that is evaluated and compared to all the <i>value</i> specifications in the WHEN clauses until one is found that is equal.
<i>value</i>	Specifies a value to compare to the <i>expression</i> .
<i>result</i>	Specifies the value to return when the <i>expression</i> is equal to the specified <i>value</i> .
ELSE <i>result</i>	Specifies the value to return when the <i>expression</i> is not equal to any <i>value</i> ; if no ELSE clause is specified, the value returned is null.

### Notes

The data types of all the result expressions must be convertible to a single output type.

## Examples

```
=> SELECT * FROM test;
```

```
a  
---
```

```
1  
2  
3
```

```
=> SELECT a,  
        CASE WHEN a=1 THEN 'one'  
              WHEN a=2 THEN 'two'  
              ELSE 'other'  
        END  
FROM test;
```

```
a | case  
---+-----
```

```
1 | one  
2 | two  
3 | other
```

```
=> SELECT a,  
        CASE a WHEN 1 THEN 'one'  
              WHEN 2 THEN 'two'  
              ELSE 'other'  
        END  
FROM test;
```

```
a | case  
---+-----
```

```
1 | one  
2 | two  
3 | other
```

## Special Example

A `CASE` expression does not evaluate subexpressions that are not needed to determine the result. You can use this behavior to avoid division-by-zero errors:

```
=> SELECT x FROM T1 WHERE  
        CASE WHEN x <> 0 THEN y/x > 1.5  
        ELSE false  
        END;
```

## Column References

### Syntax

```
[ [ schemaname. ] tablename. ] columnname
```

### Parameters

<i>schemaname</i>	Is the name of the schema
<i>tablename</i>	Is one of:

	<ul style="list-style-type: none"> <li>▪ The name of a table</li> <li>▪ An alias for a table defined by means of a FROM clause in a query</li> </ul>
<i>columnname</i>	Is the name of a column that must be unique across all the tables being used in a query

## Notes

There are no space characters in a column reference.

If you do not specify a *schemaname*, Vertica searches the existing schemas according to the order defined in the SET SEARCH\_PATH (page 774) command.

## Example

This example uses the schema from the VMart Example Database.

In the following command, *transaction\_type* and *transaction\_time* are the unique column references, *store* is the name of the schema, and *store\_sales\_fact* is the table name:

```
=> SELECT transaction_type, transaction_time
      FROM store.store_sales_fact
      ORDER BY transaction_time;
transaction_type | transaction_time
```

```
-----+-----
purchase        | 00:00:23
purchase        | 00:00:32
purchase        | 00:00:54
purchase        | 00:00:54
purchase        | 00:01:15
purchase        | 00:01:30
purchase        | 00:01:50
return          | 00:03:34
return          | 00:03:35
purchase        | 00:03:39
purchase        | 00:05:13
purchase        | 00:05:20
purchase        | 00:05:23
purchase        | 00:05:27
purchase        | 00:05:30
purchase        | 00:05:35
purchase        | 00:05:35
purchase        | 00:05:42
return          | 00:06:36
purchase        | 00:06:39
(20 rows)
```

## Comments

A comment is an arbitrary sequence of characters beginning with two consecutive hyphen characters and extending to the end of the line. For example:

```
-- This is a standard SQL comment
```

A comment is removed from the input stream before further syntax analysis and is effectively replaced by white space.

Alternatively, C-style block comments can be used where the comment begins with `/*` and extends to the matching occurrence of `*/`.

```
/* multiline comment
 * with nesting: /* nested block comment */
*/
```

These block comments nest, as specified in the SQL standard. Unlike C, you can comment out larger blocks of code that might contain existing block comments.

## Date/Time Expressions

Vertica uses an internal heuristic parser for all date/time input support. Dates and times are input as strings, and are broken up into distinct fields with a preliminary determination of what kind of information might be in the field. Each field is interpreted and either assigned a numeric value, ignored, or rejected. The parser contains internal lookup tables for all textual fields, including months, days of the week, and time zones.

The date/time type inputs are decoded using the following procedure.

- Break the input string into tokens and categorize each token as a string, time, time zone, or number.
- If the numeric token contains a colon (:), this is a time string. Include all subsequent digits and colons.
- If the numeric token contains a dash (-), slash (/), or two or more dots (.), this is a date string which might have a text month.
- If the token is numeric only, then it is either a single field or an ISO 8601 concatenated date (for example, 19990113 for January 13, 1999) or time (for example, 141516 for 14:15:16).
- If the token starts with a plus (+) or minus (-), then it is either a time zone or a special field.
- If the token is a text string, match up with possible strings.
- Do a binary-search table lookup for the token as either a special string (for example, today), day (for example, Thursday), month (for example, January), or noise word (for example, at, on).
- Set field values and bit mask for fields. For example, set year, month, day for today, and additionally hour, minute, second for now.
- If not found, do a similar binary-search table lookup to match the token with a time zone.
- If still not found, throw an error.
- When the token is a number or number field:
  - If there are eight or six digits, and if no other date fields have been previously read, then interpret as a "concatenated date" (for example, 19990118 or 990118). The interpretation is `YYYYMMDD` or `YYMMDD`.
  - If the token is three digits and a year has already been read, then interpret as day of year.
  - If four or six digits and a year has already been read, then interpret as a time (`HHMM` or `HHMMSS`).
  - If three or more digits and no date fields have yet been found, interpret as a year (this forces `yy-mm-dd` ordering of the remaining date fields).

- Otherwise the date field ordering is assumed to follow the `DateStyle` setting: mm-dd-yy, dd-mm-yy, or yy-mm-dd. Throw an error if a month or day field is found to be out of range.
- If BC has been specified, negate the year and add one for internal storage. (There is no year zero in the Gregorian calendar, so numerically 1 BC becomes year zero.)
- If BC was not specified, and if the year field was two digits in length, then adjust the year to four digits. If the field is less than 70, then add 2000, otherwise add 1900.

**Tip:** Gregorian years AD 1-99 can be entered by using 4 digits with leading zeros (for example, 0099 is AD 99).

### Month Day Year Ordering

For some formats, ordering of month, day, and year in date input is ambiguous and there is support for specifying the expected ordering of these fields. See *Date/Time Run-Time Parameters* for information about output styles.

### Special Date/Time Values

Vertica supports several special date/time values for convenience, as shown below. All of these values need to be written in single quotes when used as constants in SQL statements.

The values `INFINITY` and `-INFINITY` are specially represented inside the system and are displayed the same way. The others are simply notational shorthands that are converted to ordinary date/time values when read. (In particular, `NOW` and related strings are converted to a specific time value as soon as they are read.)

String	Valid Data Types	Description
<code>epoch</code>	<code>DATE</code> , <code>TIMESTAMP</code>	1970-01-01 00:00:00+00 (UNIX SYSTEM TIME ZERO)
<code>INFINITY</code>	<code>TIMESTAMP</code>	Later than all other time stamps
<code>-INFINITY</code>	<code>TIMESTAMP</code>	Earlier than all other time stamps
<code>NOW</code>	<code>DATE</code> , <code>TIME</code> , <code>TIMESTAMP</code>	Current transaction's start time <b>Note:</b> <code>NOW</code> is not the same as the <b><i>NOW</i></b> (see " <b><i>NOW [Date/Time]</i></b> " on page 220) function.
<code>TODAY</code>	<code>DATE</code> , <code>TIMESTAMP</code>	Midnight today
<code>TOMORROW</code>	<code>DATE</code> , <code>TIMESTAMP</code>	Midnight tomorrow
<code>YESTERDAY</code>	<code>DATE</code> , <code>TIMESTAMP</code>	Midnight yesterday
<code>ALLBALLS</code>	<code>TIME</code>	00:00:00.00 UTC

The following SQL-compatible functions can also be used to obtain the current time value for the corresponding data type:

- ***CURRENT\_DATE*** (page 190)
- ***CURRENT\_TIME*** (page 190)
- ***CURRENT\_TIMESTAMP*** (page 191)
- ***LOCALTIME*** (page 213)

- **LOCALTIMESTAMP** (page 214)

The latter four accept an optional precision specification. (See Date/Time Functions.) Note however that these are SQL functions and are not recognized as data input strings.

## NULL Value

NULL is a reserved keyword used to indicate that a data value is unknown.

Be very careful when using NULL in expressions. NULL is not greater than, less than, equal to, or not equal to any other expression. Use the **Boolean-predicate** (on page 52) for determining whether an expression value is NULL.

### Notes

- Vertica stores data in projections, which are sorted in a specific way. All columns are stored in ASC (ascending) order. For columns of data type NUMERIC, INTEGER, DATE, TIME, TIMESTAMP, and INTERVAL, NULL values are placed at the beginning of sorted projections (NULLS FIRST), while for columns of data type FLOAT, STRING, and BOOLEAN, NULL values are placed at the end (NULLS LAST). For details, see Null Placement in the Programmer's Guide.
- Vertica also accepts NUL characters ('\0') in constant strings and no longer removes null characters from VARCHAR fields on input or output. NUL is the ASCII abbreviation for the NULL character.
- You can write queries with expressions that contain the <=> operator for NULL=NULL joins. See Equi-joins and Non Equi-Joins in the Programmer's Guide.

### See Also

**NULL-handling Functions** (page 299)

## Numeric Expressions

Vertica follows the IEEE specification for floating point, including NaN.

A NaN is not greater than and at the same time not less than anything, even itself. In other words, comparisons always return false whenever a NaN is involved.

### Examples

```
=> SELECT CBRT('Nan'); -- cube root
      cbrt
-----
      NaN
(1 row)
=> SELECT 'Nan' > 1.0;
      ?column?
-----
      f
(1 row)
```

## Predicates

In general, predicates are truth-valued functions; that is, when invoked, they return a truth value. Predicates have a set of parameters and arguments. For example, in the following example `WHERE` clause:

```
WHERE name = 'Smith';
```

- `name = 'Smith'` is the predicate
- `'Smith'` is an expression

## BETWEEN-predicate

The special `BETWEEN` predicate is available as a convenience.

### Syntax

```
a BETWEEN x AND y
```

### Notes

```
a BETWEEN x AND y
```

Is equivalent to:

```
a >= x AND a <= y
```

Similarly:

```
a NOT BETWEEN x AND y
```

is equivalent to:

```
a < x OR a > y
```

## Boolean-predicate

Retrieves rows where the value of an expression is true, false, or unknown (null).

### Syntax

```
expression IS [NOT] TRUE  
expression IS [NOT] FALSE  
expression IS [NOT] UNKNOWN
```

### Notes

- A null input is treated as the value UNKNOWN.
- IS UNKNOWN and IS NOT UNKNOWN are effectively the same as the **NULL-predicate** (page 63), except that the input expression does not have to be a single column value. To check a single column value for NULL, use the NULL-predicate.
- Do not confuse the boolean-predicate with **Boolean Operators** (on page 37) or the **Boolean** (page 69) data type, which can have only two values: true and false.

## column-value-predicate

### Syntax

*column-name comparison-op constant-expression*

### Parameters

<i>column-name</i>	Is a single column of one the tables specified in the <b>FROM clause</b> (page 750).
<i>comparison-op</i>	Is one of the <b>comparison operators</b> (on page 37).
<i>constant-expression</i>	Is a constant value of the same data type as the <i>column-name</i> .

### Notes

To check a column value for `NULL`, use the ***NULL-predicate*** (page 63).

### Examples

```
table.column1 = 2
table.column2 = 'Seafood'
table.column3 IS NULL
```

## IN-predicate

### Syntax

```
column-expression [ NOT ] IN ( list-expression )
```

### Parameters

<i>column-expression</i>	One or more columns from the tables specified in the <b>FROM clause</b> (page 750).
<i>list-expression</i>	A comma-separated list of constant values matching the data type of the <i>column-expression</i>

### Examples

```
x, y IN ((1,2), (3, 4)), OR x, y IN (SELECT a, b FROM table)
x IN (5, 6, 7)
```

## INTERPOLATE

Used to join two event series together using some ordered attribute, event series joins let you compare values from two series directly, rather than having to normalize the series to the same measurement interval.

### Syntax

```
expression1 INTERPOLATE PREVIOUS VALUE expression2
```

### Parameters

<i>expression1</i> <i>expression2</i>	Is the <b>column-reference</b> (see " <b>Column References</b> " on page 46) from one the tables specified in the <b>FROM clause</b> (page 750). The column-reference can be any data type, but DATE/TIME types are the most useful, especially TIMESTAMP, since you are joining data that represents an event series.
PREVIOUS VALUE	Pads the non-preserved side with the previous values from relation when there is no match. Input rows are sorted in ascending logical order of the join column. <b>Note:</b> An ORDER BY clause, if used, does not determine the input order but only determines query output order.

### Notes

- An event series join is an extension of a regular outer join. Instead of padding the non-preserved side with null values when there is no match, the event series join pads the non-preserved side with the previous values from the table.
- The difference between expressing a regular outer join and an event series join is the INTERPOLATE predicate, which is used in the ON clause. See the **Examples** section below Notes and Restrictions. See also Event Series Joins in the Programmer's Guide.

- Data is logically partitioned on the table in which it resides, based on other ON clause equality predicates.
- Interpolated values come from the table that contains the null, not from the other table.
- Vertica does not guarantee that there will be no null values in the output. If there is no previous value for a mismatched row, that row will be padded with nulls.
- Event series join requires that both tables be sorted on columns in equality predicates, in any order, followed by the INTERPOLATED column. If data is already sorted in this order, then an explicit sort is avoided, which can improve query performance. For example, given the following tables:

```
ask: exchange, stock, ts, price
```

```
bid: exchange, stock, ts, price
```

In the query that follows

- ask is sorted on exchange, stock (or the reverse), ts
- bid is sorted on exchange, stock (or the reverse), ts

```
SELECT ask.price - bid.price, ask.ts, ask.stock, ask.exchange
FROM ask FULL OUTER JOIN bid
ON ask.stock = bid.stock AND ask.exchange = bid.exchange
AND ask.ts INTERPOLATE PREVIOUS VALUE bid.ts;
```

## Restrictions

- Only one INTERPOLATE expression is allowed per join.
- INTERPOLATE expressions are used only with ANSI SQL-99 syntax (the ON clause), which is already true for full outer joins.
- INTERPOLATE can be used with equality predicates only.
- The AND operator is supported but not the OR and NOT operators.
- Expressions and implicit or explicit casts are not supported, but subqueries are allowed.

## Example

The examples that follow use this simple schema.

```
CREATE TABLE t(x TIME);
CREATE TABLE t1(y TIME);
INSERT INTO t VALUES('12:40:23');
INSERT INTO t VALUES('14:40:25');
INSERT INTO t VALUES('14:45:00');
INSERT INTO t VALUES('14:49:55');
INSERT INTO t1 VALUES('12:40:23');
INSERT INTO t1 VALUES('14:00:00');
COMMIT;
```

## Normal full outer join

```
=> SELECT * FROM t FULL OUTER JOIN t1 ON t.x = t1.y;
```

Notice the null rows from the non-preserved table:

x	y
12:40:23	12:40:23
14:40:25	

```
14:45:00 |
14:49:55 |
          | 14:00:00
(5 rows)
```

### Full outer join with interpolation

=> SELECT \* FROM t FULL OUTER JOIN t1 ON t.x INTERPOLATE PREVIOUS VALUE t1.y;

In this case, the rows with no entry point are padded with values from the previous row.

```
   x      |      y
-----+-----
12:40:23 | 12:40:23
12:40:23 | 14:00:00
14:40:25 | 14:00:00
14:45:00 | 14:00:00
14:49:55 | 14:00:00
(5 rows)
```

### Normal Left Outer Join

=> SELECT \* FROM t LEFT OUTER JOIN t1 ON t.x = t1.y;

Again, there are nulls in the non-preserved table

```
   x      |      y
-----+-----
12:40:23 | 12:40:23
14:40:25 |
14:45:00 |
14:49:55 |
(4 rows)
```

### Left Outer Join with Interpolation

=> SELECT \* FROM t LEFT OUTER JOIN t1 ON t.x INTERPOLATE PREVIOUS VALUE t1.y;

Nulls padded with interpolated values.

```
   x      |      y
-----+-----
12:40:23 | 12:40:23
14:40:25 | 14:00:00
14:45:00 | 14:00:00
14:49:55 | 14:00:00
(4 rows)
```

### Inner joins

For inner joins, there is no difference between a regular inner join and an event series inner join. Since null values are eliminated from the result set, there is nothing to interpolate.

A regular inner join returns only the single matching row at 12:40:23:

=> SELECT \* FROM t INNER JOIN t1 ON t.x = t1.y;

```
   x      |      y
-----+-----
12:40:23 | 12:40:23
```

(1 row)

An event series inner join finds the same single-matching row at 12:40:23:

```
=> SELECT * FROM t INNER JOIN t1 ON t.x INTERPOLATE PREVIOUS VALUE t1.y;
```

```

      x      |      y
-----+-----
 12:40:23 | 12:40:23
(1 row)
```

## Semantics

When you write an event series join in place of normal join, values are evaluated as follows (using the schema in the above examples):

- `t` is the outer, preserved table
- `t1` is the inner, non-preserved table
- For each row in outer table `t`, the ON clause predicates are evaluated for each combination of each row in the inner table `t1`.
- If the ON clause predicates evaluate to true for any combination of rows, those combination rows are produced at the output.
- If the ON clause is false for all combinations, a single output row is produced with the values of the row from `t` along with the columns of `t1` chosen from the row in `t1` with the greatest `t1.y` value such that `t1.y < t.x`; if no such row is found, pad with nulls.

**Note:** `t LEFT OUTER JOIN t1` is equivalent to `t1 RIGHT OUTER JOIN t`.

In the case of a full outer join, all values from both tables are preserved.

## See Also

Event Series Joins in the Programmer's Guide

## join-predicate

Combines records from two or more tables in a database.

### Syntax

*column-reference* (see "Column References" on page 46) = *column-reference*

### Parameters

<i>column-reference</i>	Refers to a column of one the tables specified in the <b>FROM clause</b> (page 750).
-------------------------	--

## LIKE-predicate

Retrieves rows where the string value of a column matches a specified pattern. The pattern can contain one or more wildcard characters. `ILIKE` is equivalent to `LIKE` except that the match is case-insensitive (non-standard extension).

### Syntax

```
string [ NOT ]{ LIKE | ILIKE | LIKEB | ILIKEB }
... pattern [ESCAPE 'escape-character' ]
```

### Parameters

<i>string</i>	(CHAR, VARCHAR, BINARY, VARBINARY) is the column value to be compared to the <i>pattern</i> .
NOT	Returns true if <code>LIKE</code> returns false, and the reverse; equivalent to <code>NOT string LIKE pattern</code> .
<i>pattern</i>	Specifies a string containing wildcard characters. <ul style="list-style-type: none"> <li>▪ Underscore (<code>_</code>) matches any single character.</li> <li>▪ Percent sign (<code>%</code>) matches any string of zero or more characters.</li> </ul>
ESCAPE	Specifies an <i>escape-character</i> . An <code>ESCAPE</code> character can be used to escape itself, underscore ( <code>_</code> ), and <code>%</code> only. This is enforced only for non-default collations.  To match the <code>ESCAPE</code> character itself, use two consecutive escape characters. The default <code>ESCAPE</code> character is the backslash ( <code>\</code> ) character, although standard SQL specifies no default <code>ESCAPE</code> character. <code>ESCAPE</code> works for char and varchar strings only.
<i>escape-character</i>	Causes character to be treated as a literal, rather than a wildcard, when preceding an underscore or percent sign character in the <i>pattern</i> .

### Notes

- The `LIKE` predicate is compliant with the SQL standard.
- In the default locale, `LIKE` and `ILIKE` handle UTF-8 character-at-a-time, locale-insensitive comparisons. `ILIKE` handles language-independent case-folding.

**Note:** In non-default locales, `LIKE` and `ILIKE` do locale-sensitive string comparisons, including some automatic normalization, using the same algorithm as the "=" operator on `VARCHAR` types.

- The `LIKEB` and `ILIKEB` predicates do byte-at-a-time ASCII comparisons, providing access to Vertica 4.0 functionality.
- `LIKE` and `ILIKE` are stable for character strings, but immutable for binary strings, while `LIKEB` and `ILIKEB` are both immutable
- For `collation=binary` settings, the behavior is similar to Vertica 4.0. For other collations, `LIKE` operates on UTF-8 character strings, with the exact behavior dependent on collation parameters, such as strength. In particular, `ILIKE` works by setting `S=2` (ignore case) in the current session locale. See Locale Specification in the Administrator's Guide.

- Although the SQL standard specifies no default `ESCAPE` character, in Vertica the default is the backslash (`\`) and works for `CHAR` and `VARCHAR` strings only.

**Tip:** Vertica recommends that you specify an explicit escape character in all cases, to avoid problems should this behavior change. To use a backslash character as a literal, either specify a different escape character or use two backslashes.

- `ESCAPE` expressions evaluate to exactly one octet — or one UTF-8 character for non-default locales.
- An `ESCAPE` character can be used only to escape itself, `_`, and `%`. This is enforced only for non-default collations.
- `LIKE` requires that the entire string expression match the pattern. To match a sequence of characters anywhere within a string, the pattern must start and end with a percent sign.
- The `LIKE` predicate does not ignore trailing "white space" characters. If the data values that you want to match have unknown numbers of trailing spaces, tabs, etc., terminate each `LIKE` predicate pattern with the percent sign wildcard character.
- To use binary data types, you must use a valid binary character as the escape character, since backslash is not a valid `BINARY` character.
- The following symbols are substitutes for the actual keywords:

```

~~      LIKE
~#      LIKEB
~~*     ILIKE
~#*     ILIKEB
!~~     NOT LIKE
!~#     NOT LIKEB
!~~*    NOT ILIKE
!~#*    NOT ILIKEB

```

The `ESCAPE` keyword is not valid for the above symbols.

- Vertica extends support for single-row subqueries as the pattern argument for `LIKEB` and `ILIKEB`; for example:

```
SELECT * FROM t1 WHERE t1.x LIKEB (SELECT MAX (t2.a) FROM t2);
```

## Querying Case-sensitive data in System Tables

The `V_CATALOG.TABLES` (page 820).`TABLE_SCHEMA` and `TABLE_NAME` columns are case sensitive when used with an equality (`=`) predicate in queries. For example, given the following sample schema, if you execute a query using the `=` predicate, Vertica returns 0 rows:

```

=> CREATE SCHEMA SS;
=> CREATE TABLE SS.TT (c1 int);
=> INSERT INTO ss.tt VALUES (1);
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema = 'ss';

```

```

table_schema | table_name
-----+-----
(0 rows)

```

**TIP:** Use the case-insensitive `ILIKE` predicate to return the expected results.

```

=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema ILIKE 'ss';

```

```

table_schema | table_name
-----+-----
SS           | TT
(1 row)

```

## Examples

```

'abc' LIKE 'abc' true
'abc' LIKE 'a%' true
'abc' LIKE '_b_' true
'abc' LIKE 'c' false
'abc' LIKE 'ABC' false
'abc' ILIKE 'ABC' true
'abc' not like 'abc' false
not 'abc' like 'abc' false

```

The following example illustrates pattern matching in locales.

```

\locale default
=> CREATE TABLE src(c1 VARCHAR(100));
=> INSERT INTO src VALUES (U&'\00DF'); --The beta (ß)
=> INSERT INTO src VALUES ('ss');
=> COMMIT;

```

Querying the `src` table in the default locale returns both `ss` and `beta`.

```

=> SELECT * FROM src;
 c1
-----
ß
ss
(2 rows)

```

The following query combines pattern-matching predicates to return the results from column `c1`:

```

=> SELECT c1, c1 = 'ss' AS equality, c1 LIKE 'ss' AS LIKE, c1
       ILIKE 'ss' AS ILIKE FROM src;
 c1 | equality | LIKE | ILIKE
-----+-----+-----+-----
ß  | f       | f    | f
ss | t       | t    | t
(2 rows)

```

The next query specifies unicode format for `c1`:

```

=> SELECT c1, c1 = U&'\00DF' AS equality, c1 LIKE U&'\00DF' AS LIKE,
       c1 ILIKE U&'\00DF' AS ILIKE from src;
 c1 | equality | LIKE | ILIKE
-----+-----+-----+-----
ß  | t       | t    | t
ss | f       | f    | f
(2 rows)

```

Now change the locale to German with a strength of 1 (ignore case and accents):

```

\locale LDE_S1
=> SELECT c1, c1 = 'ss' AS equality, c1 LIKE 'ss' as LIKE,
       c1 ILIKE 'ss' AS ILIKE from src;
 c1 | equality | LIKE | ILIKE

```

```

-----+-----+-----+-----
ß | t          | t          | f
ss | t          | t          | t
(2 rows)

```

The following query fails because `ILIKE` forces collation into `S2`. Because the locale is `S1`, the ignore-accents part is lost and the beta is considered an accent:

```

=> SELECT c1, c1 = U&'\00DF' AS equality, c1 LIKE U&'\00DF' AS LIKE,
       c1 ILIKE U&'\00DF' AS ILIKE from src;
 c1 | equality | LIKE | ILIKE
-----+-----+-----+-----
ss | t          | t          | f
ß  | t          | t          | t
(2 rows)

```

This example illustrates binary data types with pattern-matching predicates:

```

=> CREATE TABLE t (c BINARY(1));
=> INSERT INTO t values(HEX_TO_BINARY('0x00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFF'));
=> SELECT TO_HEX(c) from t;
 TO_HEX
-----
 00
 ff
(2 rows)
select * from t;
 c
-----
 \000
 \377
(2 rows)

=> SELECT c, c = '\000', c LIKE '\000', c ILIKE '\000' from t;

 c | ?column? | ?column? | ?column?
-----+-----+-----+-----
 \000 | t          | t          | t
 \377 | f          | f          | f
(2 rows)
=> SELECT c, c = '\377', c LIKE '\377', c ILIKE '\377' from t;
 c | ?column? | ?column? | ?column?
-----+-----+-----+-----
 \000 | f          | f          | f
 \377 | t          | t          | t
(2 rows)

```

## NULL-predicate

Tests for null values.

### Syntax

```
column-name IS [ NOT ] NULL
```

### Parameters

<i>column-name</i>	Is a single column of one the tables specified in the <b>FROM clause</b> (page 750).
--------------------	--

### Examples

```
a IS NULL  
b IS NOT NULL
```

### See Also

**NULL Value** (page 50)

# SQL Data Types

---

The following tables summarizes the data types supported by Vertica, as well as the default placement of null values in projections. The Size column is shown in uncompressed bytes.

Type	Size	Description	NULL Sorting
<b>Binary types</b>			
BINARY	1 to 65000	Fixed-length binary string	NULLS LAST
VARBINARY	1 to 65000	Variable-length binary string	NULLS LAST
BYTEA	1 to 65000	Variable-length binary string (synonym for VARBINARY)	NULLS LAST
RAW	1 to 65000	Variable-length binary string (synonym for VARBINARY)	NULLS LAST
<b>Boolean types</b>			
BOOLEAN	1	True or False or NULL	NULLS LAST
<b>Character types</b>			
CHAR	1 to 65000	Fixed-length character string	NULLS LAST
VARCHAR	1 to 65000	Variable-length character string	NULLS LAST
<b>Date/time types</b>			
DATE	8	Represents a month, day, and year	NULLS FIRST
DATETIME	8	Represents a date and time with or without timezone (synonym for TIMESTAMP)	NULLS FIRST
SMALLDATETIME	8	Represents a date and time with or without timezone (synonym for TIMESTAMP)	NULLS FIRST
TIME	8	Represents a time of day without timezone	NULLS FIRST
TIME WITH TIMEZONE	8	Represents a time of day with timezone	NULLS FIRST
TIMESTAMP	8	Represents a date and time without timezone	NULLS FIRST
TIMESTAMP WITH TIMEZONE	8	Represents a date and time with timezone	NULLS FIRST
INTERVAL	8	Measures the difference between two points in time	NULLS FIRST

<b>Approximate numeric types</b>			
DOUBLE PRECISION	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
FLOAT	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
FLOAT (n)	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
FLOAT8	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
REAL	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
<b>Exact numeric types</b>			
INTEGER	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
INT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
BIGINT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
INT8	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
SMALLINT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
TINYINT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
DECIMAL	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST
NUMERIC	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST
NUMBER	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST
MONEY	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST

## Binary Data Types

Store raw-byte data, such as IP addresses, up to 65000 bytes.

## Syntax

```
BINARY ( length )
{ VARBINARY | BINARY VARYING | BYTEA | RAW } ( max-length )
```

## Parameters

<i>length</i>   <i>max-length</i>	Specifies the length of the string.
-----------------------------------	-------------------------------------

## Notes

- The data types `BINARY` and `BINARY VARYING` (`VARBINARY`) are collectively referred to as *binary string types* and the values of binary string types are referred to as *binary strings*.
  - A binary string is a sequence of octets, or bytes. Binary strings store raw-byte data, while character strings store text.
  - The binary data types, `BINARY` and `VARBINARY`, are similar to the **character data types** (page 70), `CHAR` and `VARCHAR`, respectively, except that binary data types contain byte strings, rather than character strings. The allowable maximum length is the same for binary data types as it is for character data types, except that the length for `BINARY` and `VARBINARY` is a length in bytes, rather than in characters.
  - **BINARY** — A fixed-width string of *length* bytes, where the number of bytes is declared as an optional specifier to the type. If *length* is omitted, the default is 1. Where necessary, values are right-extended to the full width of the column with the zero byte. For example:
 

```
=> SELECT TO_HEX('ab'::BINARY(4));
       to_hex
       -----
       61620000
```
  - **VARBINARY** — A variable-width string up to a length of *max-length* bytes, where the maximum number of bytes is declared as an optional specifier to the type. The default is the default attribute size, which is 80, and the maximum length is 65000 bytes. Varbinary values are not extended to the full width of the column. For example:
 

```
=> SELECT TO_HEX('ab'::VARBINARY(4));
       to_hex
       -----
       6162
```
  - `BYTEA` and `RAW` are synonyms for `VARBINARY`.
  - You can use several formats when working with binary values, but the hexadecimal format is generally the most straightforward and is emphasized in Vertica documentation.
  - The `&`, `~`, `|` and `#` binary operands have special behavior for binary data types, as described in **Binary Operators** (page 35).
  - On input, strings are translated from hexadecimal representation to a binary value using the `HEX_TO_BINARY` (page 324) function. Strings are translated from bitstring representation to binary values using the `BITSTRING_TO_BINARY` (page 316) function. Both functions take a `VARCHAR` argument and return a `VARBINARY` value. See the Examples section below.
- Binary values can also be represented in octal format by prefixing the value with a backslash `'\'`.

**Note:** If you use vsql, you must use the escape character (\) when inserting another backslash on input; for example, input '\141' as '\\141'.

You can also input values represented by printable characters. For example, the hexadecimal value '0x61' can also be represented by the symbol 'a'.

See Loading Binary Data in the Administrator's Guide.

- Like the input format the output format is a hybrid of octal codes and printable ASCII characters. A byte in the range of printable ASCII characters (the range [0x20, 0x7e]) is represented by the corresponding ASCII character, with the exception of the backslash ('\'), which is escaped as '\\'. All other byte values are represented by their corresponding octal values. For example, the bytes {97,92,98,99}, which in ASCII are {a, \, b, c}, are translated to text as 'a\\bc'.
- The following aggregate functions are supported for binary data types:
  - BIT\_AND (page 181)
  - **BIT\_OR** (page 182)
  - **BIT\_XOR** (page 183)
  - **MAX** (page 116)
  - **MIN** (page 117)

BIT\_AND, BIT\_OR, and BIT\_XOR are bitwise operations that are applied to each non-null value in a group, while MAX and MIN are bitwise comparisons of binary values.

- Like their **binary operator** (page 35) counterparts, if the values in a group vary in length, the aggregate functions treat the values as though they are all equal in length by extending shorter values with zero bytes to the full width of the column. For example, given a group containing the values 'ff', null, and 'f', a binary aggregate ignores the null value and treats the value 'f' as 'f0'. Also, like their binary operator counterparts, these aggregate functions operate on VARBINARY types explicitly and operate on BINARY types implicitly through casts. See **Data Type Coercion Operators (CAST)** (page 38).

## Examples

The following example shows VARBINARY HEX\_TO\_BINARY (page 324) (VARCHAR) and VARCHAR TO\_HEX (page 244) (VARBINARY) usage.

Table t and its projection are created with binary columns:

```
=> CREATE TABLE t (c BINARY(1));
=> CREATE PROJECTION t_p (c) AS SELECT c FROM t;
```

Insert minimum byte and maximum byte values:

```
=> INSERT INTO t values(HEX_TO_BINARY('0x00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFF'));
```

Binary values can then be formatted in hex on output using the TO\_HEX function:

```
=> SELECT TO_HEX(c) FROM t;
to_hex
-----
00
ff
(2 rows)
```

The `BIT_AND`, `BIT_OR`, and `BIT_XOR` functions are interesting when operating on a group of values. For example, create a sample table and projections with binary columns:

This examples uses the following schema, which creates table `t` with a single column of `VARBINARY` data type:

```
=> CREATE TABLE t (  
    c VARBINARY(2) );  
=> INSERT INTO t values(HEX_TO_BINARY('0xFF00'));  
=> INSERT INTO t values(HEX_TO_BINARY('0xFFFF'));  
=> INSERT INTO t values(HEX_TO_BINARY('0xF00F'));
```

Query table `t` to see column `c` output:

```
=> SELECT TO_HEX(c) FROM t;  
TO_HEX  
-----  
ff00  
ffff  
f00f  
(3 rows)
```

Now issue the bitwise AND operation. Because these are aggregate functions, an implicit `GROUP BY` operation is performed on results using `(ff00&(ffff)&f00f)`:

```
=> SELECT TO_HEX(BIT_AND(c)) FROM t;  
to_hex  
-----  
f000  
(1 row)
```

Issue the bitwise OR operation on `(ff00|(ffff)|f00f)`:

```
=> SELECT TO_HEX(BIT_OR(c)) FROM t;  
to_hex  
-----  
ffff  
(1 row)
```

Issue the bitwise XOR operation on `(ff00#(ffff)#f00f)`:

```
=> SELECT TO_HEX(BIT_XOR(c)) FROM t;  
to_hex  
-----  
f0f0  
(1 row)
```

### See Also

Aggregate functions ***BIT\_AND*** (page 181), ***BIT\_OR*** (page 182), ***BIT\_XOR*** (page 183), ***MAX*** (page 116), and ***MIN*** (page 117)

***Binary Operators*** (page 35)

***COPY*** (page 607)

**Data Type Coercion Operators (CAST)** (page 38)

IP conversion function **INET\_ATON** (page 272), **INET\_NTOA** (page 273), **V6\_ATON** (page 273), **V6\_NTOA** (page 275), **V6\_SUBNETA** (page 275), **V6\_SUBNETN** (page 276), **V6\_TYPE** (page 277)

String functions **BITCOUNT** (page 316), **BITSTRING\_TO\_BINARY** (page 316), **HEX\_TO\_BINARY** (page 324), **LENGTH** (page 335), **REPEAT** (page 345), **SUBSTRING** (page 355), **TO\_HEX** (page 244), and **TO\_BITSTRING** (page 241)

Loading Binary Data in the Administrator's Guide

## Boolean Data Type

Vertica provides the standard SQL type BOOLEAN, which has two states: true and false. The third state in SQL boolean logic is unknown, which is represented by the NULL value.

### Syntax

BOOLEAN

### Parameters

Valid literal data values for input are:

TRUE	't'	'true'	'y'	'yes'	'1'
FALSE	'f'	'false'	'n'	'no'	'0'

### Notes

- Do not confuse the `BOOLEAN` data type with **Boolean Operators** (on page 37) or the **Boolean-predicate** (on page 52).
- The keywords `TRUE` and `FALSE` are preferred and are SQL-compliant.
- All other values must be enclosed in single quotes.
- Boolean values are output using the letters t and f.

### See Also

**NULL Value** (page 50)

## Character Data Types

Stores strings of letters, numbers and symbols. Character data can be stored as fixed-length or variable-length strings; the difference is that fixed-length strings are right-extended with spaces on output, and variable-length strings are not extended.

### Syntax

```
[ CHARACTER | CHAR ] ( length )
[ VARCHAR | CHARACTER VARYING ] ( length )
```

### Parameters

<i>length</i>	Specifies the length of the string in octets.
---------------	---

### Notes

- A character is a Unicode codepoint represented as UTF-8.
- The data types `CHARACTER (CHAR)` and `CHARACTER VARYING (VARCHAR)` are collectively referred to as *character string types*, and the values of character string types are known as *character strings*.
- `CHAR` is conceptually a fixed-length, blank padded string. Any trailing blanks (spaces) are removed on input, and only restored on output. The default length is 1 and the maximum length is 65000 octets (bytes).
- `VARCHAR` is a variable-length character data type. The default length is 80 and the maximum length is 65000 octets. Values can include trailing spaces.
- When you define character columns, you specify the maximum size of any string to be stored in the column. For example, if you want to store strings up to 24 octets in length, you could use either of the following definitions:

```
CHAR(24)      /* fixed-length */
VARCHAR(24)   /* variable-length */
```

- The maximum length parameter for `VARCHAR` and `CHAR` data type refers to the number of octets that can be stored in that field and not number of characters. When using multibyte UTF-8 characters, the fields must be sized to accommodate from 1 to 4 octets per character, depending on the data. If the data being loaded into a `VARCHAR/CHAR` column exceeds the specified maximum size for that column, data is truncated on UTF-8 character boundaries to fit within the specified size. See `COPY` (page 607).

**Note:** Remember to include the extra octets required for multibyte characters in the column-width declaration, keeping in mind the 65000 octet column-width limit.

- String literals in SQL statements must be enclosed in single quotes.
- Due to compression in Vertica, the cost of over-estimating the length of these fields is incurred primarily at load time and during sorts.
- NULL appears last (largest) in ascending order. See also **GROUP BY Clause** (page 754) for additional information about null ordering.

**NULL vs NUL**

NUL represents a character whose ASCII/Unicode code is zero, sometimes qualified "ASCII NUL".

NULL means no value, and is true of a field (column) or constant, not of a character.

VARCHAR string data types accept ASCII NULs.

The following example casts the input string containing NUL values to VARCHAR:

```
=> SELECT E'vert\0ica'::CHARACTER VARYING AS varchar;
   varchar
-----
   vertica
(1 row)
```

The following example casts the input string containing NUL values to VARBINARY:

```
=> SELECT E'vert\0ica'::BINARY VARYING as varbinary;
   varbinary
-----
   vert\000ica
(1 row)
```

In both cases, the result contains 8 characters, but in the VARCHAR case, the '\000' is not visible:

```
=> SELECT LENGTH('vert\0ica'::CHARACTER VARYING);
   length
-----
         8
(1 row)
=> SELECT LENGTH('vert\0ica'::BINARY VARYING);
   length
-----
         8
(1 row)
```

**See Also**

***Data Type Coercion*** (page 104)

## Date/Time Data Types

Vertica supports the full set of SQL date and time data types. In most cases, a combination of `DATE`, `DATETIME`, `SMALLDATETIME`, `TIME`, `TIMESTAMP WITHOUT TIME ZONE`, and `TIMESTAMP WITH TIME ZONE`, and `INTERVAL` provides a complete range of date/time functionality required by any application.

In compliance with the SQL standard, Vertica also supports the `TIME WITH TIME ZONE` data type.

The following table lists the date/time data types, their sizes, values, and resolution.

### Date/Time Data Types

Name	Size	Description	Low Value	High Value	Resolution
<code>DATE</code>	8 bytes	Dates only (no time of day)	4713 BC	5874897 AD	1 day
<code>DATETIME</code>	8 bytes	Both date and time, with [w/o] time zone	4713 BC	5874897 AD	1 microsecond/14 digits
<code>INTERVAL [(p)]</code>	8 bytes	Time intervals	-178000000 yrs	178000000 yrs	1 microsecond/14 digits
<code>SMALLDATETIME</code>	8 bytes	Both date and time, with [w/o] time zone	4713 BC	5874897 AD	1 microsecond/14 digits
<code>TIME [(p)] [WITHOUT TIME ZONE]</code>	8 bytes	Times of day only (no date)	00:00:00.00	23:59:59.99	1 microsecond/14 digits
<code>TIME [(p)] WITH TIME ZONE</code>	8 bytes	Times of day only, with time zone	00:00:00.00+12	23:59:59.99-12	1 microsecond/14 digits
<code>TIMESTAMP [(p)] [{ WITH   WITHOUT} TIME ZONE]   TIMESTAMPTZ</code>	8 bytes	Both date and time, with [w/o] time zone	4713 BC	5874897 AD	1 microsecond/14 digits

### Time Zone Abbreviations for Input

The files in `/opt/vertica/share/timezonesets` are recognized by Vertica as date/time input values and define the default list of strings accepted in the `AT TIME ZONE zone` parameter. The names are not necessarily used for date/time output — output is driven by the official time zone abbreviations associated with the currently selected time zone parameter setting.

### Notes

- In Vertica, `TIME ZONE` is a synonym for `TIMEZONE`.
- Vertica uses Julian dates for all date/time calculations. They can correctly predict and calculate any date more recent than 4713 BC to far into the future, based on the assumption that the length of the year is 365.2425 days.
- All date/time types are stored in eight bytes.
- A date/time value of `NULL` appears first (smallest) in ascending order.

- All the date/time data types accept the special literal value `NOW` to specify the current date and time. For example:

```
=> SELECT TIMESTAMP 'NOW';
      ?column?
```

```
-----
2010-10-04 11:18:15.227544
(1 row)
```

- In Vertica, The `INTERVALS` (page 74) data type is SQL-2008 compliant and allows modifiers, called **interval qualifiers** (page 34), that divide the `INTERVAL` type into two primary subtypes, `DAY TO SECOND` (the default) and `YEAR TO MONTH`. You use the `SET INTERVALSTYLE` (page 768) command to change the run-time parameter for the current session.

Intervals are represented internally as some number of microseconds and printed as up to 60 seconds, 60 minutes, 24 hours, 30 days, 12 months, and as many years as necessary. Fields can be positive or negative.

### See Also

Set the Default Time Zone and Using Time Zones with Vertica in the Installation Guide

**Sources for Time Zone and Daylight Saving Time Data**  
<http://www.twinsun.com/tz/tz-link.htm>

## DATE

Consists of a month, day, and year.

### Syntax

DATE

### Parameters

Low Value	High Value	Resolution
4713 BC	32767 AD	1 DAY

See **SET DATESTYLE** (page 767) for information about ordering.

Example	Description
January 8, 1999	Unambiguous in any <code>datestyle</code> input mode
1999-01-08	ISO 8601; January 8 in any mode (recommended format)
1/8/1999	January 8 in <code>MDY</code> mode; August 1 in <code>DMY</code> mode
1/18/1999	January 18 in <code>MDY</code> mode; rejected in other modes
01/02/03	January 2, 2003 in <code>MDY</code> mode February 1, 2003 in <code>DMY</code> mode February 3, 2001 in <code>YMD</code> mode
1999-Jan-08	January 8 in any mode

Jan-08-1999	January 8 in any mode
08-Jan-1999	January 8 in any mode
99-Jan-08	January 8 in YMD mode, else error
08-Jan-99	January 8, except error in YMD mode
Jan-08-99	January 8, except error in YMD mode
19990108	ISO 8601; January 8, 1999 in any mode
990108	ISO 8601; January 8, 1999 in any mode
1999.008	Year and day of year
J2451187	Julian day
January 8, 99 BC	Year 99 before the Common Era

## DATETIME

DATETIME is an alias for **TIMESTAMP** (page 90).

## INTERVAL

Measures the difference between two points in time. The INTERVAL data type is divided into two major subtypes: DAY TO SECOND (day/time, kept in microseconds) and YEAR TO MONTH (year/month, kept in months). A day/time interval represents a span of days, hours, minutes, seconds, and fractional seconds. A year/month interval represents a span of years and months. The default *interval-qualifier*, if not specified, is INTERVAL DAY TO SECOND (6). Intervals can be positive or negative.

You can specify optional date/time units on interval input and output. The sections that follow describe each of the three methods available to you:

- The SQL-compliant implementation: no units on output (the default)
- The Vertica extension: optional units on output
- Optional units on input

You change the INTERVAL style with the **SET INTERVALSTYLE** (page 768) command. If you get unexpected results, issue the **SHOW** (page 785) command to display the run-time parameters.

### Syntax

```
INTERVAL [ (p) ] [ - 'interval-literal (on page 32)' ] [ interval-qualifier (on page 34) ]
```

### Parameters

<i>p</i>	(Precision) can specify the number of fractional digits retained in the seconds field, in the range 0 to 6. The default is 6.
<i>interval-literal</i>	[Optional] A literal character string expressing a specific interval.

	Sometimes referred to as <i>data</i> in this topic.
interval-qualifier	Specifies a range of interval subtypes with optional precision specifications. If omitted, the default is DAY TO SECOND (6). Sometimes referred to as <i>subtype</i> in this topic.

### No units on output

The default style is PLAIN (no units on output) and follows the SQL-2008 standard:

```
=> SELECT INTERVAL '3 2';
?column?
-----
3 02:00
(1 row)
```

Note that the following command returns the same result, even though units are specified inside the interval-literal. Those units are omitted from the result:

```
=> SELECT INTERVAL '3 days 2 hours';
?column?
-----
3 02:00
(1 row)
```

The same interval (3 days, 2 hours) can be expressed in several ways in SQL-2008. For example, if you issue the command SET DATESTYLE to SQL the output matches INTERVALSTYLE PLAIN (no units); thus, all of the following commands return 3 02:

```
=> SELECT INTERVAL '3' DAY + INTERVAL '2' HOUR;
=> SELECT INTERVAL '3 2' DAY TO HOUR;
=> SELECT INTERVAL '3 days 2 hours' DAY TO HOUR;
=> SELECT INTERVAL '3 days 2' DAY TO HOUR;
```

The following example extracts the HOUR value from the input string:

```
=> SELECT INTERVAL '28 days 3 hours' HOUR;
?column?
-----
675
(1 row)
```

In the next example, HOUR(2) instructs Vertica to use up to 2 places to output hours, but note that Vertica uses as many entries as needed, so the (2) specification is ignored. Note also that Vertica ignores spaces; for example HOUR(2) is processed the same as HOUR (2).

```
=> SELECT INTERVAL '28 days 3 hours' HOUR (2);
?column?
-----
675
(1 row)
```

If seconds contain decimal places, they are rounded on output to the precision you specify; for example INTERVAL (3) in the following command:

```
=> SELECT INTERVAL(3) '28 days 3 hours 1.234567 sec';
?column?
```

```
-----
28 03:00:01.235
(1 row)
```

Vertica ignores a precision placed on a unit specified inside an interval-literal:

```
=> SELECT INTERVAL '28 days 3 hours 1.234567 sec(3)';
      ?column?
```

```
-----
28 03:03:01.234567
(1 row)
```

If you move the precision outside of the interval-literal, Vertica honors it:

```
=> SELECT INTERVAL '28 days 3 hours 1.234567)' second(3);
      ?column?
```

```
-----
2430001.235
(1 row)
```

If there are two different specifiers, Vertica picks the lesser of the two for seconds. For example, in the following command, Vertica picks (1):

```
=> SELECT INTERVAL(1) '1.2467' SECOND(2);
      ?column?
```

```
-----
1.2
(1 row)
```

Intervals can be cast *within* the day/time or the year/month subtypes but not between them. For example, the following command converts to DAY TO SECOND (the default):

```
=> SELECT CAST(INTERVAL '4440' MINUTE as INTERVAL);
      ?column?
```

```
-----
3 02:00
(1 row)
```

```
=> SELECT CAST(INTERVAL '-01:15' as INTERVAL MINUTE);
      ?column?
```

```
-----
-75
(1 row)
```

The following query, however, returns an error:

```
=> SELECT INTERVAL '1 02:03:04.56' HOUR TO SECOND;
      ERROR:  invalid input syntax for type interval hour to second: "1 02:03:04.56"
```

The error is legitimate. For standalone fields without units, such as the first '1' in an interval-literal '1 02:03:04.56', the units are determined as the first not-already-matched subtype field. Thus, if the subtype range is HOUR TO SECOND, the first '1' is '1 hour' and conflicts with the '02' in the example, which is also an hour.

### Units on output

To enable interval units on output, issue the following command:

```
=> SET INTERVALSTYLE TO UNITS;
```

Units are now returned with the interval value 'days':

```
=> SELECT INTERVAL '3 days 2 hours';
      ?column?
-----
3 days 02:00
(1 row)
```

INTERVALSTYLE (page 768) and DATESTYLE (page 767) settings affect the interval output format only, not the interval input format. All interval output formats are accepted as input, independent of the current output format.

When units are enabled, their format is controlled by DATESTYLE (page 767). If you are expecting units on output but not seeing them, issue the SHOW DATESTYLE command. DATESTYLE must be set to ISO for INTERVAL to display units on output.

### Units on input

A Vertica extension lets you include units within the *interval-literal* (page 32). These units do not control or affect the declared subtype range, which is declared by the *interval-qualifier* (on page 34).

```
=> SELECT INTERVAL '3 days 2 hours';
      ?column?
-----
3 days 02:00
(1 row)
```

Using the same interval-literal from the previous example, the following command still specifies units as days and hours, but the interval-qualifier extracts minutes values from the inputs:

```
=> SELECT INTERVAL '3 days 2 hours' MINUTE;
      ?column?
-----
4440 mins
(1 row)
```

**Note:** Inside the single quotes of an interval-literal, units can be plural, but outside the quotes, the interval-qualifier must take the singular form.

Vertica allows combinations of units, such as second and millisecond together in an INTERVAL DAY TO SECOND (or HOUR TO SECOND) subtype; however, each unit can be used one time only in the interval-literal string. The follow commands shows some of the combinations of units that are allowed:

```
=> SELECT INTERVAL '1 second 1 millisecond' DAY TO SECOND;
      ?column?
-----
00:00:01.001
(1 row)

=> SELECT INTERVAL '12:13:14 15 microseconds' DAY TO SECOND;
      ?column?
-----
12:13:14.000015
(1 row)
```

```
=> SELECT INTERVAL '12:13:14.123 15 microseconds' DAY TO SECOND;
      ?column?
```

```
-----
12:13:14.123015
(1 row)
```

The following command, however, is rejected because there are two seconds fields:

```
=> SELECT INTERVAL '12:13:14 15 seconds' DAY TO SECOND;
      ERROR:  invalid input syntax for type interval: "12:13:14 15 seconds"
```

If you remove the `seconds` unit, the command returns the expected result of 15 days, 12 hours, 13 minutes, and 14 seconds:

```
=> SELECT INTERVAL '12:13:14 15' DAY TO SECOND;
      ?column?
```

```
-----
15 12:13:14
(1 row)
```

There are cases where the data (interval-literal) looks like a year/month type, but the type is day/second, and the reverse. Vertica reads interval-literal data from left to right, where number-number is years-months, and number <space> <signed number> is whatever the units specify.

The following command is processed as follows: (-) 1 year 1 month as (-)  $365 + 30 = -395$  days:

```
=> SELECT INTERVAL '-1-1' DAY TO HOUR;
      ?column?
```

```
-----
-395
(1 row)
```

The next command is processed as follows: (-) 1 day - 1 hour as (-)  $24 - 1 = -23$  hours:

```
=> SELECT INTERVAL '-1 -1' DAY TO HOUR;
      ?column?
```

```
-----
-23
(1 row)
```

The next command is processed as follows: (-) 1 year - 1 month as (-)  $365 - 30 = -335$  days

```
=> SELECT INTERVAL '-1--1' DAY TO HOUR;
      ?column?
```

```
-----
-335
(1 row)
```

The next command is processed as follows: 1 year 0 month -1 day as  $365 + 0 - 1 = -364$  days

```
=> SELECT INTERVAL '1- -1' DAY TO HOUR;
      ?column?
```

```
-----
364
(1 row)
```

In the following example, the inputs '1 4 5 6' returns 1 day, 4 hours, 5 minutes, 6 seconds:

```
=> SELECT INTERVAL '1 4 5 6';
```

```

?column?
-----
1 04:05:06
(1 row)

```

The following example shows the previous command with units turned on:

```

=> SELECT INTERVAL '1 4 5 6';
?column?
-----
1 day 04:05:06
(1 row)

```

In this example, the system recognizes the colon as being part of the timestamp and outputs 4 hours, 5 minutes, 6 seconds appropriately. When it reaches the 1, it knows it has already processed hours, minutes, and seconds and assigns the 1 value to the day field:

```

=> SELECT INTERVAL '4:5:6 1';
?column?
-----
1 04:05:06
(1 row)

```

You get the same results if you rewrite the command as follows:

```

=> SELECT INTERVAL '1 4:5:6';

```

In the next example, Vertica recognizes the 4:5 combination as hour/minute, so input value 1 is assigned to day and the final value 2 is assigned to seconds:

```

SELECT INTERVAL '4:5 1 2';
?column?
-----
1 04:05:02
(1 row)

```

You get the same results if you rewrite the command as follows:

```

=> SELECT INTERVAL '1 4:5 2';

```

If you reverse the 1 and the 2, the results change because of how Vertica processes the command:

```

=> SELECT INTERVAL '2 4:5 1';
?column?
-----
2 04:05:01
(1 row)

```

Units less than a month are invalid for YEAR TO MONTH intervals

```

=> SELECT INTERVAL '1 y 30 days' YEAR TO MONTH;
ERROR:  invalid input syntax for type interval year to month: "1 y 30 days"

```

If you replace the days interval-literal with an appropriate unit, for example one that represents months, Vertica returns the correct information of 1 year, 3 months:

```

=> SELECT INTERVAL '1 y 3 m' YEAR TO MONTH;
?column?
-----
1-3

```

(1 row)

Notice that `m` was used as the interval-literal in the previous example, representing months. If you specify a `DAY TO SECOND` interval-qualifier, Vertica knows that `m` represents minutes. The following command, for example, returns 1 day, 0 hours, and three minutes:

```
=> SELECT INTERVAL '1 d 3 m' DAY TO SECOND;
   ?column?
-----
1 00:03
(1 row)
```

The following series of examples use units in the input to return microseconds:

```
=> SELECT INTERVAL '4:5 1 2 34us';
   ?column?
-----
1 04:05:02.000034
(1 row)
=> SELECT INTERVAL '4:5 1d 2 34us' HOUR TO SECOND;
   ?column?
-----
28:05:02.000034
(1 row)
```

In the following example, `4:5` represents `min:sec`.

```
=> SELECT INTERVAL '4:5 1d 34us' MINUTE TO SECOND;
   ?column?
-----
1444:05.000034
(1 row)
```

The input unit '`m`' can represent either 'months' or 'minutes,' depending on context. To illustrate, the following command creates a one-column table with some interval values:

```
=> CREATE TABLE int_test(i INTERVAL YEAR TO MONTH);
```

In the first `INSERT` statement, the values are inserted as 1 year, six months:

```
=> INSERT INTO int_test VALUES('1 year 6 months');
```

In the second `INSERT` statement, the minutes value is ignored, as the `DAY TO SECOND` part is truncated:

```
=> INSERT INTO int_test VALUES('1 year 6 minutes');
```

In the third `INSERT` statement, the '`m`' counts as minutes value is ignored, as the `DAY TO SECOND` part is truncated:

```
=> INSERT INTO int_test VALUES('1 year 6 m'); -- the m counts as months
Query the table and you will notice that the second row does not contain the minutes input:
```

```
=> SELECT * FROM int_test;
   i
-----
1-6
1-0
1-6
```

(3 rows)

In the following command, the the 'm' counts as minutes, because the `DAY TO SECOND` interval-qualifier extracts day/time values from the input:

```
=> SELECT INTERVAL '1y6m' DAY TO SECOND;
   ?column?
-----
 365 00:06
(1 row)
```

## Notes

- The Vertica `INTERVAL` data type is SQL-2008 compliant, with extensions. It maintains compatibility with existing interval data. On Vertica databases created prior to version 4.0, all `INTERVAL` columns are interpreted as `INTERVAL DAY TO SECOND`, as in the previous release.
- On input, day/time intervals can be expressed as a combination of fields. Vertica converts these to microseconds, adds them together, and operates on the sum.
- An `INTERVAL` can include only the subset of units that you need; however, year/month intervals represent calendar years and months with no fixed fixed number of days, so year/month interval values cannot include days, hours, minutes. Similarly, day/time intervals cannot include year, month, and so on..
- Day/time and year/month intervals are logically independent and cannot be combined with or compared to one another. In the following example, data that contains `days` cannot be combined with the `YEAR TO MONTH` type.
- The primary day/time (`DAY TO SECOND`) and year/month (`YEAR TO MONTH`) subtype ranges can be restricted to more specific range of types by an interval-qualifier. For example, `HOURLY TO MINUTE` is a limited form of day/time interval, which can be used to express time zone offsets.
- Quantities of days, hours, minutes, and seconds can be specified without explicit unit markings. For example, '1 12:59:10' is read the same as '1 day 12 hours 59 minutes 10 seconds'.
- Vertica accepts intervals up to  $2^{63} - 1$  microseconds or months (about 18 digits).
- If an interval-qualifier is not specified, the default type is `DAY TO SECOND (6)`, no matter what data goes inside the quotes. For example, as an extension to SQL-2008, both of the following commands return 910 (days):

```
=> SELECT INTERVAL '2-6';
=> SELECT INTERVAL '2 years 6 months';
```

However, if you change the interval-qualifier to `YEAR TO MONTH`, as in the following command, the returned value is 2-6 for 2 years 6 months:

```
=> SELECT INTERVAL '2 years 6 months' YEAR TO MONTH;
```

- SQL-2008 allows both the leftmost units field and the `SECOND` units field to include a precision specification of up to 6 fractional second places, with rounding, if fewer digits are wanted. When `SECOND` is not the first field, it has a precision of 2 places before the decimal point.

The following command specifies that the day field can hold 4 digits, the hour field 2 digits, the minutes field 2 digits, the seconds field 2 digits, and the fractional seconds field 6 digits:

```
=> SELECT INTERVAL '1000 12:00:01.123456' DAY(4) TO SECOND(6);
       ?column?
```

```
-----
1000 days 12:00:01.123456
```

A Vertica extension also lets you specify the seconds precision in the leftmost field. The result is the same:

```
=> SELECT INTERVAL(6) '1000 12:00:01.123456' DAY(4) TO SECOND;
       1000 12:00:01.123456
```

If you specify the seconds precision in both places, Vertica chooses the lesser value, rounding down:

```
=> SELECT INTERVAL(4) '1000 12:00:01.123456' DAY(4) TO SECOND(6);
       1000 12:00:01.1235
```

Notice that the placement of the seconds precision does not matter; Vertica chooses the lesser value, rounding down:

```
=> SELECT INTERVAL(6) '1000 12:00:01.123456' DAY(4) TO SECOND(4);
       1000 12:00:01.1235
```

- An interval-qualifier subtype can extract other values from the input parameters. For example, the following commands extract the HOUR value from the input parameters:

```
=> SELECT INTERVAL '3 days 2 hours' HOUR;
       ?column?
```

```
-----
74
```

- When specifying intervals that use subtype YEAR TO MONTH, the returned value is kept as months. For example, in SQL format, SELECT INTERVAL '2 years 6 months' YEAR TO MONTH; returns 2-6, for two years and six months. If you use interval-qualifier month, you force the system to extract months from the input parameter; for example:

```
=> SELECT INTERVAL '2 years 6 months' MONTH;
       ?column?
```

```
-----
30
```

- INTERVAL YEAR TO MONTH can be used in an analytic RANGE window when the ORDER BY column type is TIMESTAMP/TIMESTAMP WITH TIMEZONE, or DATE; TIME/TIME WITH TIMEZONE are not supported. INTERVAL DAY TO SECOND can be used when the ORDER BY column type is TIMESTAMP/TIMESTAMP WITH TIMEZONE, DATE, and TIME/TIME WITH TIMEZONE.
- When months or years are specified for day/time intervals, the intervals extension assumes 30 days per month and 365 days per year.
- Since the length of a given month or year varies, day/time intervals are never output as months or years, only as days, hours, minutes, and so on.
- If you divide an interval by an interval, you get a pure number. For example, an interval divided by an interval returns FLOAT:

```
=> SELECT INTERVAL '28 days 3 hours' HOUR(4) / INTERVAL '27 days 3 hours'
       HOUR(4);
       1.036866359447
```

- INTERVAL divided by FLOAT is INTERVAL:

```
=> SELECT INTERVAL '3' MINUTE / 1.5;
2
```

- **INTERVAL MODULO (remainder) INTERVAL returns an INTERVAL:**

```
=> SELECT INTERVAL '28 days 3 hours' HOUR(4) % INTERVAL '27 days 3 hours'
HOUR(4);
24
```

- You can add INTERVAL and TIME. TIME implicitly converts to INTERVAL, if necessary.

```
=> SELECT INTERVAL '1' HOUR + TIME '1:30';
02:30:00
```

- Vertica supports intervals in milliseconds (hh:mm:ss:ms), where 01:02:03:25 represents 1 hour, 2 minutes, 3 seconds, and 025 milliseconds.

Milliseconds are converted to fractional seconds; for example, the following command returns 1 day, 2 hours, 3 minutes, 4 seconds, and 25.5 milliseconds:

```
=> SELECT INTERVAL '1 02:03:04:25.5';
1 02:03:04.0255
```

- In the SQL-2008 standard, the placement of a minus sign either before an INTERVAL literal or as the first character of the literal negates the entire literal, not just the first component.

In Vertica a leading minus sign negates the entire interval, not just the first component. For example, both of the following commands return -29 23:59:59:

```
=> SELECT INTERVAL '-1 month - 1 second';
=> SELECT INTERVAL '-1 month - 1 second';
```

Use one of the following commands instead, which return the intended -30 00:00:01:

```
=> SELECT INTERVAL '-1 month 1 second';
=> SELECT INTERVAL '-30 00:00:01';
```

Note that two negatives together return a positive:

```
=> SELECT INTERVAL '--1 month - 1 second';
29 23:59:59
=> SELECT INTERVAL '--1 month 1 second';
30 00:00:01
```

- Vertica allows the input of negative months but requires two negatives when paired with years. Note that the year-hyphen-month syntax allows no spaces:

```
=> SELECT INTERVAL '3-3' YEAR TO MONTH;
3-3
=> SELECT INTERVAL '3--3' YEAR TO MONTH;
2-9
```

- Vertica allows fractional minutes. If the number comes out uneven enough it goes into the seconds field. In the following example, the command returns a value of 0 hours and 10 minutes:

```
=> SELECT INTERVAL '10 minutes';
00:10
```

Now specify an interval of 10.5 minutes:

```
=> SELECT INTERVAL '10.5 minutes';
00:10:30
```

- INTERVALYM is an alias for the INTERVAL YEAR TO MONTH subtypes and is used only on input. For example, the following command returns 1 year:

```
=> SELECT INTERVALYM '1' year;
1
```

However, you cannot use day as the input:

```
=> SELECT INTERVALYM '1' day;
ERROR: Conflicting INTERVAL subtypes
```

## Examples

The table that follows shows additional interval examples. The INTERVALSTYLE is set to plain (omitting units on output) for brevity.

**Note:** Remember that if you omit the *interval-qualifier* (page 34), the type defaults to DAY TO SECOND(6).

Command	Result
select interval '00:2500:00';	1 17:40
select interval '2500' minute to second;	2500
select interval '2500' minute;	2500
select interval '28 days 3 hours' hour to second;	675.00
select interval(3) '28 days 3 hours';	28 03:00
select interval(3) '28 days 3 hours 1.234567';	28 03:01:14.074
select interval(3) '28 days 3 hours 1.234567 sec';	28 03:00:01.235
select interval(3) '28 days 3.3 hours' hour to second;	675.18
select interval(3) '28 days 3.35 hours' hour to second;	675.21
select interval(3) '28 days 3.37 hours' hour to second;	675:22:12
select interval '1.234567 days' hour to second;	29:37:46.5888
select interval '1.23456789 days' hour to second;	29:37:46.665696
select interval(3) '1.23456789 days' hour to second;	29:37:46.666
select interval(3) '1.23456789 days' hour to second(2);	29:37:46.67
select interval(3) '01:00:01.234567' as "one hour+";	01:00:01.235
select interval(3) '01:00:01.234567' = interval(3) '01:00:01.234567';	t
select interval(3) '01:00:01.234567' = interval '01:00:01.234567';	f
select interval(3) '01:00:01.234567' = interval '01:00:01.234567' hour to second(3);	t
select interval(3) '01:00:01.234567' = interval '01:00:01.234567' minute to second(3);	t
select interval '255 1.1111' minute to second(3);	255:01.111
select interval '@ - 5 ago';	5
select interval '@ - 5 minutes ago';	00:05
select interval '@ 5 minutes ago';	-00:05
select interval '@ ago -5 minutes';	00:05
select date_part('month', interval '2-3' year to month);	3
SELECT FLOOR((TIMESTAMP '2005-01-17 10:00' - TIMESTAMP '2005-01-01') / INTERVAL '7');	2

## See Also

**Interval Values** (page 30) for a description of the values that can be represented in an INTERVAL type

**INTERVALSTYLE** (page 768) and **DATESTYLE** (page 767)

**AGE\_IN\_MONTHS** (page 187) and **AGE\_IN\_YEARS** (page 188)

**interval-literal**

The following table lists the units allowed for an `interval-literal` parameter.

Unit	Description
a	Julian year, 365.25 days exactly
ago	Indicates negative time offset
c, cent, century	Century
centuries	Centuries
d, day	Day
days	Days
dec, decade	Decade
decades, decs	Decades
h, hour, hr	Hour
hours, hrs	Hours
ka	Julian kilo-year, 365250 days exactly
m	Minute or month for year/month, depending on context. See Notes below this table.
microsecond	Microsecond
microseconds	Microseconds
mil, millennium	Millennium
millennia, mils	Millennia
millisecond	Millisecond
milliseconds	Milliseconds
min, minute, mn	Minute
mins, minutes	Minutes
mon, month	Month
mons, months	Months
ms, msec, millisecond	Millisecond
mseconds, msecs	Milliseconds
q, qtr, quarter	Quarter
qtrs, quarters	Quarters
s, sec, second	Second
seconds, secs	Seconds
us, usec	Microsecond

microseconds, useconds, usecs	Microseconds
w, week	Week
weeks	Weeks
y, year, yr	Year
years, yrs	Years

## Notes

The input unit 'm' can represent either 'months' or 'minutes,' depending on context. To illustrate, the following command creates a one-column table with some interval vales:

```
=> CREATE TABLE int_test(i INTERVAL YEAR TO MONTH);
```

In the first INSERT statement, the values are inserted as 1 year, six months:

```
=> INSERT INTO int_test VALUES('1 year 6 months');
```

In the second INSERT statement, the minutes value is ignored, as the DAY TO SECOND part is truncated:

```
=> INSERT INTO int_test VALUES('1 year 6 minutes');
```

In the third INSERT statement, the 'm' counts as minutes value is ignored, as the DAY TO SECOND part is truncated:

```
=> INSERT INTO int_test VALUES('1 year 6 m'); -- the m counts as months
Query the table and you will notice that the second row does not contain the minutes input:
```

```
=> SELECT * FROM int_test;
   i
-----
 1-6
 1-0
 1-6
(3 rows)
```

In the following command, the the 'm' counts as minutes, because the DAY TO SECOND interval-qualifier extracts day/time values from the input:

```
=> SELECT INTERVAL '1y6m' DAY TO SECOND;
?column?
-----
 365 00:06
(1 row)
```

## interval-qualifier

The following table lists the optional interval qualifiers. Values in `INTERVAL` fields, other than `SECOND`, are integers with a default precision of 2 when they are not the first field.

Interval Type	Units	Valid interval-literal entries
Day/time intervals	DAY	Unconstrained.
	DAY TO HOUR	An interval that represents a span of days and hours.
	DAY TO MINUTE	An interval that represents a span of days and minutes.
	DAY TO SECOND	(Default) interval that represents a span of days, hours, minutes, seconds, and fractions of a second if subtype unspecified.
	HOUR	Hours within days.
	HOUR TO MINUTE	An interval that represents a span of hours and minutes.
	HOUR TO SECOND	An interval that represents a span of hours and seconds.
	MINUTE	Minutes within hours.
	MINUTE TO SECOND	An interval that represents a span of minutes and seconds.
	SECOND	Seconds within minutes. <b>Note:</b> The <code>SECOND</code> field can have an interval fractional seconds precision, which indicates the number of decimal digits maintained following the decimal point in the <code>SECONDS</code> value. When <code>SECOND</code> is not the first field, it has a precision of 2 places before the decimal point.
Year/month intervals	MONTH	Months within year.
	YEAR	Unconstrained.
	YEAR TO MONTH	An interval that represents a span of years and months.

## Notes

You cannot combine day/time and year/month qualifiers. For example, the following intervals are not allowed:

- DAY TO YEAR
- HOUR TO MONTH

## SMALLDATETIME

SMALLDATETIME is an alias for *TIMESTAMP* (page 90).

## TIME

Consists of a time of day with or without a time zone.

### Syntax

```
TIME [ (p) ] [ { WITH | WITHOUT } TIME ZONE ] | TIMETZ
[ AT TIME ZONE (see "TIME AT TIME ZONE" on page 89) ]
```

### Parameters

<i>p</i>	(Precision) specifies the number of fractional digits retained in the seconds field. By default, there is no explicit bound on precision. The allowed range 0 to 6.
WITH TIME ZONE	Specifies that valid values must include a time zone
WITHOUT TIME ZONE	Specifies that valid values do not include a time zone (default). If a time zone is specified in the input it is silently ignored.
TIMETZ	Is the same as TIME WITH TIME ZONE with no precision

### Limits

Data Type	Low Value	High Value	Resolution
TIME [ <i>p</i> ]	00:00:00.00	23:59:59.99	1 MS / 14 digits
TIME [ <i>p</i> ] WITH TIME ZONE	00:00:00.00+1 2	23:59:59.99-12	1 ms / 14 digits

Example	Description
04:05:06.789	ISO 8601
04:05:06	ISO 8601
04:05	ISO 8601
040506	ISO 8601
04:05 AM	Same as 04:05; AM does not affect value
04:05 PM	Same as 16:05; input hour must be <= 12
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601

04:05:06 PST	Time zone specified by name
--------------	-----------------------------

## Notes

- `TIME` is purely a time-of-day, so you cannot **`ADD_MONTHS`** (page 185) to it or cast it to a `TIMESTAMP`; both of these need a date-part.
- Vertica supports adding milliseconds to a `TIME` or `TIMETZ` value.

```
=> CREATE TABLE temp (datecol TIME);
=> INSERT INTO temp VALUES (TIME '12:47:32.62');
=> INSERT INTO temp VALUES (TIME '12:55:49.123456');
=> INSERT INTO temp VALUES (TIME '01:08:15.12374578');
=> SELECT * FROM temp;
      datecol
-----
12:47:32.62
12:55:49.123456
01:08:15.123746
(3 rows)
```

## TIME AT TIME ZONE

The `TIME AT TIME ZONE` construct converts `TIMESTAMP` and `TIMESTAMP WITH ZONE` types to different time zones.

`TIME ZONE` is a synonym for `TIMEZONE`. Both are allowed in Vertica syntax.

## Syntax

```
timestamp AT TIME ZONE zone
```

## Parameters

<i>timestamp</i>	<code>TIMESTAMP</code>	Converts UTC to local time in given time zone
	<code>TIMESTAMP WITH TIME ZONE</code>	Converts local time in given time zone to UTC
	<code>TIME WITH TIME ZONE</code>	Converts local time across time zones
<i>zone</i>	<p>Is the desired time zone specified either as a text string (for example: <code>'PST'</code>) or as an interval (for example: <code>INTERVAL '-08:00'</code>). In the text case, the available zone names are abbreviations.</p> <p>The files in <code>/opt/vertica/share/timezonesets</code> define the default list of strings accepted in the <i>zone</i> parameter</p>	

## Examples

The local time zone is `PST8PDT`. The first example takes a zone-less timestamp and interprets it as `MST` time (UTC- 7) to produce a UTC timestamp, which is then rotated to `PST` (UTC-8) for display:

```
=> SELECT TIMESTAMP '2001-02-16 20:38:40' AT TIME ZONE 'MST';
      timezone
-----
```

```
2001-02-16 22:38:40-05
(1 row)
```

The second example takes a timestamp specified in EST (UTC-5) and converts it to local time in MST (UTC-7):

```
=> SELECT TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40-05' AT TIME ZONE 'MST';
timezone
-----
2001-02-16 18:38:40
(1 row)
```

## TIMESTAMP

Consists of a date and a time with or without a time zone and with or without a historical epoch (AD or BC).

### Syntax

```
TIMESTAMP [ (p) ] [ { WITH | WITHOUT } TIME ZONE ] | TIMESTAMPTZ
[ AT TIME ZONE (see "TIME AT TIME ZONE" on page 89) ]
```

### Parameters

<i>p</i>	Optional precision value that specifies the number of fractional digits retained in the seconds field. By default, there is no explicit bound on precision. The allowed range of <i>p</i> is 0 to 6.
WITH TIME ZONE	Specifies that valid values must include a time zone. All <code>TIMESTAMP WITH TIME ZONE</code> values are stored internally in UTC. They are converted to local time in the zone specified by the time zone configuration parameter before being displayed to the client.
WITHOUT TIME ZONE	Specifies that valid values do not include a time zone (default). If a time zone is specified in the input it is silently ignored.
TIMESTAMPTZ	Is the same as <code>TIMESTAMP WITH TIME ZONE</code> .

### Limits

In the following table, values are rounded.

Data Type	Low Value	High Value	Resolution
TIMESTAMP [ (p) ] [ WITHOUT TIME ZONE ]	290279 BC	294277 AD	1 US / 14 digits
TIMESTAMP [ (p) ] WITH TIME ZONE	290279 BC	294277 AD	1 US / 14 digits

### Notes

- `TIMESTAMP` is an alias for `DATETIME` and `SMALLDATETIME`.
- Valid input for `TIMESTAMP` types consists of a concatenation of a date and a time, followed by an optional time zone, followed by an optional `AD` or `BC`.
- `AD/BC` can appear before the time zone, but this is not the preferred ordering.

- The SQL standard differentiates `TIMESTAMP WITHOUT TIME ZONE` and `TIMESTAMP WITH TIME ZONE` literals by the existence of a "+" or "-". Hence, according to the standard:

`TIMESTAMP '2004-10-19 10:23:54'` is a `TIMESTAMP WITHOUT TIME ZONE`.

`TIMESTAMP '2004-10-19 10:23:54+02'` is a `TIMESTAMP WITH TIME ZONE`.

**Note:** Vertica differs from the standard by requiring that `TIMESTAMP WITH TIME ZONE` literals be explicitly typed:

```
TIMESTAMP WITH TIME ZONE '2004-10-19 10:23:54+02'
```

- If a literal is not explicitly indicated as being of `TIMESTAMP WITH TIME ZONE`, Vertica silently ignores any time zone indication in the literal. That is, the resulting date/time value is derived from the date/time fields in the input value, and is not adjusted for time zone.
- For `TIMESTAMP WITH TIME ZONE`, the internally stored value is always in UTC. An input value that has an explicit time zone specified is converted to UTC using the appropriate offset for that time zone. If no time zone is stated in the input string, then it is assumed to be in the time zone indicated by the system's `TIME ZONE` parameter, and is converted to UTC using the offset for the `TIME ZONE` zone.
- When a `TIMESTAMP WITH TIME ZONE` value is output, it is always converted from UTC to the current `TIME ZONE` zone and displayed as local time in that zone. To see the time in another time zone, either change `TIME ZONE` or use the `AT TIME ZONE` construct.
- Conversions between `TIMESTAMP WITHOUT TIME ZONE` and `TIMESTAMP WITH TIME ZONE` normally assume that the `TIMESTAMP WITHOUT TIME ZONE` value are taken or given as `TIME ZONE` local time. A different zone reference can be specified for the conversion using `AT TIME ZONE`.
- `TIMESTAMPTZ` and `TIMETZ` are not parallel SQL constructs. `TIMESTAMPTZ` records a time and date in GMT, converting from the specified `TIME ZONE`. `TIMETZ` records the specified time and the specified time zone, in minutes, from GMT.`timezone`
- The following list represents typical date/time input variations:
  - 1999-01-08 04:05:06
  - 1999-01-08 04:05:06 -8:00
  - January 8 04:05:06 1999 PST
- Vertica supports adding a floating-point (in days) to a `TIMESTAMP` or `TIMESTAMPTZ` value.
- Vertica supports adding milliseconds to a `TIMESTAMP` or `TIMESTAMPTZ` value.
- In Vertica, *intervals* (page 74) are represented internally as some number of microseconds and printed as up to 60 seconds, 60 minutes, 24 hours, 30 days, 12 months, and as many years as necessary. Fields are either positive or negative.

## Examples

You can return infinity by specifying 'infinity':

```
=> SELECT TIMESTAMP 'infinity';
timestamp
-----
infinity
(1 row)
```

To use the minimum `TIMESTAMP` value lower than the minimum rounded value:

```
=> SELECT '-infinity'::timestamp;
       timestamp
-----
 -infinity
(1 row)
```

**TIMESTAMP/TIMESTAMPTZ has +/-infinity values.**

**AD/BC can be placed almost anywhere within the input string; for example:**

```
SELECT TIMESTAMPTZ 'June BC 1, 2000 03:20 PDT';
       timestamptz
-----
 2000-06-01 05:20:00-05 BC
(1 row)
```

**Notice the results are the same if you move the BC after the 1:**

```
SELECT TIMESTAMPTZ 'June 1 BC, 2000 03:20 PDT';
       timestamptz
-----
 2000-06-01 05:20:00-05 BC
(1 row)
```

**And the same if you place the BC in front of the year:**

```
SELECT TIMESTAMPTZ 'June 1, BC 2000 03:20 PDT';
       timestamptz
-----
 2000-06-01 05:20:00-05 BC
(1 row);
```

**The following example returns the year 45 before the Common Era:**

```
=> SELECT TIMESTAMP 'April 1, 45 BC';
       timestamp
-----
 0045-04-01 00:00:00 BC
(1 row)
```

**If you omit the BC from the date input string, the system assumes you want the year 45 in the current century:**

```
=> SELECT TIMESTAMP 'April 1, 45';
       timestamp
-----
 2045-04-01 00:00:00
(1 row)
```

**In the following example, Vertica returns results in years, months, and days, whereas other RDBMS might return results in days only:**

```
=> SELECT TIMESTAMP WITH TIME ZONE '02/02/294276'- TIMESTAMP WITHOUT TIME ZONE
'02/20/2009' AS result;
       result
-----
 292266 years 11 mons 12 days
(1 row)
```

To specify a specific time zone, add it to the statement, such as the use of 'ACST' in the following example:

```
=> SELECT T1 AT TIME ZONE 'ACST', t2 FROM test;
      timezone          | t2
-----+-----
2009-01-01 04:00:00 | 02:00:00-07
2009-01-01 01:00:00 | 02:00:00-04
2009-01-01 04:00:00 | 02:00:00-06
```

You can specify a floating point in days:

```
=> SELECT 'NOW'::TIMESTAMPTZ + INTERVAL '1.5 day' AS '1.5 days from now';
      1.5 days from now
-----
2009-03-18 21:35:23.633-04
(1 row)
```

The following example illustrates the difference between TIMESTAMPTZ with and without a precision specified:

```
=> SELECT TIMESTAMPTZ(3) 'now', TIMESTAMPTZ 'now';
      timestamptz          |          timestamptz
-----+-----
2009-02-24 11:40:26.177-05 | 2009-02-24 11:40:26.177368-05
(1 row)
```

The following statement returns an error because the TIMESTAMP is out of range:

```
=> SELECT TIMESTAMP '294277-01-09 04:00:54.775808';
ERROR:  date/time field value out of range: "294277-01-09 04:00:54.775808"
```

There is no 0 AD, so be careful when you subtract BC years from AD years:

```
=> SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40');
      date_part
-----
      2001
(1 row)
```

The following commands create a table with a TIMESTAMP column that contains milliseconds:

```
CREATE TABLE temp (datecol TIMESTAMP);
INSERT INTO temp VALUES (TIMESTAMP '2010-03-25 12:47:32.62');
INSERT INTO temp VALUES (TIMESTAMP '2010-03-25 12:55:49.123456');
INSERT INTO temp VALUES (TIMESTAMP '2010-03-25 01:08:15.12374578');
SELECT * FROM temp;
      datecol
-----
2010-03-25 12:47:32.62
2010-03-25 12:55:49.123456
2010-03-25 01:08:15.123746
(3 rows)
```

## Additional Examples

Command	Result
select (timestamp '2005-01-17 10:00' - timestamp '2005-01-01');	16 10:10

<code>select (timestamp '2005-01-17 10:00' - timestamp '2005-01-01') / 7;</code>	2 08:17:08.571429
<code>select (timestamp '2005-01-17 10:00' - timestamp '2005-01-01') day;</code>	16
<code>select cast((timestamp '2005-01-17 10:00' - timestamp '2005-01-01') day as integer) / 7;</code>	2
<code>select floor((timestamp '2005-01-17 10:00' - timestamp '2005-01-01') / interval '7');</code>	2
<code>select timestamptz '2009-05-29 15:21:00.456789';</code>	2009-05-29 15:21:00.456789-04
<code>select timestamptz '2009-05-28';</code>	2009-05-28 00:00:00-04
<code>select timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2009-05-28';</code>	1 15:21:00.456789
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2009-05-28');</code>	1 15:21:00.456789
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2009-05-28') (3);</code>	1 15:21:00.457
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2009-05-28') second;</code>	141660.456789
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2009-05-28') year;</code>	0
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2007-01-01') month;</code>	28
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2007-01-01') year;</code>	2
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2007-01-01') year to month;</code>	2-4
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2009-05-28') second(3);</code>	141660.457
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2009-05-28') minute(3);</code>	2361
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2009-05-28') minute;</code>	2361
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2009-05-28') minute to second(3);</code>	2361:00.457
<code>select (timestamptz '2009-05-29 15:21:00.456789'-timestamptz '2009-05-28') minute to second;</code>	2361:00.456789

## TIMESTAMP AT TIME ZONE

The **TIMESTAMP AT TIME ZONE** construct converts **TIMESTAMP** and **TIMESTAMP WITH ZONE** types to different time zones.

**TIME ZONE** is a synonym for **TIMEZONE**. Both are allowed in Vertica syntax.

### Syntax

`timestamp AT TIME ZONE zone`

### Parameters

<i>timestamp</i>	TIMESTAMP TIMESTAMP WITH TIME ZONE TIME WITH TIME ZONE	Converts UTC to local time in given time zone Converts local time in given time zone to UTC Converts local time across time zones
<i>zone</i>	Is the desired time zone specified either as a text string (for example: 'PST') or as an interval (for example: INTERVAL '-08:00'). In the text case, the available zone names are abbreviations. The files in <code>/opt/vertica/share/timezonesets</code> define the default list of strings	

accepted in the <i>zone</i> parameter.
--

## Examples

The local time zone is PST8PDT. The first example takes a zone-less timestamp and interprets it as MST time (UTC- 7) to produce a UTC timestamp, which is then rotated to PST (UTC-8) for display:

```
=> SELECT TIMESTAMP '2001-02-16 20:38:40' AT TIME ZONE 'MST';
           timezone
-----
2001-02-16 22:38:40-05
(1 row)
```

The second example takes a timestamp specified in EST (UTC-5) and converts it to local time in MST (UTC-7):

```
=> SELECT TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40-05' AT TIME ZONE 'MST';
           timezone
-----
2001-02-16 18:38:40
(1 row)
```

## Numeric Data Types

Numeric data types are numbers stored in database columns. These data types are typically grouped by:

- **Exact** numeric types , values where the precision and scale need to be preserved. The exact numeric types are `BIGINT`, `DECIMAL`, `INTEGER`, `NUMERIC`, `NUMBER`, and `MONEY`.
- **Approximate** numeric types, values where the precision needs to be preserved and the scale can be floating. The approximate numeric types are `DOUBLE PRECISION`, `FLOAT`, and `REAL`.

Implicit casts from `INTEGER`, `FLOAT`, and `NUMERIC` to `VARCHAR` are not supported. If you need that functionality, write an explicit cast using one of the following forms:

`CAST(x AS data-type-name)` or `x::data-type-name`

The following example casts a float to an integer:

```
=> SELECT (FLOAT '123.5')::INT;
           ?column?
-----
           124
(1 row)
```

String-to-numeric data type conversions accept formats of quoted constants for scientific notation, binary scaling, hexadecimal, and combinations of numeric-type literals:

- **Scientific notation :**

```
=> SELECT FLOAT '1e10';
           ?column?
-----
10000000000
(1 row)
```

- **BINARY scaling:**

```
=> SELECT NUMERIC '1p10';
   ?column?
-----
      1024
(1 row)
```

- **Hexadecimal:**

```
=> SELECT NUMERIC '0x0abc';
   ?column?
-----
      2748
(1 row)
```

- **Combinations:**

```
=> SELECT NUMERIC '0x1pe3';
   ?column?
-----
      1000
(1 row)
```

**Note:** The *p* (which defaults to *p*0) is required for hexadecimal, because `SELECT '0x1e3'::NUMERIC = 483`.

## DOUBLE PRECISION (FLOAT)

Vertica supports the numeric data type `DOUBLE PRECISION`, which is the IEEE-754 8-byte floating point type, along with most of the usual floating point operations.

### Syntax

```
[ DOUBLE PRECISION | FLOAT | FLOAT(n) | FLOAT8 | REAL ]
```

### Parameters

**Note:** On a machine whose floating-point arithmetic does not follow IEEE-754, these values probably do not work as expected.

Double precision is an inexact, variable-precision numeric type. In other words, some values cannot be represented exactly and are stored as approximations. Thus, input and output operations involving double precision might show slight discrepancies.

- All of the `DOUBLE PRECISION` data types are synonyms for 64-bit IEEE FLOAT.
- The *n* in `FLOAT(n)` must be between 1 and 53, inclusive, but a 53-bit fraction is always used. See the IEEE-754 standard for details.
- For exact numeric storage and calculations (money for example), use `NUMERIC`.
- Floating point calculations depend on the behavior of the underlying processor, operating system, and compiler.
- Comparing two floating-point values for equality might not work as expected.

### Values

`COPY` (page 607) accepts floating-point data in the following format:

- 1 Optional leading white space
- 2 An optional plus ("+") or minus sign ("-")
- 3 A decimal number, a hexadecimal number, an infinity, a NAN, or a null value

A decimal number consists of a non-empty sequence of decimal digits possibly containing a radix character (decimal point "."), optionally followed by a decimal exponent. A decimal exponent consists of an "E" or "e", followed by an optional plus or minus sign, followed by a non-empty sequence of decimal digits, and indicates multiplication by a power of 10.

A hexadecimal number consists of a "0x" or "0X" followed by a non-empty sequence of hexadecimal digits possibly containing a radix character, optionally followed by a binary exponent. A binary exponent consists of a "P" or "p", followed by an optional plus or minus sign, followed by a non-empty sequence of decimal digits, and indicates multiplication by a power of 2. At least one of radix character and binary exponent must be present.

An infinity is either `INF` or `INFINITY`, disregarding case.

A NaN (Not A Number) is `NAN` (disregarding case) optionally followed by a sequence of characters enclosed in parentheses. The character string specifies the value of NAN in an implementation-dependent manner. (The Vertica internal representation of NAN is 0xfff8000000000000LL on x86 machines.)

When writing infinity or NAN values as constants in a SQL statement, enclose them in single quotes. For example:

```
=> UPDATE table SET x = 'Infinity'
```

**Note:** Vertica follows the IEEE definition of NaNs (IEEE 754). The SQL standards do not specify how floating point works in detail.

IEEE defines NaNs as a set of floating point values where each one is not equal to anything, even to itself. A NaN is not greater than and at the same time not less than anything, even itself. In other words, comparisons always return false whenever a NaN is involved.

However, for the purpose of sorting data, NaN values must be placed somewhere in the result. The value generated 'NaN' appears in the context of a floating point number matches the NaN value generated by the hardware. For example, Intel hardware generates (0xfff800000000000LL), which is technically a Negative, Quiet, Non-signaling NaN.

Vertica uses a different NaN value to represent floating point NULL (0x7ffffffffffffeLL). This is a Positive, Quiet, Non-signaling NaN and is reserved by Vertica

The load file format of a null value is user defined, as described in the `COPY` (page 607) command. The Vertica internal representation of a null value is 0x7ffffffffffffeLL. The interactive format is controlled by the `vsq` printing option `null`. For example:

```
\pset null '(null)'
```

The default option is not to print anything.

### Rules

- `-0 == +0`
- `1/0 = Infinity`
- `0/0 == Nan`
- `NaN != anything (even NaN)`

To search for NaN column values, use the following predicate:

```
... WHERE column != column
```

This is necessary because `WHERE column = 'Nan'` cannot be true by definition.

### Sort Order (Ascending)

- NaN
- -Inf
- numbers
- +Inf
- NULL

### Notes

- Vertica does not support `REAL (FLOAT4)` or `NUMERIC`.

- `NULL` appears last (largest) in ascending order.
- All overflows in floats generate +/-infinity or NaN, per the IEEE floating point standard.

## INTEGER

A signed 8-byte (64-bit) data type.

### Syntax

```
[ INTEGER | INT | BIGINT | INT8 | SMALLINT | TINYINT ]
```

### Parameters

`INT`, `INTEGER`, `INT8`, and `BIGINT` are all synonyms for the same signed 64-bit integer data type. Automatic compression techniques are used to conserve disk space in cases where the full 64 bits are not required.

### Notes

- The range of values is  $-2^{63}+1$  to  $2^{63}-1$ .
- $2^{63} = 9,223,372,036,854,775,808$  (19 digits).
- The value  $-2^{63}$  is reserved to represent `NULL`.
- `NULL` appears first (smallest) in ascending order.
- Vertica does not have an explicit 4-byte (32-bit integer) or smaller types. Vertica's encoding and compression automatically eliminate the storage overhead of values that fit in less than 64 bits.

### Restrictions

- The JDBC type `INTEGER` is 4 bytes and is not supported by Vertica. Use `BIGINT` instead.
- Vertica does not support the SQL/JDBC types `NUMERIC`, `SMALLINT`, or `TINYINT`.
- Vertica does not check for overflow (positive or negative) except in the aggregate function `SUM` (page 124) (). If you encounter overflow when using `SUM`, use `SUM_FLOAT` (page 125) () which converts to floating point.

## NUMERIC

Numeric data types store numeric data. For example, a money value of \$123.45 could be stored in a `NUMERIC(5, 2)` field.

### Syntax

```
NUMERIC | DECIMAL | NUMBER | MONEY [ ( precision [ , scale ] ) ]
```

### Parameters

<i>precision</i>	The number of significant decimal digits, or the number of digits that the data type stores. Precision <i>p</i> must be positive and $\leq 1024$ .
<i>scale</i>	Expressed in decimal digits and can be any integer representable in a 16-bit field. The default scale <i>s</i> is $0 \leq \text{scale} \leq \text{precision}$ ; omitting scale is the same as $s=0$ .

## Notes

- `NUMERIC`, `DECIMAL`, `NUMBER`, and `MONEY` are all synonyms that return `NUMERIC` types. Note, however, that the default values for `NUMBER` and `MONEY` are implemented a bit differently:

Type	Precision	Scale
<code>NUMERIC</code>	37	15
<code>DECIMAL</code>	37	15
<code>NUMBER</code>	38	0
<code>MONEY</code>	18	4

- `NUMERIC` data types support exact representations of numbers that can be expressed with a number of digits before and after a decimal point. This contrasts slightly with existing Vertica data types:
  - `DOUBLE PRECISION` (page 97) (`FLOAT`) types support ~15 digits, variable exponent, and represent numeric values approximately.
  - `INTEGER` (page 100) (and similar) types support ~18 digits, whole numbers only.
- `NUMERIC` data types are generally called *exact* numeric data types because they store numbers of a specified precision and scale. The *approximate* numeric data types, such as `DOUBLE PRECISION`, use floating points and are less precise.
- Supported numeric operations include the following:
  - Basic math; for example, `+`, `-`, `*`, `/`
  - Aggregation; for example, `SUM`, `MIN`, `MAX`, `COUNT`
  - Comparison operators; for example, `<=`, `=`, `<=>`, `<>`, `>`, `>=`
- `NUMERIC` divide operates directly on numeric values, without converting to floating point. The result has at least 18 decimal places and is rounded.
- `NUMERIC` mod (including `%`) operates directly on numeric values, without converting to floating point. The result has the same scale as the numerator and never needs rounding.
- **`COPY`** (page 607) accepts `DECIMAL` number with a decimal point (`'.'`), prefixed by `-` or `+` (optional).
- `LZO`, `RLE`, and `BLOCK_DICT` are supported encoding types. Anything that can be used on an `INTEGER` can also be used on a `NUMERIC`, as long as the precision is `<= 18`.
- `NUMERIC` is preferred for non-integer constants, as this typically improves precision. For example:
 

```
=> SELECT 1.1 + 2.2 = 3.3;
      ?column?
-----
      t
(1 row)
=> SELECT 1.1::float + 2.2::float = 3.3::float;
      ?column?
-----
      f
(1 row)
```
- Performance of the `NUMERIC` data type has been fine tuned for the common case of 18 digits of precision.

- Some of the more complex operations used with `NUMERIC` data types result in an implicit cast to `FLOAT`. When using `SQRT`, `STDDEV`, transcendental functions such as `LOG`, and `TO_CHAR/TO_NUMBER` formatting, the result is always `FLOAT`.

### Examples

The following series of commands creates a table that contains a `NUMERIC` data type and then performs some mathematical operations on the data:

```
=> CREATE TABLE num1 (id INTEGER, amount NUMERIC(8,2));
```

Now insert some values into the table:

```
=> INSERT INTO num1 VALUES (1, 123456.78);
```

Query the table:

```
=> SELECT * FROM num1;
   id | amount
-----+-----
     1 | 123456.78
(1 row)
```

The following example returns the `NUMERIC` column, `amount`, from table `num1`:

```
=> SELECT amount FROM num1;
 amount
-----
123456.78
(1 row)
```

The following syntax adds one (1) to the amount:

```
=> SELECT amount+1 AS 'amount' FROM num1;
 amount
-----
123457.78
(1 row)
```

The following syntax multiplies the `amount` column by 2:

```
=> SELECT amount*2 AS 'amount' FROM num1;
 amount
-----
246913.56
(1 row)
```

The following syntax returns a negative number for the `amount` column:

```
=> SELECT -amount FROM num1;
?column?
-----
-123456.78
(1 row)
```

The following syntax returns the absolute value of the `amount` argument:

```
=> SELECT ABS(amount) FROM num1;
 ABS
-----
```

```
123456.78
(1 row)
```

The following syntax casts the NUMERIC amount as a FLOAT data type:

```
=> SELECT amount::float FROM num1;
      amount
-----
123456.78
(1 row)
```

## See Also

**Mathematical Functions** (page 279)

## Numeric Data Type Overflow

Vertica does not check for overflow (positive or negative) except in the aggregate function SUM (page 124) (). If you encounter overflow when using SUM, use SUM\_FLOAT (page 125) () which converts to floating point.

For INTEGER data types, dividing zero by zero returns zero:

```
=> SELECT 0/0;
      ?column?
-----
              0
(1 row)
```

Dividing anything else by zero returns a run-time error.

```
=> SELECT 1/0;
      ERROR:  division by zero
=> SELECT 0.0/0;
      ERROR:  numeric division by zero
```

Add, subtract, and multiply operations ignore overflow. Sum and average operations use 128-bit arithmetic internally. SUM (page 124) () reports an error if the final result overflows, suggesting the use of SUM\_FLOAT (page 125) (INT), which converts the 128-bit sum to a FLOAT8. For example:

```
=> CREATE TEMP TABLE t (i INT);
=> INSERT INTO t VALUES (1<<62);
=> SELECT SUM(i) FROM t;
      ERROR:  sum() overflowed
      HINT:  try sum_float() instead
=> SELECT SUM_FLOAT(i) FROM t;
      sum_float
-----
2.30584300921369e+19
```

## Data Type Coercion

Vertica currently has two types of cast, implicit and explicit. Vertica implicitly casts (coerces) expressions from one type to another under certain circumstances.

To illustrate, first get today's date:

```
=> SELECT DATE 'now';
   ?column?
-----
2010-10-19
(1 row)
```

The following command converts `DATE` to a `TIMESTAMP` and adds a day and a half to the results by using `INTERVAL`:

```
=> SELECT DATE 'now' + INTERVAL '1 12:00:00';
   ?column?
-----
2010-10-20 12:00:00
(1 row)
```

When there is no ambiguity as to the data type of an expression value, it is implicitly coerced to match the expected data type. In the following command, the quoted string constant '2' is implicitly coerced into an `INTEGER` value so that it can be the operand of an arithmetic operator (addition):

```
=> SELECT 2 + '2';
   ?column?
-----
         4
(1 row)
```

The result of the following arithmetic expression `2 + 2` and the `INTEGER` constant `2` are implicitly coerced into `VARCHAR` values so that they can be concatenated.

```
=> SELECT 2 + 2 || 2;
   ?column?
-----
        42
(1 row)
```

Most implicit casts stay within their relational family and go in one direction, from less detailed to more detailed. For example:

- `DATE` to `TIMESTAMP/TZ`
- `INTEGER` to `NUMERIC` to `FLOAT`
- `CHAR` to `FLOAT`
- `CHAR` to `VARCHAR`

`CHAR` and/or `VARCHAR` to `FLOAT` More specifically, data type coercion works in this manner in Vertica:

- `INT8` -> `FLOAT8`—implicit, can lose significance
- `FLOAT8` -> `INT8`—explicit, rounds

- VARCHAR <-> CHAR—implicit, adjusts trailing spaces
- VARBINARY <-> BINARY—implicit, adjusts trailing NULs

No other types cast to or from varbinary or binary. In the following list, <any> means one these types: INT8, FLOAT8, DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPTZ, INTERVAL

- <any> -> VARCHAR—implicit
- VARCHAR -> <any>—explicit, except that VARCHAR->FLOAT is implicit
- <any> <-> CHAR—explicit
- DATE -> TIMESTAMP/TZ—implicit
- TIMESTAMP/TZ -> DATE—explicit, loses time-of-day
- TIME -> TIMETZ—implicit, adds local timezone
- TIMETZ -> TIME—explicit, loses timezone
- TIME -> INTERVAL—implicit, day to second with days=0
- INTERVAL -> TIME—explicit, truncates non-time parts
- TIMESTAMP <-> TIMESTAMPTZ—implicit, adjusts to local timezone
- TIMESTAMP/TZ -> TIME—explicit, truncates non-time parts
- TIMESTAMPTZ -> TIMETZ—explicit

---

**IMPORTANT:** Implicit casts from INTEGER, FLOAT, and NUMERIC to VARCHAR are not supported. If you need that functionality, write an explicit cast:

```
CAST(x AS data-type-name)
```

or

```
x::data-type-name
```

The following example casts a FLOAT to an INTEGER:

```
=> SELECT (FLOAT '123.5')::INT;
   ?column?
-----
          124
(1 row)
```

---

String-to-numeric data type conversions accept formats of quoted constants for scientific notation, binary scaling, hexadecimal, and combinations of numeric-type literals:

- Scientific notation :

```
=> SELECT FLOAT '1e10';
   ?column?
-----
10000000000
(1 row)
```

- BINARY scaling :

```
=> SELECT NUMERIC '1p10';
   ?column?
-----
          1024
```



-----  
 2010-03-02  
 (1 row)

## See Also

**Data Type Coercion Chart** (page 107)

**Data Type Coercion Operators (CAST)** (page 38)

## Data Type Coercion Chart

### Conversion Types

The following table defines all possible type conversions that Vertica supports. The values across the top row are the data types you want, and the values down the first column on the left are the data types that you have.

Want >	BOOL	INT	FLT	CHR	VCHR	DTM	TM	TS	TSTZ	INVL	TTZ	NUM	VBIN	BIN	INTYM
Have															
BOOL	N/A			a	a										
INT		N/A	i	a**	a**					a		i			a
FLT		a	N/A	a	a							a			
CHR	e	e	i	Yes	i	e	e	e	e	e	e	e			e
VCHR	e	e	i	i	Yes	e	e	e	e	e	e	e			e
DTM				a	a	N/A		i	a						
TM				a	a		Yes			i	i				
TS				a	a	a	a	Yes	i						
TSTZ				a	a	a	a	i	Yes		a				
INVL		a		a	a		a			Yes					
TTZ				a	a		a				Yes				
NUM		a	i	a	a							Yes			
VBIN													Yes	i	
BIN													i	Yes	
INTYM		a		a	a										Yes

## KEY

### Type:

(i)mplicit,  
(a)ssignment,  
(e)xplicit

### Matrix:

\*\* means that the numeric meaning is lost, and the value is subject to (VAR)CHAR compares

### Abbreviation:

BOOL = Boolean  
INT = Integer  
FLT = Float  
CHR = Char  
VCHR = Varchar  
DTM = Date  
TM = Time  
TS = Timestamp  
TSTZ = Timestamp with Time Zone  
INVL = Interval Day to Second  
TTZ = Time with time zone  
NUM = Numeric  
VBIN = Varbinary  
BIN = Binary  
INTYM = Interval Year to Month

## See Also

***Data Type Coercion Operators (CAST)*** (page 38)

# SQL Functions

---

Functions return information from the database and are allowed anywhere an expression is allowed. The exception is **Vertica-specific functions** (page 374), which are not allowed everywhere.

Some functions could produce different results on different invocations with the same set of arguments. The following three categories of functions are defined based on their behavior:

- **Immutable (invariant):** When run with a given set of arguments, immutable functions always produces the same result. The function is independent on any environment or session settings, such as locale. For example, 2+2 always equals 4. Another immutable function is AVG(). Some immutable functions can take an optional stable argument; in this case they are treated as stable functions.
- **Stable:** When run with a given set of arguments, stable functions produce the same result within a single query or scan operation. However, a stable function could produce different results when issued under a different environment, such as a change of locale and time zone. Expressions that could give different results in the future are also stable, for example `SYSDATE()` or `'today'`.
- **Volatile:** Regardless of the arguments or environment, volatile functions can return different results on multiple invocations. `RANDOM()` is one example.

This chapter describes the functions that Vertica supports.

- Each function is annotated with behavior type as immutable, stable or volatile.
- All Vertica-specific functions can be assumed to be volatile and are not annotated individually.

## Aggregate Functions

**Note:** All functions in this section that have an *analytic* (page 128) function counterpart are appended with [Aggregate] to avoid confusion between the two.

Aggregate functions summarize data over groups of rows from a query result set. The groups are specified using the **GROUP BY** (page 754) clause. They are allowed only in the select list and in the **HAVING** (page 756) and **ORDER BY** (page 757) clauses of a **SELECT** (page 747) statement (as described in **Aggregate Expressions** (page 44)).

### Notes

- Except for COUNT, these functions return a null value when no rows are selected. In particular, SUM of no rows returns NULL, not zero.
- In some cases you can replace an expression that includes multiple aggregates with a single aggregate of an expression. For example SUM(x) + SUM(y) can be expressed as SUM(x+y) (where x and y are NOT NULL).
- Vertica does not support nested aggregate functions.

You can also use some of the simple aggregate functions as analytic (window) functions. See **Analytic Functions** (page 128) for details. See also Using SQL Analytics in the Programmer's Guide.

### AVG [Aggregate]

Computes the average (arithmetic mean) of an expression over a group of rows. It returns a DOUBLE PRECISION value for a floating-point expression. Otherwise, the return value is the same as the expression data type.

### Behavior Type

Immutable

### Syntax

```
AVG ( [ ALL | DISTINCT ] expression )
```

### Parameters

ALL	Invokes the aggregate function for all rows in the group (default).
DISTINCT	Invokes the aggregate function for all distinct non-null values of the expression found in the group.
<i>expression</i>	The value whose average is calculated over a set of rows. Can be any expression resulting in DOUBLE PRECISION.

### Notes

The AVG () aggregate function is different from the AVG () analytic function, which computes an average of an expression over a group of rows within a window.

## Examples

The following example returns the average income from the `customer` table:

```
=> SELECT AVG(annual_income) FROM customer_dimension;
      avg
-----
2104270.6485
(1 row)
```

## See Also

**AVG** (page 136) analytic function

**COUNT** (page 111) and **SUM** (page 124)

**Numeric Data Types** (page 95)

## CORR

Returns the coefficient of correlation of a set of expression pairs (*expression1* and *expression2*). The return value is of type DOUBLE PRECISION. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL. Syntax

```
SELECT CORR (expression1,expression2)
```

## Parameters

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.

## Example

```
=> SELECT CORR (Annual_salary, Employee_age) FROM employee_dimension;
      CORR
-----
-0.00719153413192422
(1 row)
```

## COUNT [Aggregate]

Returns the number of rows in each group of the result set for which the expression is not NULL. The return value is a BIGINT.

## Behavior Type

Immutable

## Syntax

```
COUNT ( [ * ] [ ALL | DISTINCT ] expression )
```

## Parameters

*	Indicates that the count does not apply to any specific column or expression in
---	---

	the select list. Requires a <code>FROM clause</code> (page 750).
ALL	Invokes the aggregate function for all rows in the group (default).
DISTINCT	Invokes the aggregate function for all distinct non-null values of the expression found in the group.
<i>expression</i>	Returns the number of rows in each group for which the <i>expression</i> is not null. Can be any expression resulting in <code>BIGINT</code> .

## Notes

The `COUNT ()` aggregate function is different from the `COUNT ()` analytic function, which returns the number over a group of rows within a window.

## Examples

The following query returns the number of distinct values in the `primary_key` column of the `date_dimension` table:

```
=> SELECT COUNT (DISTINCT date_key) FROM date_dimension;
count
-----
  1826
(1 row)
```

The next example returns all distinct values of evaluating the expression `x+y` for all records of `fact`.

```
=> SELECT COUNT (DISTINCT date_key + product_key) FROM inventory_fact;
count
-----
 21560
(1 row)
```

An equivalent query is as follows (using the `LIMIT` key to restrict the number of rows returned):

```
=> SELECT COUNT(date_key + product_key) FROM inventory_fact
      GROUP BY date_key LIMIT 10;
count
-----
   173
    31
   321
   113
   286
    84
   244
   238
   145
   202
(10 rows)
```

Each distinct `product_key` value in table `inventory_fact` and returns the number of distinct values of `date_key` in all records with the specific distinct `product_key` value.

```
=> SELECT product_key, COUNT (DISTINCT date_key) FROM inventory_fact
```

```
GROUP BY product_key LIMIT 10;
```

```
product_key | count
-----+-----
          1 |    12
          2 |    18
          3 |    13
          4 |    17
          5 |    11
          6 |    14
          7 |    13
          8 |    17
          9 |    15
         10 |    12
```

(10 rows)

**This query counts each distinct `product_key` value in table `inventory_fact` with the constant "1".**

```
=> SELECT product_key, COUNT (DISTINCT product_key) FROM inventory_fact
      GROUP BY product_key LIMIT 10;
```

```
product_key | count
-----+-----
          1 |     1
          2 |     1
          3 |     1
          4 |     1
          5 |     1
          6 |     1
          7 |     1
          8 |     1
          9 |     1
         10 |     1
```

(10 rows)

**This query selects each distinct `date_key` value and counts the number of distinct `product_key` values for all records with the specific `product_key` value. It then sums the `qty_in_stock` values in all records with the specific `product_key` value and groups the results by `date_key`.**

```
=> SELECT date_key, COUNT (DISTINCT product_key), SUM(qty_in_stock) FROM
inventory_fact
      GROUP BY date_key LIMIT 10;
```

```
date_key | count | sum
-----+-----+-----
          1 |    173 | 88953
          2 |     31 | 16315
          3 |    318 | 156003
          4 |    113 | 53341
          5 |    285 | 148380
          6 |     84 | 42421
          7 |    241 | 119315
          8 |    238 | 122380
          9 |    142 | 70151
         10 |    202 | 95274
```

(10 rows)

This query selects each distinct `product_key` value and then counts the number of distinct `date_key` values for all records with the specific `product_key` value and counts the number of distinct `warehouse_key` values in all records with the specific `product_key` value.

```
=> SELECT product_key, COUNT (DISTINCT date_key), COUNT (DISTINCT warehouse_key)
      FROM inventory_fact GROUP BY product_key LIMIT 15;
```

product_key	count	count
1	12	12
2	18	18
3	13	12
4	17	18
5	11	9
6	14	13
7	13	13
8	17	15
9	15	14
10	12	12
11	11	11
12	13	12
13	9	7
14	13	13
15	18	17

(15 rows)

This query selects each distinct `product_key` value, counts the number of distinct `date_key` and `warehouse_key` values for all records with the specific `product_key` value, and then sums all `qty_in_stock` values in records with the specific `product_key` value. It then returns the number of `product_version` values in records with the specific `product_key` value.

```
=> SELECT product_key, COUNT (DISTINCT date_key), COUNT (DISTINCT warehouse_key),
      SUM (qty_in_stock), COUNT (product_version)
      FROM inventory_fact GROUP BY product_key LIMIT 15;
```

product_key	count	count	sum	count
1	12	12	5530	12
2	18	18	9605	18
3	13	12	8404	13
4	17	18	10006	18
5	11	9	4794	11
6	14	13	7359	14
7	13	13	7828	13
8	17	15	9074	17
9	15	14	7032	15
10	12	12	5359	12
11	11	11	6049	11
12	13	12	6075	13
13	9	7	3470	9
14	13	13	5125	13
15	18	17	9277	18

(15 rows)

The following example returns the number of warehouses from the `warehouse` dimension table:

```
=> SELECT COUNT(warehouse_name) FROM warehouse_dimension;
count
```

```
-----
      100
(1 row)
```

The next example returns the total number of vendors:

```
=> SELECT COUNT(*) FROM vendor_dimension;
      count
-----
       50
(1 row)
```

### See Also

**Analytic Functions** (page 128)

**AVG** (page 110)

**SUM** (page 124)

Using SQL Analytics in the Programmer's Guide

## COVAR\_POP

Returns the population covariance for a set of expression pairs (*expression1* and *expression2*). The return value is of type DOUBLE PRECISION. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

### Syntax

```
SELECT COVAR_POP (expression1,expression2)
```

### Parameters

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.

### Example

```
=> SELECT COVAR_POP (Annual_salary, Employee_age) FROM employee_dimension;
      COVAR_POP
-----
-9032.34810730019
(1 row)
```

## COVAR\_SAMP

Returns the sample covariance for a set of expression pairs (*expression1* and *expression2*). The return value is of type DOUBLE PRECISION. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

### Syntax

```
COVAR_SAMP (expression1,expression2)
```

## Parameters

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.

## Example

```
=> SELECT COVAR_SAMP (Annual_salary, Employee_age) FROM employee_dimension;  
      COVAR_SAMP
```

```
-----  
-9033.25143244343  
(1 row)
```

## MAX [Aggregate]

Returns the greatest value of an expression over a group of rows. The return value is the same as the expression data type.

## Behavior Type

Immutable

## Syntax

```
MAX ( [ ALL | DISTINCT ] expression )
```

## Parameters

ALL   DISTINCT	Are meaningless in this context.
<i>expression</i>	Can be any expression for which the maximum value is calculated, typically a <b>column reference</b> (see " <b>Column References</b> " on page 46).

## Notes

The MAX () aggregate function is different from the MAX () analytic function, which returns the maximum value of an expression over a group of rows within a window.

## Example

This example returns the largest value (dollar amount) of the `sales_dollar_amount` column.

```
=> SELECT MAX(sales_dollar_amount) AS highest_sale FROM store.store_sales_fact;  
highest_sale
```

```
-----  
                600  
(1 row)
```

## See Also

**Analytic Functions** (page 128)

**MIN** (page 117)

## MIN [Aggregate]

Returns the smallest value of an expression over a group of rows. The return value is the same as the expression data type.

### Behavior Type

Immutable

### Syntax

```
MIN ( [ ALL | DISTINCT ] expression )
```

### Parameters

ALL   DISTINCT	Are meaningless in this context.
<i>expression</i>	Can be any expression for which the minimum value is calculated, typically a <b>column reference</b> (see " <b>Column References</b> " on page 46).

### Notes

The `MIN()` aggregate function is different from the `MIN()` analytic function, which returns the minimum value of an expression over a group of rows within a window.

### Example

This example returns the lowest salary from the `employee` dimension table.

```
=> SELECT MIN(annual_salary) AS lowest_paid FROM employee_dimension;
   lowest_paid
-----
          1200
(1 row)
```

### See Also

**Analytic Functions** (page 128)

**MAX** (page 116)

Using SQL Analytics in the Programmer's Guide

## REGR\_AVGX

Returns the average of the independent expression in an expression pair (*expression1* and *expression2*). The return value is of type DOUBLE PRECISION. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

### Syntax

```
SELECT REGR_AVGX (expression1,expression2)
```

## Parameters

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.

## Example

```
=>=> SELECT REGR_AVGX (Annual_salary, Employee_age) FROM employee_dimension;
      REGR_AVGX
-----
      39.321
(1 row)
```

## REGR\_AVGY

Returns the average of the dependent expression in an expression pair (*expression1* and *expression2*). The return value is of type DOUBLE PRECISION. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

## Syntax

```
REGR_AVGY (expression1,expression2)
```

## Parameters

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.

## Example

```
=> SELECT REGR_AVGY (Annual_salary, Employee_age) FROM employee_dimension;
      REGR_AVGY
-----
 58354.4913
(1 row)

(1 row)
```

## REGR\_COUNT

Returns the number of expression pairs (*expression1* and *expression2*). The return value is of type INTEGER. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns 0.

## Syntax

```
SELECT REGR_COUNT (expression1, expression2)
```

## Parameters

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
--------------------	--

<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.
--------------------	--

**Example**

```
=> SELECT REGR_COUNT (Annual_salary, Employee_age) FROM employee_dimension;
      REGR_COUNT
-----
          10000
(1 row)
```

**REGR\_INTERCEPT**

Returns the y-intercept of the regression line determined by a set of expression pairs (*expression1* and *expression2*). The return value is of type DOUBLE PRECISION. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

**Syntax**

```
SELECT REGR_INTERCEPT (expression1,expression2)
```

**Parameters**

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.

**Example**

```
b=> SELECT REGR_INTERCEPT (Annual_salary, Employee_age) FROM employee_dimension;
      REGR_INTERCEPT
-----
59929.5490163437
(1 row)
```

**REGR\_R2**

Returns the square of the correlation coefficient of a set of expression pairs (*expression1* and *expression2*). The return value is of type DOUBLE PRECISION. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

**Syntax**

```
SELECT REGR_R2 (expression1,expression2)
```

**Parameters**

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.

**Example**

```
=> SELECT REGR_R2 (Annual_salary, Employee_age) FROM employee_dimension;
```

```
      REGR_R2
-----
5.17181631706311e-05
(1 row)
```

## REGR\_SLOPE

Returns the slope of the regression line, determined by a set of expression pairs (*expression1* and *expression2*). The return value is of type DOUBLE PRECISION. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

### Syntax

```
SELECT REGR_SLOPE (expression1,expression2)
```

### Parameters

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.

### Example

```
=> SELECT REGR_SLOPE (Annual_salary, Employee_age) FROM employee_dimension;
      REGR_SLOPE
-----
-40.056400303749
(1 row)
```

## REGR\_SXX

Returns the sum of squares of the independent expression in an expression pair (*expression1* and *expression2*). The return value is of type DOUBLE PRECISION. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.,

### Syntax

```
SELECT REGR_SXX (expression1,expression2)
```

### Parameters

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.

### Example

```
=> SELECT REGR_SXX (Annual_salary, Employee_age) FROM employee_dimension;
```

```

REGR_SXX
-----
2254907.59
(1 row)

```

## REGR\_SXY

Returns the sum of products of the independent expression multiplied by the dependent expression in an expression pair (*expression1* and *expression2*). The return value is of type DOUBLE PRECISION. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

### Syntax

```
SELECT REGR_SXY (expression1,expression2)
```

### Parameters

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.

### Example

```

=> SELECT REGR_SXY (Annual_salary, Employee_age) FROM employee_dimension;
      REGR_SXY
-----
-90323481.0730019
(1 row)

```

## REGR\_SYY

Returns the sum of squares of the dependent expression in an expression pair (*expression1* and *expression2*). The return value is of type DOUBLE PRECISION. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

### Syntax

```
SELECT REGR_SYY (expression1,expression2)
```

### Parameters

<i>expression1</i>	The dependent expression. Is of type DOUBLE PRECISION.
<i>expression2</i>	The independent expression. Is of type DOUBLE PRECISION.

### Example

```

=> SELECT REGR_SYY (Annual_salary, Employee_age) FROM employee_dimension;
      REGR_SYY
-----
69956728794707.2
(1 row)

```

## STDDEV [Aggregate]

**Note:** The non-standard function `STDDEV()` is provided for compatibility with other databases. It is semantically identical to `STDDEV_SAMP()` (page 123).

Evaluates the statistical sample standard deviation for each member of the group. The `STDDEV_SAMP()` return value is the same as the square root of the `VAR_SAMP()` function:

```
STDDEV(expression) = SQRT(VAR_SAMP(expression))
```

When `VAR_SAMP()` returns null, this function returns null.

### Behavior Type

Immutable

### Syntax

```
STDDEV ( expression )
```

### Parameters

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
-------------------	--

### Notes

The `STDDEV()` aggregate function is different from the `STDDEV()` analytic function, which computes the statistical sample standard deviation of the current row with respect to the group of rows within a window.

### Examples

The following example returns the statistical sample standard deviation for each household ID from the `customer` dimension table.

```
=> SELECT STDDEV_SAMP(household_id) FROM customer_dimension;
      stddev_samp
-----
8651.50842400771
(1 row)
```

### See Also

**Analytic Functions** (page 128)

**STDDEV\_SAMP** (page 123)

Using SQL Analytics in the Programmer's Guide

## STDDEV\_POP [Aggregate]

Evaluates the statistical population standard deviation for each member of the group. The `STDDEV_POP()` return value is the same as the square root of the `VAR_POP()` function

```
STDDEV_POP(expression) = SQRT(VAR_POP(expression))
```

When `VAR_SAMP()` returns null, this function returns null.

### Behavior Type

Immutable

### Syntax

```
STDDEV_POP ( expression )
```

### Parameters

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
-------------------	--

### Notes

The `STDDEV_POP()` aggregate function is different from the `STDDEV_POP()` analytic function, which evaluates the statistical population standard deviation for each member of the group of rows within a window.

### Examples

The following example returns the statistical population standard deviation for each household ID in the `customer` table.

```
=> SELECT STDDEV_POP(household_id) FROM customer_dimension;
      stddev_samp
-----
8651.41895973367
(1 row)
```

### See Also

**Analytic Functions** (page 128)

Using SQL for Analytics in the Programmer's Guide

## STDDEV\_SAMP [Aggregate]

Evaluates the statistical sample standard deviation for each member of the group. The `STDDEV_SAMP()` return value is the same as the square root of the `VAR_SAMP()` function:

```
STDDEV_SAMP(expression) = SQRT(VAR_SAMP(expression))
```

When `VAR_SAMP()` returns null, this function returns null.

**Behavior Type:**

Immutable

**Syntax**`STDDEV_SAMP ( expression )`**Parameters**

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
-------------------	--

**Notes**

- `STDDEV_SAMP()` is semantically identical to the non-standard function, `STDDEV()` (page 122), which is provided for compatibility with other databases.
- The `STDDEV_SAMP()` aggregate function is different from the `STDDEV_SAMP()` analytic function, which computes the statistical sample standard deviation of the current row with respect to the group of rows within a window.

**Examples**

The following example returns the statistical sample standard deviation for each household ID from the `customer` dimension table.

```
=> SELECT STDDEV_SAMP(household_id) FROM customer_dimension;
      stddev_samp
-----
8651.50842400771
(1 row)
```

**See Also**

**Analytic Functions** (page 128)

**STDDEV** (page 122)

Using SQL Analytics in the Programmer's Guide

**SUM [Aggregate]**

Computes the sum of an expression over a group of rows. It returns a `DOUBLE PRECISION` value for a floating-point expression. Otherwise, the return value is the same as the expression data type.

**Behavior Type**

Immutable

**Syntax**`SUM ( [ ALL | DISTINCT ] expression )`

## Parameters

ALL	Invokes the aggregate function for all rows in the group (default)
DISTINCT	Invokes the aggregate function for all distinct non-null values of the expression found in the group
<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.

## Notes

- The `SUM()` aggregate function is different from the `SUM()` analytic function, which computes the sum of an expression over a group of rows within a window.
- If you encounter data overflow when using `SUM()`, use `SUM_FLOAT()` (page 125) which converts the data to a floating point.

## Example

This example returns the total sum of the `product_cost` column.

```
=> SELECT SUM(product_cost) AS cost FROM product_dimension;
      cost
-----
  9042850
(1 row)
```

## See Also

**AVG** (page 110)

**COUNT** (page 111)

**Numeric Data Types** (page 95)

Using SQL Analytics in the Programmer's Guide

## SUM\_FLOAT [Aggregate]

Computes the sum of an expression over a group of rows. It returns a `DOUBLE PRECISION` value for the expression, regardless of the expression type.

## Behavior Type

Immutable

## Syntax

```
SUM_FLOAT ( [ ALL | DISTINCT ] expression )
```

## Parameters

ALL	Invokes the aggregate function for all rows in the group (default).
DISTINCT	Invokes the aggregate function for all distinct non-null values of the

	expression found in the group.
<i>expression</i>	Can be any expression resulting in DOUBLE PRECISION.

**Example**

The following example returns the floating point sum of the average price from the product table:

```
=> SELECT SUM_FLOAT(average_competitor_price) AS cost FROM product_dimension;
cost
-----
18181102
(1 row)
```

**VAR\_POP [Aggregate]**

Evaluates the population variance for each member of the group. This is defined as the sum of squares of the difference of *expression* from the mean of *expression*, divided by the number of rows remaining.

$$\frac{(\text{SUM}(\text{expression} * \text{expression}) - \text{SUM}(\text{expression}) * \text{SUM}(\text{expression}))}{\text{COUNT}(\text{expression})} / \text{COUNT}(\text{expression})$$
**Behavior Type**

Immutable

**Syntax**

```
VAR_POP ( expression )
```

**Parameters**

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
-------------------	--

**Notes**

The VAR\_POP () aggregate function is different from the VAR\_POP () analytic function, which computes the population variance of the current row with respect to the group of rows within a window.

**Examples**

The following example returns the population variance for each household ID in the customer table.

```
=> SELECT VAR_POP(household_id) FROM customer_dimension;
var_pop
-----
74847050.0168393
(1 row)
```

## VAR\_SAMP [Aggregate]

Evaluates the sample variance for each row of the group. This is defined as the sum of squares of the difference of *expression* from the mean of *expression*, divided by the number of rows remaining minus 1 (one).

$$\frac{(\text{SUM}(\text{expression} * \text{expression}) - \text{SUM}(\text{expression}) * \text{SUM}(\text{expression}))}{\text{COUNT}(\text{expression}) * (\text{COUNT}(\text{expression}) - 1)}$$

### Behavior Type

Immutable

### Syntax

```
VAR_SAMP ( expression )
```

### Parameters

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
-------------------	--

### Notes

- `VAR_SAMP()` is semantically identical to the non-standard function, `VARIANCE (page 127) ()`, which is provided for compatibility with other databases.
- The `VAR_SAMP()` aggregate function is different from the `VAR_SAMP()` analytic function, which computes the sample variance of the current row with respect to the group of rows within a window.

### Examples

The following example returns the sample variance for each household ID in the `customer` table.

```
=> SELECT VAR_SAMP(household_id) FROM customer_dimension;
       var_samp
-----
74848598.0106764
(1 row)
```

### See Also

**Analytic Functions** (page 128)

**VARIANCE** (page 127)

Using SQL Analytics in the Programmer's Guide

## VARIANCE [Aggregate]

**Note:** The non-standard function `VARIANCE ()` is provided for compatibility with other databases. It is semantically identical to `VAR_SAMP ()` (page 127).

Evaluates the sample variance for each row of the group. This is defined as the sum of squares of the difference of *expression* from the mean of *expression*, divided by the number of rows remaining minus 1 (one).

$$\frac{(\text{SUM}(\text{expression} * \text{expression}) - \text{SUM}(\text{expression}) * \text{SUM}(\text{expression}) / \text{COUNT}(\text{expression}))}{(\text{COUNT}(\text{expression}) - 1)}$$

## Behavior Type

Immutable

## Syntax

```
VARIANCE ( expression )
```

## Parameters

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
-------------------	--

## Notes

The `VARIANCE ()` aggregate function is different from the `VARIANCE ()` analytic function, which computes the sample variance of the current row with respect to the group of rows within a window.

## Examples

The following example returns the sample variance for each household ID in the `customer` table.

```
=> SELECT VARIANCDE(household_id) FROM customer_dimension;
      variance
-----
74848598.0106764
(1 row)
```

## See Also

**Analytic Functions** (page 128)

**VAR\_SAMP** (page 127)

Using SQL Analytics in the Programmer's Guide

## Analytic Functions

**Note:** All analytic functions in this section that have an aggregate counterpart are appended with [Analytics] in the heading to avoid confusion between the two.

The ANSI SQL 99 standard introduced a set of functionality, called SQL analytic functions, that handle complex analysis and reporting, for example, a moving average of retail volume over a specified time frame or a running total.

Analytic aggregate functions differ from standard aggregate functions in that, rather than return a single summary value, they return the same number of rows as the input. Moreover, unlike standard aggregate functions, the groups of rows on which the analytic aggregate function operates are not defined by a `GROUP BY` clause, but by window partitioning and frame clauses. You can sort these partitions using a window `ORDER BY` clause, but the order affects only the function result set, not the entire query result set. This ordering concept is described more fully later.

The windowing components (partitioning, ordering, and framing) are specified in the analytic `OVER()` clause. For example, window framing defines the unique construct of a moving window, whose size is based on either logical intervals (such as time) or on a physical number of rows. For each row, a window is computed in relation to the current row. As the current row advances, the window moves along with it.

### Analytic Function Syntax

```
ANALYTIC_FUNCTION( argument-1, ..., argument-n )
  OVER( [ window_partition_clause (on page 130) ]
        [ window_order_clause (on page 131) ]
        [ window_frame_clause (on page 133) ] )
```

### Analytic Syntactic Construct

<code>OVER(...)</code>	Specifies partitioning, ordering, and window framing for the function—important elements that determine what data the analytic function takes as input with respect to the current row. The <code>OVER()</code> clause is evaluated after the <code>FROM</code> , <code>WHERE</code> , <code>GROUP BY</code> , and <code>HAVING</code> clauses. The SQL <code>OVER()</code> clause must follow the analytic function.
<code>window_partition_clause</code>	Divides the rows in the input table by a given list of columns or expressions. If the <code>window_partition_clause</code> is omitted, all input rows are treated as a single partition. See <code>window_partition_clause</code> (page 130).
<code>window_order_clause</code>	Sorts the rows specified by the <code>window_partition_clause</code> and supplies an ordered set of rows to the analytic function. See <code>window_order_clause</code> (page 131).
<code>window_frame_clause</code>	Allowed for some analytic function, the <code>window_frame_clause</code> represents a moving window, defined in the analytic <code>OVER()</code> clause, and specifies the beginning and end of the window relative to the current row. See <code>window_frame_clause</code> . (page 133)

### Notes

Analytic functions:

- Require the `OVER()` clause. However, depending on the analytic function, the `window_frame_clause` and `window_order_clause` might not apply.

**Note:** When used with analytic aggregate functions, `OVER()` may be used without supplying any of the windowing clauses; in this case, the aggregate returns the same aggregated value for each row of the result set.

- Are allowed only in the `SELECT` and `ORDER BY` clauses.

- Can be used in a subquery or in the parent query.
- Cannot be nested; for example, the following is not allowed:  
=> SELECT MEDIAN(RANK() OVER(ORDER BY sal) OVER() .

### See Also

**Performance Optimization for Analytic Sort Computation** (page 178)

Using SQL Analytics in the Programmer's Guide

Named Windows

## window\_partition\_clause

The `window_partition_clause` is an optional clause that, when specified, divides the rows in the input by a given list of columns or expressions. If the clause is omitted, all input rows are treated as a single partition. Window partitioning is similar to a GROUP BY operation, except the function returns only one result row per input row.

The analytic function is computed per partition and starts over again (resets) at the beginning of each subsequent partition. The `window_partition_clause` is specified within the `OVER()` clause.

### Syntax

```
PARTITION BY expression [ , ... ]
```

### Parameters

<code><i>expression</i></code>	Expression to sort the partition on. May involve columns, constants or an arbitrary expression formed on columns.
--------------------------------	---

### Sample schema

The examples in this topic use the following schema:

```
=> CREATE TABLE allsales(  
    state VARCHAR(20),  
    name VARCHAR(20),  
    sales INT);  
=> INSERT INTO allsales VALUES('MA', 'A', 60);  
=> INSERT INTO allsales VALUES('NY', 'B', 20);  
=> INSERT INTO allsales VALUES('NY', 'C', 15);  
=> INSERT INTO allsales VALUES('MA', 'D', 20);  
=> INSERT INTO allsales VALUES('MA', 'E', 50);  
=> INSERT INTO allsales VALUES('NY', 'F', 40);  
=> INSERT INTO allsales VALUES('MA', 'G', 10);  
=> COMMIT;
```

Create the example allsales table, insert the data, and query the table:

```
=> SELECT * FROM allsales;
state | name | sales
-----+-----+-----
MA    | A    | 60
NY    | B    | 20
NY    | C    | 15
MA    | D    | 20
MA    | E    | 50
NY    | F    | 40
MA    | G    | 10
(7 rows)
```

## Examples

The first example uses the analytic function `MEDIAN` to partition the results by state and then calculate the median of sales:

```
=> SELECT state, name, sales, MEDIAN(sales)
      OVER (PARTITION BY state) AS MEDIAN from allsales;
state | name | sales | MEDIAN
-----+-----+-----+-----
NY    | C    | 15    | 20
NY    | B    | 20    | 20
NY    | F    | 40    | 20
MA    | G    | 10    | 35
MA    | D    | 20    | 35
MA    | E    | 50    | 35
MA    | A    | 60    | 35
(7 rows)
```

**Note:** In the above results, notice the two partitions for MA and NY under the `state` column.

The next example calculates the median of total sales among states. Note that when you use `OVER ()` with no parameters, there is one partition, the entire input:

```
=> SELECT state, SUM(sales), MEDIAN(SUM(sales))
      OVER () AS MEDIAN FROM allsales GROUP BY state;
state | SUM | MEDIAN
-----+-----+-----
NY    | 75  | 107.5
MA    | 140 | 107.5
(2 rows)
```

## window\_order\_clause

Sorts the rows specified by the `window_partition_clause` (on page 130) and supplies an ordered set of rows to the `window_frame_clause` (if present), to the analytic function, or to both. The `window_order_clause` specifies whether data is returned in ascending or descending order and specifies where null values appear in the sorted result as either first or last. The ordering of the data affects the results.

Using `ORDER BY` in an `OVER` clause changes the default window to `RANGE UNBOUNDED PRECEDING AND CURRENT ROW`.

The `window_order_clause` does not guarantee the order of the SQL result. Use the **SQL ORDER BY clause** (page 757) to guarantee the ordering of the final result set.

The `window_order_clause` is part of the `OVER()` clause.

## Syntax

```
ORDER BY expression
... [ { ASC | DESC } ]
... [ NULLS { FIRST | LAST | AUTO } ] [,expression ...]
```

## Parameters

<code><i>expression</i></code>	Expression to sort the partition on. May involve columns, constants or an arbitrary expression formed on columns.
ASC   DESC	Specifies the ordering sequence as ascending (default) or descending.
NULLS { FIRST   LAST   AUTO }	Indicates the position of nulls in the ordered sequence as either first or last. The order makes nulls compare either high or low with respect to non-null values.  If the sequence is specified as ascending order, <code>ASC NULLS FIRST</code> implies that nulls are smaller than other non-null values. <code>ASC NULLS LAST</code> implies that nulls are larger than non-null values. The opposite is true for descending order. If you specify <code>NULLS AUTO</code> , Vertica chooses the most efficient placement of nulls (for example, either <code>NULLS FIRST</code> or <code>NULLS LAST</code> ) based on your query. The default is <code>ASC NULLS LAST</code> and <code>DESC NULLS FIRST</code> . See also <b>Performance Optimization for Analytic Sort Computation</b> (page 178).

The following list shows the default ordering, with bold clauses to indicate what is implicit:

- ORDER BY `column1` = ORDER BY a **ASC NULLS LAST**
- ORDER BY `column1 ASC` = ORDER BY a **ASC NULLS LAST**
- ORDER BY `column1 DESC` = ORDER BY a **DESC NULLS FIRST**

The placement of the `ORDER BY` clause might not guarantee the final result order. For example, the `window_order_clause` is different from the final `ORDER BY` in that the `window_order_clause` specifies the order within each partition and affects the result of the analytic calculation; it does not guarantee the order of the SQL result. Use the **SQL ORDER BY clause** (page 757) to guarantee the ordering of the final result set. See also Null Placement.

The following examples continue with the sample schema introduced in the **window\_partition\_clause** (page 130) topic.

### Example 1

In this example, the query orders the sales inside each sales partition:

```
=> SELECT state, sales, name, RANK()
      OVER (PARTITION BY state
```

### Example 2

In this example, the final `ORDER BY` clause sorts the results by name:

```
=> SELECT state, sales, name, RANK()
      OVER (PARTITION BY state
```

```

ORDER BY sales) AS RANK
FROM allsales;

state | sales | name | RANK
-----+-----+-----+-----
MA    |    10 | G    |    1
MA    |    20 | D    |    2
MA    |    50 | E    |    3
MA    |    60 | A    |    4
NY    |    15 | C    |    1
NY    |    20 | B    |    2
NY    |    40 | F    |    3
(7 rows)

```

```

ORDER BY sales) AS RANK
FROM allsales ORDER BY name;

state | sales | name | RANK
-----+-----+-----+-----
MA    |    60 | A    |    4
NY    |    20 | B    |    2
NY    |    15 | C    |    1
MA    |    20 | D    |    2
MA    |    50 | E    |    3
NY    |    40 | F    |    3
MA    |    10 | G    |    1
(7 rows)

```

## window\_frame\_clause

Allowed for some analytic functions, the `window_frame_clause` specifies the beginning and end of the window relative to the current row. Each analytic function is computed based on the data within the window frame boundaries. As Vertica computes an analytic function for each row, the window slides according to the `window_frame_clause`, and rows are excluded or included based on the position (`ROWS`) or value (`RANGE`) relative to the current row. The `CURRENT ROW` is the next row for which the analytic function computes results.

**Note:** If you omit the window frame clause, the default window is `RANGE UNBOUNDED PRECEDING AND CURRENT ROW`.

## Syntax

```

{ ROWS | RANGE }
{
  {
    BETWEEN
    { UNBOUNDED PRECEDING
    | CURRENT ROW
    | constant-value { PRECEDING | FOLLOWING }
    }
    AND
    { UNBOUNDED FOLLOWING
    | CURRENT ROW
    | constant-value { PRECEDING | FOLLOWING }
    }
  }
|
  {
    { UNBOUNDED PRECEDING
    | CURRENT ROW
    | constant-value PRECEDING
    }
  }
}

```

## Parameters

ROWS   RANGE	<p>The <code>ROWS</code> and <code>RANGE</code> keywords define the window frame type.</p> <p><code>ROWS</code> specifies a window as a physical offset and defines the window's start and end point by the number of rows before or after the current row. The value can be <code>INTEGER</code> data type only.</p> <p><code>RANGE</code> specifies the window as a logical offset, such as time. The range value must match the <code>window_order_clause</code> data type, which can be <code>NUMERIC</code>, <code>DATE/TIME</code>, <code>FLOAT</code> or <code>INTEGER</code>.</p> <p><b>Note:</b> The value returned by an analytic function with a logical offset is always deterministic. However, the value returned by an analytic function with a physical offset could produce nondeterministic results unless the ordering expression results in a unique ordering. You might have to specify multiple columns in the <code>window_order_clause</code> to achieve this unique ordering.</p>
BETWEEN ... AND	<p>Specifies a start point and end point for the window. The first expression (before <code>AND</code>) defines the start point and the second expression (after <code>AND</code>) defines the end point.</p> <p><b>Note:</b> If you use the keyword <code>BETWEEN</code>, you must also use <code>AND</code>.</p>
UNBOUNDED PRECEDING	<p>Indicates that the window starts at the first row of the partition. This start-point specification cannot be used as an end-point specification.</p>
UNBOUNDED FOLLOWING	<p>Indicates that the window ends at the last row of the partition. This end-point specification cannot be used as a start-point specification.</p>
CURRENT ROW	<p>As a start point, <code>CURRENT ROW</code> specifies that the window begins at the current row or value, depending on whether you have specified <code>ROW</code> or <code>RANGE</code>, respectively. In this case, the end point cannot be <i>constant-value</i> <code>PRECEDING</code>.</p> <p>As an end point, <code>CURRENT ROW</code> specifies that the window ends at the current row or value, depending on whether you have specified <code>ROW</code> or <code>RANGE</code>, respectively. In this case the start point cannot be <i>constant-value</i> <code>FOLLOWING</code>.</p>
<i>constant-value</i> { PRECEDING   FOLLOWING }	<p>For <code>RANGE</code> or <code>ROW</code>:</p> <ul style="list-style-type: none"> <li>▪ If <i>constant-value</i> <code>FOLLOWING</code> is the start point, the end point must be <i>constant-value</i> <code>FOLLOWING</code>.</li> <li>▪ If <i>constant-value</i> <code>PRECEDING</code> is the end point, the start point must be <i>constant-value</i> <code>PRECEDING</code>.</li> <li>▪ If you specify a logical window that is defined by a time interval in <code>NUMERIC</code> format, you might need to use conversion functions.</li> </ul> <p>If you specified <code>ROWS</code>:</p> <ul style="list-style-type: none"> <li>▪ <i>constant-value</i> is a physical offset. It must be a constant or expression and must evaluate to an <code>INTEGER</code> data type value.</li> <li>▪ If <i>constant-value</i> is part of the start point, it must evaluate to a</li> </ul>

	<p>row before the end point.</p> <p>If you specified <code>RANGE</code>:</p> <ul style="list-style-type: none"> <li>▪ <i>constant-value</i> is a logical offset. It must be a constant or expression that evaluates to a positive numeric value or an <code>INTERVAL</code> literal.</li> <li>▪ If <i>constant-value</i> evaluates to a <code>NUMERIC</code> value, the <code>ORDER BY</code> column type must be a <code>NUMERIC</code> data type..</li> <li>▪ If the <i>constant-value</i> evaluates to an <code>INTERVAL DAY TO SECOND</code> subtype, the <code>ORDER BY</code> column type can only be <code>TIMESTAMP</code>, <code>TIME</code>, <code>DATE</code>, or <code>INTERVAL DAY TO SECOND</code>.</li> <li>▪ If the <i>constant-value</i> evaluates to an <code>INTERVAL YEAR TO MONTH</code>, the <code>ORDER BY</code> column type can only be <code>TIMESTAMP</code>, <code>DATE</code>, or <code>INTERVAL YEAR TO MONTH</code>.</li> <li>▪ You can specify only one expression in the <code>window_order_clause</code>.</li> </ul>
--	--

## named\_windows

You can avoid typing long `OVER()` clause syntax by naming a window using the `WINDOW` clause, which takes the following form:

```
WINDOW window_name AS ( window_definition_clause );
```

In the following example, `RANK()` and `DENSE_RANK()` use the partitioning and ordering specifications in the window definition for `w`:

```
=> SELECT RANK() OVER w , DENSE_RANK() OVER w
       FROM employee_dimension
       WINDOW w AS (PARTITION BY employee_region ORDER BY annual_salary);
```

Though analytic functions can reference a named window to inherit the `window_partition_clause` (page 130), you can define your own `window_order_clause` (page 131); for example:

```
=> SELECT RANK() OVER(w ORDER BY annual_salary ASC) ,
       DENSE_RANK() OVER(w ORDER BY annual_salary DESC)
       FROM employee_dimension
       WINDOW w AS (PARTITION BY employee_region);
```

### Notes:

- The `window_partition_clause` is defined in the named window specification, not in the `OVER()` clause.
- The `OVER()` clause can specify its own `window_order_clause` only if the `window_definition_clause` did not already define it. For example, if the second example above is rewritten as follows, the system returns an error:

```
=> SELECT RANK() OVER(w ORDER BY annual_salary ASC) ,
```

```
DENSE_RANK() OVER(w ORDER BY annual_salary DESC)
FROM employee_dimension
WINDOW w AS (PARTITION BY employee_region ORDER BY annual_salary);
ERROR: cannot override ORDER BY clause of window "w"
```

- A window definition cannot contain a `window_frame_clause`.
- Each window defined in the `window_definition_clause` must have a unique name. You can reference window names within their scope only. For example, because named window `w1` below is defined before `w2`, `w2` is within the scope of `w1`:

```
=> SELECT RANK() OVER(w1 ORDER BY sal DESC)
       RANK() OVER w2
FROM EMP AS
WINDOW w1 AS (PARTITION BY deptno), w2 AS (w1 ORDER BY sal);
```

## AVG [Analytic]

Computes an average of an expression in a group within a window.

### Behavior Type

Immutable

### Syntax

```
AVG ( expression ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )
```

### Parameters

<i>expression</i>	The value whose average is calculated over a set of rows. Can be any expression resulting in <code>DOUBLE PRECISION</code> .
<code>OVER(...)</code>	See <i>Analytic Functions</i> . (page 128)

### Notes

`AVG()` takes as an argument any numeric data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the argument's numeric data type.

### Examples

The following query finds the sales for that calendar month and returns a running/cumulative average (sometimes called a moving average) using the default window of `RANGE UNBOUNDED PRECEDING AND CURRENT ROW`:

```
=> SELECT calendar_month_number_in_year, SUM(product_price) AS sales,
       AVG(SUM(product_price)) OVER (ORDER BY calendar_month_number_in_year)
FROM product_dimension, date_dimension, inventory_fact
WHERE date_dimension.date_key = inventory_fact.date_key
AND product_dimension.product_key = inventory_fact.product_key
GROUP BY calendar_month_number_in_year;
```

calendar_month_number_in_year	sales	?column?
1	23869547	23869547
2	19604661	21737104
3	22877913	22117373.6666667
4	22901263	22313346
5	23670676	22584812
6	22507600	22571943.3333333
7	21514089	22420821.2857143
8	24860684	22725804.125
9	21687795	22610469.7777778
10	23648921	22714314.9
11	21115910	22569005.3636364
12	24708317	22747281.3333333

(12 rows)

To return a moving average that is not a running (cumulative) average, the window should specify ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING:

```
=> SELECT calendar_month_number_in_year, SUM(product_price) AS sales,
AVG(SUM(product_price)) OVER (ORDER BY calendar_month_number_in_year
    ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)
FROM product_dimension, date_dimension, inventory_fact
WHERE date_dimension.date_key = inventory_fact.date_key
AND product_dimension.product_key = inventory_fact.product_key
GROUP BY calendar_month_number_in_year;
```

### See Also

**AVG** (page 110) aggregate function

**COUNT** (page 139) and **SUM** (page 173) analytic functions

Using SQL Analytics in the Programmer's Guide

## CONDITIONAL\_CHANGE\_EVENT [Analytic]

Assigns an event window number to each row, starting from 0, and increments by 1 when the result of evaluating the argument expression on the current row differs from that on the previous row.

### Behavior Type

Immutable

### Syntax

```
CONDITIONAL_CHANGE_EVENT ( expression ) OVER (
... [ window_partition_clause (page 130) ]
... window_order_clause (page 131) )
```

### Parameters

<i>expression</i>	Is a SQL scalar expression that is evaluated on an input record. The result of <i>expression</i> can be of any data type.
-------------------	---

OVER(...)	See <i>Analytic Functions</i> . (page 128)
-----------	--

**Notes**

The analytic `window_order_clause` is required but the `window_partition_clause` is optional.

**Example**

```
=> SELECT CONDITIONAL_CHANGE_EVENT (bid)
      OVER (PARTITION BY symbol ORDER BY ts) AS cce
      FROM TickStore;
```

The system returns an error when no `ORDER BY` is present:

```
=> SELECT CONDITIONAL_CHANGE_EVENT (bid)
      OVER (PARTITION BY symbol) AS cce
      FROM TickStore;
```

```
ERROR: conditional_change_event must contain an ORDER BY clause within
its analytic clause
```

For more examples, see Event-based Windows in the Programmer's Guide.

**See Also**

**CONDITIONAL\_TRUE\_EVENT** (page 138)

**ROW\_NUMBER** (page 168)

Using Time Series Analytics and Event-based Windows in the Programmer's Guide

**CONDITIONAL\_TRUE\_EVENT [Analytic]**

Assigns an event window number to each row, starting from 0, and increments the number by 1 when the result of the boolean argument expression evaluates true. For example, given a sequence of values for column `a`:

```
( 1, 2, 3, 4, 5, 6 )
```

```
CONDITIONAL_TRUE_EVENT (a > 3) returns 0, 0, 0, 1, 2, 3.
```

**Behavior Type:**

Immutable

**Syntax**

```
CONDITIONAL_TRUE_EVENT ( boolean-expression ) OVER
... ( [ window_partition_clause (page 130) ]
... window_order_clause (page 131) )
```

**Parameters**

<i>boolean-expression</i>	Is a SQL scalar expression that is evaluated on an input record. The result of <i>boolean-expression</i> is boolean type.
---------------------------	---

OVER(...)	See <b>Analytic Functions</b> (page 128).
-----------	---

## Notes

The analytic `window_order_clause` is required but the `window_partition_clause` is optional.

## Example

```
=> SELECT CONDITIONAL_TRUE_EVENT (bid > 10.6)
      OVER(PARTITION BY bid ORDER BY ts) AS cte
FROM Tickstore;
```

The system returns an error if the `ORDER BY` clause is omitted:

```
=> SELECT CONDITIONAL_TRUE_EVENT (bid > 10.6)
      OVER(PARTITION BY bid) AS cte
FROM Tickstore;
ERROR: conditional_true_event must contain an ORDER BY clause within its
analytic clause
```

For more examples, see Event-based Windows in the Programmer's Guide.

## See Also

**CONDITIONAL\_CHANGE\_EVENT** (page 137)

Using Time Series Analytics and Event-based Windows in the Programmer's Guide

## COUNT [Analytic]

Counts occurrences within a group within a window. If you specify `*` or some non-null constant, `COUNT ()` counts all rows.

## Behavior Type

Immutable

## Syntax

```
COUNT ( expression ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )
```

## Parameters

<i>expression</i>	Returns the number of rows in each group for which the <i>expression</i> is not null. Can be any expression resulting in BIGINT.
OVER(...)	See <b>Analytic Functions</b> . (page 128)

## Example

The following query finds the number of employees who make less than or equivalent to the hourly rate of the current employee. The query returns a running/cumulative average (sometimes called a moving average) using the default window of RANGE UNBOUNDED PRECEDING AND CURRENT ROW:

```
=> SELECT employee_last_name AS "last_name", hourly_rate, COUNT(*)  
       OVER (ORDER BY hourly_rate) AS moving_count from employee_dimension;  
last_name | hourly_rate | moving_count
```

last_name	hourly_rate	moving_count
Gauthier	6	4
Taylor	6	4
Jefferson	6	4
Nielson	6	4
McNulty	6.01	11
Robinson	6.01	11
Dobisz	6.01	11
Williams	6.01	11
Kramer	6.01	11
Miller	6.01	11
Wilson	6.01	11
Vogel	6.02	14
Moore	6.02	14
Vogel	6.02	14
Carcetti	6.03	19

...

To return a moving average that is not also a running (cumulative) average, the window should specify ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING:

```
=> SELECT employee_last_name AS "last_name", hourly_rate, COUNT(*)  
       OVER (ORDER BY hourly_rate ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)  
       AS moving_count from employee_dimension;
```

## See Also

**COUNT** (page 111) aggregate function

**AVG** (page 136) and **SUM** (page 173) analytic functions

Using SQL Analytics in the Programmer's Guide

## CUME\_DIST [Analytic]

Calculates the cumulative distribution, or relative rank, of the current row with regard to other rows in the same partition within a window.

`CUME_DIST()` returns a number greater than 0 and less than or equal to 1, where the number represents the relative position of the specified row within a group of *N* rows. For a row *x* (assuming `ASC` ordering), the `CUME_DIST` of *x* is the number of rows with values lower than or equal to the value of *x*, divided by the number of rows in the partition. In a group of three rows, for example, the cumulative distribution values returned would be 1/3, 2/3, and 3/3.

**Note:** Because the result for a given row depends on the number of rows preceding that row in the same partition, Vertica recommends that you always specify a `window_order_clause` when you call this function.

## Behavior Type

Immutable

## Syntax

```
CUME_DIST ( ) OVER (
... [ window_partition_clause (page 130) ]
... window_order_clause (page 131) )
```

## Parameters

OVER(...)	See <i>Analytic Functions</i> . (page 128)
-----------	--

## Notes

The analytic `window_order_clause` is required but the `window_partition_clause` is optional.

## Examples

The following example returns the cumulative distribution of sales for different transaction types within each month of the first quarter.

```
=> SELECT calendar_month_name AS month, tender_type, SUM(sales_quantity),
        CUME_DIST()
        OVER (PARTITION BY calendar_month_name ORDER BY SUM(sales_quantity)) AS
CUME_DIST
FROM store.store_sales_fact JOIN date_dimension
USING(date_key) WHERE calendar_month_name IN ('January','February','March')
AND tender_type NOT LIKE 'Other'
GROUP BY calendar_month_name, tender_type;
```

month	tender_type	SUM	CUME_DIST
March	Credit	469858	0.25
March	Cash	470449	0.5
March	Check	473033	0.75
March	Debit	475103	1
January	Cash	441730	0.25
January	Debit	443922	0.5
January	Check	446297	0.75
January	Credit	450994	1
February	Check	425665	0.25
February	Debit	426726	0.5
February	Credit	430010	0.75
February	Cash	430767	1

(12 rows)

**See Also****PERCENT\_RANK** (page 160)**PERCENTILE\_DISC** (page 164)

Using SQL Analytics in the Programmer's Guide

**DENSE\_RANK [Analytic]**

Computes the relative rank of each row returned from a query with respect to the other rows, based on the values of the expressions in the window `ORDER BY` clause.

The data within a group is sorted by the `ORDER BY` clause and then a numeric ranking is assigned to each row in turn starting with 1 and continuing from there. The rank is incremented every time the values of the `ORDER BY` expressions change. Rows with equal values receive the same rank (nulls are considered equal in this comparison). A `DENSE_RANK()` function returns a ranking number without any gaps, which is why it is called "DENSE."

**Behavior Type**

Immutable

**Syntax**

```
DENSE_RANK ( ) OVER (
... [ window_partition_clause (page 130) ]
... window_order_clause (page 131) )
```

**Parameters**

OVER(...)	See <b>Analytic Functions</b> . (page 128)
-----------	--

**Notes**

- The analytic `window_order_clause` is required but the `window_partition_clause` is optional.
- The ranks are consecutive integers beginning with 1. The largest rank value is the number of unique values returned by the query.
- The primary difference between `DENSE_RANK()` and `RANK()` (page 166) is that `RANK` leaves gaps when ranking records whereas `DENSE_RANK` leaves no gaps. For example, N records occupy a particular position (say, a tie for rank X), `RANK` assigns all those records with rank X and skips the next N ranks, therefore the next assigned rank is X+N. `DENSE_RANK` places all the records in that position only—it does not skip any ranks.

If there is a tie at the third position with two records having the same value, `RANK` and `DENSE_RANK` place both the records in the third position, but `RANK` places the next record at the fifth position, while `DENSE_RANK` places the next record at the fourth position.

- If you omit `NULLS FIRST | LAST | AUTO`, the ordering of the `NULL` values depends on the `ASC` or `DESC` arguments. `NULL` values are considered larger than any other value. If the ordering sequence is `ASC`, then nulls appear last; nulls appear first otherwise. Nulls are considered equal to other nulls and, therefore, the order in which nulls are presented is non-deterministic.

## Examples

The following example shows the difference between `RANK` and `DENSE_RANK` when ranking customers by their annual income. Notice that `RANK` has a tie at 10 and skips 11, while `DENSE_RANK` leaves no gaps in the ranking sequence:

```
=> SELECT customer_name, SUM(annual_income),
       RANK () OVER (ORDER BY TO_CHAR(SUM(annual_income),'100000') DESC) rank,
       DENSE_RANK () OVER (ORDER BY TO_CHAR(SUM(annual_income),'100000') DESC)
dense_rank
```

```
FROM customer_dimension GROUP BY customer_name LIMIT 15;
```

customer_name	sum	rank	dense_rank
Brian M. Garnett	99838	1	1
Tanya A. Brown	99834	2	2
Tiffany P. Farmer	99826	3	3
Jose V. Sanchez	99673	4	4
Marcus D. Rodriguez	99631	5	5
Alexander T. Nguyen	99604	6	6
Sarah G. Lewis	99556	7	7
Ruth Q. Vu	99542	8	8
Theodore T. Farmer	99532	9	9
Daniel P. Li	99497	<b>10</b>	<b>10</b>
Seth E. Brown	99497	<b>10</b>	<b>10</b>
Matt X. Gauthier	99402	<b>12</b>	<b>11</b>
Rebecca W. Lewis	99296	13	12
Dean L. Wilson	99276	14	13
Tiffany A. Smith	99257	15	14

(15 rows)

## See Also

**RANK** (page 166)

Using SQL Analytics in the Programmer's Guide

## EXPONENTIAL\_MOVING\_AVERAGE [Analytic]

Calculates the exponential moving average of expression E with smoothing factor X.

The exponential moving average (EMA) is calculated by adding the previous EMA value to the current data point scaled by the smoothing factor, as in the following formula, where `EMA0` is the previous row's EMA value, `X` is the smoothing factor, and `E` is the current data point:  $EMA = EMA0 + (X * (E - EMA0))$ .

## Behavior Type

Immutable

## Syntax

```
EXPONENTIAL_MOVING_AVERAGE ( E , X ) OVER (
... [ window_partition_clause (page 130) ]
... window_order_clause (page 131) )
```

## Parameters

<i>E</i>	The value whose average is calculated over a set of rows. Can be INTEGER, FLOAT or NUMERIC type and must be a constant.
<i>X</i>	A positive FLOAT value between 0 and 1 that is used as the smoothing factor.
OVER(...)	See <b>Analytic Functions</b> . (page 128)

## Notes

- The analytic `window_order_clause` is required but the `window_partition_clause` is optional.
- There is no [Aggregate] equivalent of this function because of its unique semantics.
- `EXPONENTIAL_MOVING_AVERAGE()` is different from a simple moving average in that it provides a more stable picture of changes to data over time.
- The `EXPONENTIAL_MOVING_AVERAGE()` function also works at the row level; for example, it assumes the data in a given column is sampled at uniform intervals. If the users' data points are sampled at non-uniform intervals, they should run the time series gap filling and interpolation (GFI) operations before `EMA()`. See the Example section below.

## Examples

The following example uses time series gap filling and interpolation (GFI) first in a subquery, and then performs an `EXPONENTIAL_MOVING_AVERAGE` operation on the subquery result.

Create a simple 4-column table:

```
=> CREATE TABLE ticker(
    time TIMESTAMP,
    symbol VARCHAR(8),
    bid1 FLOAT,
    bid2 FLOAT );
```

Now insert some data, including nulls, so GFI can do its interpolation and gap filling:

```
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:00', 'ABC', 60.45, 60.44);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:01', 'ABC', 60.49, 65.12);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:02', 'ABC', 57.78, 59.25);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:03', 'ABC', null, 65.12);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:04', 'ABC', 67.88, null);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:00', 'XYZ', 47.55, 40.15);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:01', 'XYZ', 44.35, 46.78);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:02', 'XYZ', 71.56, 75.78);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:03', 'XYZ', 85.55, 70.21);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:04', 'XYZ', 45.55, 58.65);
=> COMMIT;
```

**Note:** During gap filling and interpolation, Vertica takes the closest non null value on either side of the time slice and uses that value. For example, if you use a linear interpolation scheme and you do not specify `IGNORE NULLS`, and your data has one real value and one null, the result is null. If the value on either side is null, the result is null. See *When Time Series Data Contains Nulls in the Programmer's Guide* for details.

Query the table you just created to you can see the output:

```
=> SELECT * FROM ticker;
      time          | symbol | bid1 | bid2
-----+-----+-----+-----
2009-07-12 03:00:00 | ABC   | 60.45 | 60.44
2009-07-12 03:00:01 | ABC   | 60.49 | 65.12
2009-07-12 03:00:02 | ABC   | 57.78 | 59.25
2009-07-12 03:00:03 | ABC   |      | 65.12
2009-07-12 03:00:04 | ABC   | 67.88 |
2009-07-12 03:00:00 | XYZ   | 47.55 | 40.15
2009-07-12 03:00:01 | XYZ   | 44.35 | 46.78
2009-07-12 03:00:02 | XYZ   | 71.56 | 75.78
2009-07-12 03:00:03 | XYZ   | 85.55 | 70.21
2009-07-12 03:00:04 | XYZ   | 45.55 | 58.65
(10 rows)
```

The following query processes the first and last values that belong to each 2-second time slice in table `trades`' column `a`. The query then calculates the exponential moving average of expression `fv` and `lv` with a smoothing factor of 5%:

```
=> SELECT symbol, slice_time, fv, lv,
      EXPONENTIAL_MOVING_AVERAGE(fv, 0.5)
      OVER (PARTITION BY symbol ORDER BY slice_time) AS ema_first,
      EXPONENTIAL_MOVING_AVERAGE(lv, 0.5)
      OVER (PARTITION BY symbol ORDER BY slice_time) AS ema_last
FROM (
  SELECT symbol, slice_time,
         TS_FIRST_VALUE(bid1 IGNORE NULLS) as fv,
         TS_LAST_VALUE(bid2 IGNORE NULLS) AS lv
  FROM ticker TIMESERIES slice_time AS '2 seconds'
  OVER (PARTITION BY symbol ORDER BY time) ) AS sq;
```

```
symbol | slice_time          | fv  | lv  | ema_first | ema_last
-----+-----+-----+-----+-----+-----
ABC    | 2009-07-12 03:00:00 | 60.45 | 65.12 | 60.45    | 65.12
ABC    | 2009-07-12 03:00:02 | 57.78 | 65.12 | 59.115   | 65.12
ABC    | 2009-07-12 03:00:04 | 67.88 | 65.12 | 63.4975  | 65.12
XYZ    | 2009-07-12 03:00:00 | 47.55 | 46.78 | 47.55    | 46.78
XYZ    | 2009-07-12 03:00:02 | 71.56 | 70.21 | 59.555   | 58.495
XYZ    | 2009-07-12 03:00:04 | 45.55 | 58.65 | 52.5525  | 58.5725
(6 rows)
```

## See Also

***TIMESERIES Clause*** (page 761)

Using Time Series Analytics and Using SQL Analytics in the Programmer's Guide

## FIRST\_VALUE [Analytic]

Returns values of the expression from the first row of a window for the current row. If no window is specified for the current row, the default window is UNBOUNDED PRECEDING AND CURRENT ROW.

### Behavior Type

Immutable

### Syntax

```
FIRST_VALUE ( expression [ IGNORE NULLS ] ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )
```

### Parameters

<i>expression</i>	Is the expression to evaluate; for example, a constant, column, nonanalytic function, function expression, or expressions involving any of these.
IGNORE NULLS	Returns the first non-null value in the set, or NULL if all values are NULL.
OVER(...)	See <b>Analytic Functions</b> . (page 128)

### Notes

- The FIRST\_VALUE() function lets you select a table's first value (determined by the window\_order\_clause) without having to use a self join. This function is useful when you want to use the first value as a baseline in calculations.
- Vertica recommends that you use FIRST\_VALUE with the window\_order\_clause to produce deterministic results.
- If the first value in the set is null, then the function returns NULL unless you specify IGNORE NULLS. If you specify IGNORE NULLS, FIRST\_VALUE returns the first non-null value in the set, or NULL if all values are null.

### Examples

The following query, which asks for the first value in the partitioned day of week, illustrates the potential nondeterministic nature of the FIRST\_VALUE function:

```
=> SELECT calendar_year, date_key, day_of_week, full_date_description,
FIRST_VALUE(full_date_description)
OVER(PARTITION BY calendar_month_number_in_year ORDER BY day_of_week) AS "first_value"
FROM date_dimension
WHERE calendar_year=2003 AND calendar_month_number_in_year=1;
```

The first value returned is January 31, 2003; however, the next time the same query is run, the first value could be January 24 or January 3, or the 10th or 17th. The reason is because the analytic `ORDER BY` column (`day_of_week`) returns rows that contain ties (multiple Fridays). These repeated values make the `ORDER BY` evaluation result nondeterministic, because rows that contain ties can be ordered in any way, and any one of those rows qualifies as being the first value of `day_of_week`.

```

calendar_year | date_key | day_of_week | full_date_description | first_value
-----+-----+-----+-----+-----
2003 |      31 | Friday    | January 31, 2003    | January 31, 2003
2003 |      24 | Friday    | January 24, 2003    | January 31, 2003
2003 |       3 | Friday    | January 3, 2003     | January 31, 2003
2003 |      10 | Friday    | January 10, 2003   | January 31, 2003
2003 |      17 | Friday    | January 17, 2003   | January 31, 2003
2003 |       6 | Monday      | January 6, 2003      | January 31, 2003
2003 |      27 | Monday      | January 27, 2003     | January 31, 2003
2003 |      13 | Monday      | January 13, 2003     | January 31, 2003
2003 |      20 | Monday      | January 20, 2003     | January 31, 2003
2003 |      11 | Saturday    | January 11, 2003     | January 31, 2003
2003 |      18 | Saturday    | January 18, 2003     | January 31, 2003
2003 |      25 | Saturday    | January 25, 2003     | January 31, 2003
2003 |       4 | Saturday    | January 4, 2003      | January 31, 2003
2003 |      12 | Sunday      | January 12, 2003     | January 31, 2003
2003 |      26 | Sunday      | January 26, 2003     | January 31, 2003
2003 |       5 | Sunday      | January 5, 2003      | January 31, 2003
2003 |      19 | Sunday      | January 19, 2003     | January 31, 2003
2003 |      23 | Thursday    | January 23, 2003     | January 31, 2003
2003 |       2 | Thursday    | January 2, 2003      | January 31, 2003
2003 |       9 | Thursday    | January 9, 2003      | January 31, 2003
2003 |      16 | Thursday    | January 16, 2003     | January 31, 2003
2003 |      30 | Thursday    | January 30, 2003     | January 31, 2003
2003 |      21 | Tuesday     | January 21, 2003     | January 31, 2003
2003 |      14 | Tuesday     | January 14, 2003     | January 31, 2003
2003 |       7 | Tuesday     | January 7, 2003      | January 31, 2003
2003 |      28 | Tuesday     | January 28, 2003     | January 31, 2003
2003 |      22 | Wednesday   | January 22, 2003     | January 31, 2003
2003 |      29 | Wednesday   | January 29, 2003     | January 31, 2003
2003 |      15 | Wednesday   | January 15, 2003     | January 31, 2003
2003 |       1 | Wednesday   | January 1, 2003      | January 31, 2003
2003 |       8 | Wednesday   | January 8, 2003      | January 31, 2003

```

(31 rows)

**Note:** The `day_of_week` results are returned in alphabetical order because of lexical rules. The fact that each day does not appear ordered by the 7-day week cycle (for example, starting with Sunday followed by Monday, Tuesday, and so on) has no affect on results.

To return deterministic results, modify the query so that it performs its analytic `ORDER BY` operations on a **unique** field, such as `date_key`:

```

=> SELECT calendar_year, date_key, day_of_week, full_date_description,
       FIRST_VALUE(full_date_description) OVER
         (PARTITION BY calendar_month_number_in_year ORDER BY date_key) AS "first_value"
FROM date_dimension WHERE calendar_year=2003;

```

Notice that the results return a first value of January 1 for the January partition and the first value of February 1 for the February partition. Also, there are no ties in the `full_date_description` column:

<code>calendar_year</code>	<code>date_key</code>	<code>day_of_week</code>	<code>full_date_description</code>	<code>first_value</code>
2003	<b>1</b>	<b>Wednesday</b>	<b>January 1, 2003</b>	<b>January 1, 2003</b>
2003	2	Thursday	January 2, 2003	January 1, 2003
2003	3	Friday	January 3, 2003	January 1, 2003
2003	4	Saturday	January 4, 2003	January 1, 2003
2003	5	Sunday	January 5, 2003	January 1, 2003
2003	6	Monday	January 6, 2003	January 1, 2003
2003	7	Tuesday	January 7, 2003	January 1, 2003
2003	8	Wednesday	January 8, 2003	January 1, 2003
2003	9	Thursday	January 9, 2003	January 1, 2003
2003	10	Friday	January 10, 2003	January 1, 2003
2003	11	Saturday	January 11, 2003	January 1, 2003
2003	12	Sunday	January 12, 2003	January 1, 2003
2003	13	Monday	January 13, 2003	January 1, 2003
2003	14	Tuesday	January 14, 2003	January 1, 2003
2003	15	Wednesday	January 15, 2003	January 1, 2003
2003	16	Thursday	January 16, 2003	January 1, 2003
2003	17	Friday	January 17, 2003	January 1, 2003
2003	18	Saturday	January 18, 2003	January 1, 2003
2003	19	Sunday	January 19, 2003	January 1, 2003
2003	20	Monday	January 20, 2003	January 1, 2003
2003	21	Tuesday	January 21, 2003	January 1, 2003
2003	22	Wednesday	January 22, 2003	January 1, 2003
2003	23	Thursday	January 23, 2003	January 1, 2003
2003	24	Friday	January 24, 2003	January 1, 2003
2003	25	Saturday	January 25, 2003	January 1, 2003
2003	26	Sunday	January 26, 2003	January 1, 2003
2003	27	Monday	January 27, 2003	January 1, 2003
2003	28	Tuesday	January 28, 2003	January 1, 2003
2003	29	Wednesday	January 29, 2003	January 1, 2003
2003	30	Thursday	January 30, 2003	January 1, 2003
2003	31	Friday	January 31, 2003	January 1, 2003
2003	<b>32</b>	<b>Saturday</b>	<b>February 1, 2003</b>	<b>February 1, 2003</b>
2003	33	Sunday	February 2, 2003	February 1, 2003

...

(365 rows)

**See Also**

***LAST\_VALUE*** (page 152)

***TIME\_SLICE*** (page 225)

Using SQL Analytics in the Programmer's Guide

## LAG [Analytic]

Returns the value of the input expression at the given offset *before* the current row within a window.

### Behavior Type

Immutable

### Syntax

```
LAG ( expression [, offset ] [, default ] ) OVER (
... [ window_partition_clause (page 130) ]
... window_order_clause (page 131) )
```

### Parameters

<i>expression</i>	Is the expression to evaluate; for example, a constant, column, nonanalytic function, function expression, or expressions involving any of these.
<i>offset</i>	Is an optional parameter that defaults to 1 (the previous row). The <i>offset</i> parameter must be (or can be evaluated to) a constant positive integer.
<i>default</i>	Is NULL. This optional parameter is the value returned if <i>offset</i> falls outside the bounds of the table or partition. <b>Note:</b> The third input argument must be a constant value or an expression that can be evaluated to a constant; its data type is coercible to that of the first argument.
OVER (...)	See <b>Analytic Functions</b> . (page 128)

### Notes

- The analytic `window_order_clause` is required but the `window_partition_clause` is optional.
- The `LAG()` function returns values from the row before the current row, letting you access more than one row in a table at the same time. This is useful for comparing values when the relative positions of rows can be reliably known. It also lets you avoid the more costly self join, which enhances query processing speed.
- See **LEAD()** (page 153) for how to get the *next* rows.
- Analytic functions, such as `LAG()`, cannot be nested within aggregate functions.

### Examples

This example sums the current balance by date in a table and also sums the previous balance from the last day. Given the inputs that follow, the data satisfies the following conditions:

- For each `some_id`, there is exactly 1 row for each date represented by `month_date`.
- For each `some_id`, the set of dates is consecutive; that is, if there is a row for February 24 and a row for February 26, there would also be a row for February 25.
- Each `some_id` has the same set of dates.

```

=> CREATE TABLE balances (
    month_date DATE,
    current_bal INT,
    some_id INT);
=> INSERT INTO balances values ('2009-02-24', 10, 1);
=> INSERT INTO balances values ('2009-02-25', 10, 1);
=> INSERT INTO balances values ('2009-02-26', 10, 1);
=> INSERT INTO balances values ('2009-02-24', 20, 2);
=> INSERT INTO balances values ('2009-02-25', 20, 2);
=> INSERT INTO balances values ('2009-02-26', 20, 2);
=> INSERT INTO balances values ('2009-02-24', 30, 3);
=> INSERT INTO balances values ('2009-02-25', 20, 3);
=> INSERT INTO balances values ('2009-02-26', 30, 3);

```

Now run the LAG () function to sum the current balance for each date and sum the previous balance from the last day:

```

=> SELECT month_date,
    SUM(current_bal) as current_bal_sum,
    SUM(previous_bal) as previous_bal_sum FROM
    (SELECT month_date, current_bal,
    LAG(current_bal, 1, 0) OVER
    (PARTITION BY some_id ORDER BY month_date)
    AS previous_bal FROM balances) AS subQ
    GROUP BY month_date ORDER BY month_date;

```

month_date	current_bal_sum	previous_bal_sum
2009-02-24	60	0
2009-02-25	50	60
2009-02-26	60	50

(3 rows)

Using the same example data, the following query would not be allowed because LAG () is nested inside an aggregate function:

```

=> SELECT month_date,
    SUM(current_bal) as current_bal_sum,
    SUM(LAG(current_bal, 1, 0) OVER
    (PARTITION BY some_id ORDER BY month_date)) AS previous_bal_sum
    FROM some_table GROUP BY month_date ORDER BY month_date;

```

In the next example, which uses the VMart example database, the LAG () function first returns the annual income from the previous row, and then it calculates the difference between the income in the current row from the income in the previous row. Note: The vmart example database returns over 50,000 rows, so we'll limit the results to 20 records:

```

=> SELECT occupation, customer_key, customer_name, annual_income,
    LAG(annual_income, 1, 0) OVER (PARTITION BY occupation ORDER BY annual_income) AS prev_income,
    annual_income -
    LAG(annual_income, 1, 0) OVER (PARTITION BY occupation ORDER BY annual_income) AS difference
    FROM customer_dimension ORDER BY occupation, customer_key LIMIT 20;

```

occupation	customer_key	customer_name	annual_income	prev_income	difference
Accountant	15	Midori V. Peterson	692610	692535	75
Accountant	43	Midori S. Rodriguez	282359	280976	1383
Accountant	93	Robert P. Campbell	471722	471355	367
Accountant	102	Sam T. McNulty	901636	901561	75

Accountant	134	Martha B. Overstreet	705146	704335	811
Accountant	165	James C. Kramer	376841	376474	367
Accountant	225	Ben W. Farmer	70574	70449	125
Accountant	270	Jessica S. Lang	684204	682274	1930
Accountant	273	Mark X. Lampert	723294	722737	557
Accountant	295	Sharon K. Gauthier	29033	28412	621
Accountant	338	Anna S. Jackson	816858	815557	1301
Accountant	377	William I. Jones	915149	914872	277
Accountant	438	Joanna A. McCabe	147396	144482	2914
Accountant	452	Kim P. Brown	126023	124797	1226
Accountant	467	Meghan K. Carcetti	810528	810284	244
Accountant	478	Tanya E. Greenwood	639649	639029	620
Accountant	511	Midori P. Vogel	187246	185539	1707
Accountant	525	Alexander K. Moore	677433	677050	383
Accountant	550	Sam P. Reyes	735691	735355	336
Accountant	577	Robert U. Vu	616101	615439	662

(20 rows)

Continuing with the Vmart database, the next example uses both LEAD () and LAG () to return the third row after the salary in the current row and fifth salary before the salary in the current row.

```
=> SELECT hire_date, employee_key, employee_last_name,
        LEAD(hire_date, 1) OVER (ORDER BY hire_date) AS "next_hired" ,
        LAG(hire_date, 1) OVER (ORDER BY hire_date) AS "last_hired"
        FROM employee_dimension ORDER BY hire_date, employee_key;
```

hire_date	employee_key	employee_last_name	next_hired	last_hired
1956-04-11	2694	Farmer	1956-05-12	
1956-05-12	5486	Winkler	1956-09-18	1956-04-11
1956-09-18	5525	McCabe	1957-01-15	1956-05-12
1957-01-15	560	Greenwood	1957-02-06	1956-09-18
1957-02-06	9781	Bauer	1957-05-25	1957-01-15
1957-05-25	9506	Webber	1957-07-04	1957-02-06
1957-07-04	6723	Kramer	1957-07-07	1957-05-25
1957-07-07	5827	Garnett	1957-11-11	1957-07-04
1957-11-11	373	Reyes	1957-11-21	1957-07-07
1957-11-21	3874	Martin	1958-02-06	1957-11-11

(10 rows)

The following example specifies arguments that use different data types; for example annual\_income (INT) and occupation (VARCHAR). The query returns an error:

```
=> SELECT customer_key, customer_name, occupation, annual_income,
        LAG (annual_income, 1, occupation) OVER
          (PARTITION BY occupation ORDER BY customer_key) LAG1
        FROM customer_dimension ORDER BY 3, 1;
ERROR:  Third argument of lag could not be converted from type character varying
to type int8
HINT:  You may need to add explicit type cast.
```

## See Also

**LEAD** (page 153)

Using SQL Analytics in the Programmer's Guide

## LAST\_VALUE [Analytic]

Returns values of the expression from the last row of a window for the current row. If no window is specified for the current row, the default window is `UNBOUNDED PRECEDING AND CURRENT ROW`.

### Behavior Type

Immutable

### Syntax

```
LAST_VALUE ( expression [ IGNORE NULLS ] ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )
```

### Parameters

<i>expression</i>	Is the expression to evaluate; for example, a constant, column, nonanalytic function, function expression, or expressions involving any of these.
IGNORE NULLS	Returns the last non-null value in the set, or NULL if all values are NULL.
OVER(...)	See <b>Analytic Functions</b> . (page 128)

### Notes

- The `LAST_VALUE()` function lets you select a window's last value (determined by the `window_order_clause`), without having to use a self join. This function is useful when you want to use the last value as a baseline in calculations.
- `LAST_VALUE()` takes the last record from the partition after the analytic `window_order_clause`. The expression is then computed against the last record, and results are returned.
- Vertica recommends that you use `LAST_VALUE` with the `window_order_clause` to produce deterministic results.

**Note:** Due to default window semantics, `LAST_VALUE` does not always return the last value of a partition. If the `window_frame_clause` is omitted from the analytic clause, `LAST_VALUE` operates on this default window. Results, therefore, can seem non-intuitive because the function does not return the bottom of the current partition. It returns the bottom of the window, which continues to change along with the current input row being processed. If you want to return the last value of a partition, use `UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`.

- If the last value in the set is null, then the function returns `NULL` unless you specify `IGNORE NULLS`. If you specify `IGNORE NULLS`, `LAST_VALUE` returns the first non-null value in the set, or `NULL` if all values are null.
- For examples, see `FIRST_VALUE()` (page 146).

**See Also****FIRST\_VALUE** (page 146)**TIME\_SLICE** (page 225)

Using SQL for Analytics in the Programmer's Guide

**LEAD [Analytic]**Returns the value of the input expression at the given offset *after* the current row within a window.**Behavior Type**

Immutable

**Syntax**

```
LEAD ( expression [, offset ] [, default ] ) OVER (
... [ window_partition_clause (page 130) ]
... window_order_clause (page 131) )
```

**Parameters**

<i>expression</i>	Is the expression to evaluate; for example, a constant, column, nonanalytic function, function expression, or expressions involving any of these.
<i>offset</i>	Is an optional parameter that defaults to 1 (the next row). The <i>offset</i> parameter must be (or can be evaluated to) a constant positive integer.
<i>default</i>	Is NULL. This optional parameter is the value returned if <i>offset</i> falls outside the bounds of the table or partition. <b>Note:</b> The third input argument must be a constant value or an expression that can be evaluated to a constant; its data type is coercible to that of the first argument.
OVER(...)	See <b>Analytic Functions</b> . (page 128)

**Notes**

- The analytic *window\_order\_clause* is required but the *window\_partition\_clause* is optional.
- The LEAD() function returns values from the row after the current row, letting you access more than one row in a table at the same time. This is useful for comparing values when the relative positions of rows can be reliably known. It also lets you avoid the more costly self join, which enhances query processing speed.
- Analytic functions, such as LEAD(), cannot be nested within aggregate functions.

**Examples**

In this example, the LEAD() function finds the hire date of the employee hired just after the current row:

```
=> SELECT employee_region, hire_date, employee_key, employee_last_name,
       LEAD(hire_date, 1) OVER (PARTITION BY employee_region ORDER BY hire_date) AS "next_hired"
       FROM employee_dimension ORDER BY employee_region, hire_date, employee_key;
```

employee_region	hire_date	employee_key	employee_last_name	next_hired
East	1956-04-08	9218	Harris	<b>1957-02-06</b>
East	<b>1957-02-06</b>	7799	Stein	1957-05-25
East	1957-05-25	3687	Farmer	1957-06-26
East	1957-06-26	9474	Bauer	1957-08-18
East	1957-08-18	570	Jefferson	1957-08-24
East	1957-08-24	4363	Wilson	1958-02-17
East	1958-02-17	6457	McCabe	1958-06-26
East	1958-06-26	6196	Li	1958-07-16
East	1958-07-16	7749	Harris	1958-09-18
East	1958-09-18	9678	Sanchez	1958-11-10

(10 rows)

The next example uses both LEAD() and LAG() to return the third row after the salary in the current row and fifth salary before the salary in the current row.

```
=> SELECT hire_date, employee_key, employee_last_name,
       LEAD(hire_date, 1) OVER (ORDER BY hire_date) AS "next_hired" ,
       LAG(hire_date, 1) OVER (ORDER BY hire_date) AS "last_hired"
       FROM employee_dimension ORDER BY hire_date, employee_key;
```

hire_date	employee_key	employee_last_name	next_hired	last_hired
1956-04-11	2694	Farmer	1956-05-12	
1956-05-12	5486	Winkler	1956-09-18	1956-04-11
1956-09-18	5525	McCabe	1957-01-15	1956-05-12
1957-01-15	560	Greenwood	1957-02-06	1956-09-18
1957-02-06	9781	Bauer	1957-05-25	1957-01-15
1957-05-25	9506	Webber	1957-07-04	1957-02-06
1957-07-04	6723	Kramer	1957-07-07	1957-05-25
1957-07-07	5827	Garnett	1957-11-11	1957-07-04
1957-11-11	373	Reyes	1957-11-21	1957-07-07
1957-11-21	3874	Martin	1958-02-06	1957-11-11

(10 rows)

The following example returns employee name and salary, along with the next highest and lowest salaries.

```
=> SELECT employee_last_name, annual_salary,
       NVL(LEAD(annual_salary) OVER (ORDER BY annual_salary),
          MIN(annual_salary) OVER()) "Next Highest",
       NVL(LAG(annual_salary) OVER (ORDER BY annual_salary),
          MAX(annual_salary) OVER()) "Next Lowest"
       FROM employee_dimension;
```

employee_last_name	annual_salary	Next Highest	Next Lowest
Nielson	1200	1200	995533
Lewis	1200	1200	1200
Harris	1200	1202	1200
Robinson	1202	1202	1200

```
Garnett          |          1202 |          1202 |          1202
Weaver          |          1202 |          1202 |          1202
Nielson         |          1202 |          1202 |          1202
McNulty        |          1202 |          1204 |          1202
Farmer          |          1204 |          1204 |          1202
Martin         |          1204 |          1204 |          1204
(10 rows)
```

The next example returns, for each assistant director in the employees table, the hire date of the director hired just after the director on the current row. For example, Jackson was hired on 2007-12-28, and the next director hired was Bauer:

```
=> SELECT employee_last_name, hire_date,
        LEAD(hire_date, 1) OVER (ORDER BY hire_date DESC) as "NextHired"
   FROM employee_dimension WHERE job_title = 'Assistant Director';
employee_last_name | hire_date | NextHired
-----+-----+-----
Jackson           | 2007-12-28 | 2007-12-26
Bauer             | 2007-12-26 | 2007-12-11
Miller           | 2007-12-11 | 2007-12-07
Fortin           | 2007-12-07 | 2007-11-27
Harris           | 2007-11-27 | 2007-11-15
Goldberg         | 2007-11-15 |
(5 rows)
```

## See Also

**LAG** (page 149)

Using SQL for Analytics in the Programmer's Guide

## MAX [Analytic]

Returns the maximum value of an expression within a window. The return value is the same as the expression data type.

### Behavior Type

Immutable

### Syntax

```
MAX ( [ DISTINCT ] expression ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )
```

### Parameters

DISTINCT	Is meaningless in this context.
<i>expression</i>	Can be any expression for which the maximum value is calculated, typically a <b>column reference</b> (see " <b>Column References</b> " on page 46).
OVER(...)	See <b>Analytic Functions</b> . (page 128)

## Example

The following query computes the deviation between the employees' annual salary and the maximum annual salary in Massachusetts:

```
=> SELECT employee_state, annual_salary,
       MAX(annual_salary)
       OVER(PARTITION BY employee_state ORDER BY employee_key) max,
       annual_salary- MAX(annual_salary)
       OVER(PARTITION BY employee_state ORDER BY employee_key) diff
FROM employee_dimension
WHERE employee_state = 'MA';
```

employee_state	annual_salary	max	diff
MA	1918	995533	-993615
MA	2058	995533	-993475
MA	2586	995533	-992947
MA	2500	995533	-993033
MA	1318	995533	-994215
MA	2072	995533	-993461
MA	2656	995533	-992877
MA	2148	995533	-993385
MA	2366	995533	-993167
MA	2664	995533	-992869

(10 rows)

## See Also

**MAX** (page 116) aggregate function

**MIN** (page 158) analytic function

Using SQL Analytics in the Programmer's Guide

## MEDIAN [Analytic]

Returns the middle value of an expression in a result set within a window. A median value has the same number of records below it as above it. If there are an even number of elements, `MEDIAN()` returns the average of the two.

`MEDIAN()` is an alias for `50% PERCENTILE()`:

```
PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY expression)
```

## Behavior Type

Immutable

## Syntax

```
MEDIAN ( expression ) OVER ( [ window_partition_clause (page 130) ] )
```

## Parameters

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the middle value or an interpolated value that would be the middle value once the values are sorted. Null values are ignored in the calculation.
OVER (...)	See <b>Analytic Functions</b> . (page 128)

## Notes

- For each row, MEDIAN() returns the value that would fall in the middle of a value set within each partition.
- Vertica determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that data type, and returns that data type.
- MEDIAN() does not allow the `window_order_clause` or `window_frame_clause`.

## Examples

The following query computes the median annual income for first 500 customers in Wisconsin and in the District of Columbia. Note that median is reported for every row in the result set:

```
=> SELECT customer_state, annual_income,
        MEDIAN(annual_income) OVER (PARTITION BY customer_state) AS MEDIAN
   FROM customer_dimension
  WHERE customer_state IN ('DC','WI')
  ORDER BY customer_state;
```

customer_state	customer_key	annual_income	MEDIAN
DC	120	299768	535413
DC	113	535413	535413
DC	130	848360	535413
WI	372	34962	668147
WI	437	47128	668147
WI	435	67770	668147
WI	282	638054	668147
WI	314	668147	668147
WI	128	675608	668147
WI	179	825304	668147
WI	302	827618	668147
WI	29	922760	668147

(12 rows)

## See Also

**PERCENTILE\_CONT** (page 162)

Using SQL Analytics in the Programmer's Guide

## MIN [Analytic]

Returns the minimum value of an expression within a window. The return value is the same as the expression data type.

### Behavior Type

Immutable

### Syntax

```
MIN ( [ DISTINCT ] expression ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )
```

### Parameters

DISTINCT	Is meaningless in this context.
<i>expression</i>	Can be any expression for which the minimum value is calculated, typically a <b>column reference</b> (see " <b>Column References</b> " on page 46).
OVER(...)	See <b>Analytic Functions</b> . (page 128)

### Examples

The following query computes the deviation between the employees' annual salary and the minimum annual salary in Massachusetts:

```
=> SELECT employee_state, annual_salary,
        MIN(annual_salary)
        OVER(PARTITION BY employee_state ORDER BY employee_key) min,
        annual_salary- MIN(annual_salary)
        OVER(PARTITION BY employee_state ORDER BY employee_key) diff
FROM employee_dimension
WHERE employee_state = 'MA';
```

employee_state	annual_salary	min	diff
MA	1918	1204	714
MA	2058	1204	854
MA	2586	1204	1382
MA	2500	1204	1296
MA	1318	1204	114
MA	2072	1204	868
MA	2656	1204	1452
MA	2148	1204	944
MA	2366	1204	1162
MA	2664	1204	1460

(10 rows)

**See Also****MIN** (page 117) aggregate function**MAX** (page 155) analytic function

Using SQL Analytics in the Programmer's Guide

**NTILE [Analytic]**

Divides an ordered data set (partition) into buckets within a window, with the buckets numbered 1 through *constant-value*. For example, if *constant-value* = 4, then each row in the partition is assigned a number from 1 to 4. If the partition contains 20 rows, the first 5 would be assigned 1, the next 5 would be assigned 2, and so on.

**Behavior Type**

Immutable

**Syntax**

```
NTILE ( constant-value ) OVER (
... [ window_partition_clause (page 130) ]
... window_order_clause (page 131) )
```

**Parameters**

<i>constant-value</i>	Represents the number of buckets and must resolve to a positive constant for each partition.
OVER(...)	See <b>Analytic Functions</b> . (page 128)

**Notes**

- The analytic *window\_order\_clause* is required but the *window\_partition\_clause* is optional.
- If the number of buckets is greater than the number of rows, then a number of buckets equal to the number of rows is filled, and the remaining buckets are empty.
- In the event the cardinality of the partition is not evenly divisible by the number of buckets, the rows are distributed so no bucket has more than 1 row more than any other bucket, and the lowest buckets are the ones that have extra rows. For example, using *constant-value* = 4 again and the number of rows = 21, bucket = 1 has 6 rows, bucket = 2 has 5, and so on.
- Analytic functions, such as `NTILE()`, cannot be nested within aggregate functions.

**Examples**

The following query assigns each month's sales total into one of four buckets:

```
=> SELECT calendar_month_name AS MONTH, SUM(sales_quantity),
       NTILE(4) OVER (ORDER BY SUM(sales_quantity)) AS NTILE
FROM store.store_sales_fact JOIN date_dimension
USING(date_key)
```

```
GROUP BY calendar_month_name
ORDER BY NTILE;
```

MONTH	SUM	NTILE
February	755	1
June	842	1
September	849	1
January	881	2
May	882	2
July	894	2
August	921	3
April	952	3
March	987	3
October	1010	4
November	1026	4
December	1094	4

(12 rows)

**See Also****PERCENTILE\_CONT** (page 162)**WIDTH\_BUCKET** (page 298)

Using SQL Analytics in the Programmer's Guide

**PERCENT\_RANK [Analytic]**

Calculates the relative rank of a row for a given row in a group within a window by dividing that row's rank less 1 by the number of rows in the partition, also less 1. This function always returns values from 0 to 1 inclusive. The first row in any set has a `PERCENT_RANK()` of 0. The return value is `NUMBER`.

$$(\text{rank} - 1) / ([\text{rows}] - 1)$$

In the above formula, `rank` is the rank position of a row in the group and `rows` is the total number of rows in the partition defined by the `OVER()` clause.

**Behavior Type**

Immutable

**Syntax**

```
PERCENT_RANK ( ) OVER (
... [ window_partition_clause (page 130) ]
... window_order_clause (page 131) )
```

**Parameters**

OVER(...)	See <b>Analytic Functions</b> . (page 128)
-----------	--

## Notes

The `window_order_clause` is required but the `window_partition_clause` is optional.

## Examples

The following example finds the percent rank of gross profit for different states within each month of the first quarter:

```
=> SELECT calendar_month_name AS MONTH, store_state ,
        SUM(gross_profit_dollar_amount),
        PERCENT_RANK() OVER (PARTITION BY calendar_month_name
        ORDER BY SUM(gross_profit_dollar_amount)) AS PERCENT_RANK
FROM store.store_sales_fact JOIN date_dimension
USING(date_key)
JOIN store.store_dimension
USING (store_key)
WHERE calendar_month_name IN ('January','February','March')
AND store_state IN ('OR','IA','DC','NV','WI')
GROUP BY calendar_month_name, store_state
ORDER BY calendar_month_name, PERCENT_RANK;
```

MONTH	store_state	SUM	PERCENT_RANK
February	OR	16	0
February	IA	47	0.25
February	DC	94	0.5
February	NV	113	0.75
February	WI	119	1
January	IA	-263	0
January	OR	91	0.3333333333333333
January	NV	372	0.6666666666666667
January	DC	497	1
March	NV	-141	0
March	OR	224	1

(11 rows)

The following example calculates, for each employee, the percent rank of the employee's salary by their job title:

```
=> SELECT job_title, employee_last_name, annual_salary,
        PERCENT_RANK()
        OVER (PARTITION BY job_title ORDER BY annual_salary DESC) AS percent_rank
FROM employee_dimension
ORDER BY percent_rank, annual_salary;
```

job_title	employee_last_name	annual_salary	PERCENT_RANK
CEO	Campbell	963914	0
Co-Founder	Nguyen	968625	0
Founder	Overstreet	995533	0
Greeter	Peterson	3192	0.00113895216400911
Greeter	Greenwood	3192	0.00113895216400911

Customer Service	Peterson		3190		0.00121065375302663
Delivery Person	Rodriguez		3192		0.00121065375302663
Shelf Stocker	Martin		3194		0.00125786163522013
Shelf Stocker	Vu		3194		0.00125786163522013
Marketing	Li		99711		0.00190114068441065
Assistant Director	Sanchez		99913		0.00190839694656489
Branch Manager	Perkins		99901		0.00192307692307692
Advertising	Lampert		99809		0.00204918032786885
Sales	Miller		99727		0.00211416490486258
Shift Manager	King		99904		0.00215982721382289
Custodian	Bauer		3196		0.00235849056603774
Custodian	Goldberg		3196		0.00235849056603774
Customer Service	Fortin		3184		0.00242130750605327
Delivery Person	Greenwood		3186		0.00242130750605327
Cashier	Overstreet		3178		0.00243605359317905
Regional Manager	McCabe		199688		0.00306748466257669
VP of Sales	Li		199309		0.00313479623824451
Director of HR	Goldberg		199592		0.00316455696202532
Head of Marketing	Stein		199941		0.00317460317460317
VP of Advertising	Goldberg		199036		0.00323624595469256
Head of PR	Stein		199767		0.00323624595469256
Customer Service	Rodriguez		3180		0.0036319612590799
Delivery Person	King		3184		0.0036319612590799
Cashier	Dobisz		3174		0.00365408038976857
Cashier	Miller		3174		0.00365408038976857
Marketing	Dobisz		99655		0.00380228136882129
Branch Manager	Gauthier		99082		0.025
Branch Manager	Moore		98415		0.05
...					

## See Also

**CUME\_DIST** (page 140)

Using SQL Analytics in the Programmer's Guide

## PERCENTILE\_CONT [Analytic]

An inverse distribution function where, for each row, `PERCENTILE_CONT()` returns the value that would fall into the specified percentile among a set of values in each partition within a window. For example, if the argument to the function is 0.5, the result of the function is the median of the data set (the 50th percentile). `PERCENTILE_CONT()` assumes a continuous distribution data model. Nulls are ignored.

### Behavior Type

Immutable

### Syntax

```
PERCENTILE_CONT ( %_number ) WITHIN GROUP (
... ORDER BY expression [ ASC | DESC ] ) OVER (
... [ window_partition_clause (page 130) ] )
```

## Parameters

<code>%_number</code>	Is the percentile value, which must be a <code>FLOAT</code> constant ranging from 0 to 1 (inclusive).
<code>WITHIN GROUP (ORDER BY expression)</code>	Specifies how the data is sorted within each group. <code>ORDER BY</code> takes only one column/expression that must be <code>INTEGER</code> , <code>FLOAT</code> , <code>INTERVAL</code> , or <code>NUMERIC</code> data type. Nulls are discarded. <b>Note:</b> The <code>WITHIN GROUP (ORDER BY)</code> clause does not guarantee the order of the SQL result. Use the <b>SQL ORDER BY clause</b> (page 757) to guarantee the ordering of the final result set.
<code>ASC   DESC</code>	Specifies the ordering sequence as ascending (default) or descending.
<code>OVER (...)</code>	See <b>Analytic Functions</b> . (page 128)

## Notes

- Vertica computes the percentile by first computing the row number where the percentile row would exist; for example:  

$$\text{ROW\_NUMBER} = 1 + \text{PERCENTILE\_VALUE} * (\text{NUMBER\_OF\_ROWS\_IN\_PARTITION} - 1)$$
 If the  $\text{CEILING}(\text{ROW\_NUMBER}) = \text{FLOOR}(\text{ROW\_NUMBER})$ , then the percentile is the value at the `ROW_NUMBER`. Otherwise there was an even number of rows, and Vertica interpolates the value between the rows. In this case, the percentile  $\text{CEILING\_VAL} = \text{get the value at the CEILING}(\text{ROW\_NUMBER})$ .  $\text{FLOOR\_VAL} = \text{get the value at the FLOOR}(\text{ROW\_NUMBER})$  would be  $(\text{CEILING}(\text{ROW\_NUMBER}) - \text{ROW\_NUMBER}) * \text{CEILING\_VAL} + (\text{ROW\_NUMBER} - \text{FLOOR}(\text{ROW\_NUMBER})) * \text{FLOOR\_VAL}$ .  
 If  $\text{CEIL}(\text{num}) = \text{FLOOR}(\text{num}) = \text{num}$ , then retrieve the value in that row. Otherwise compute values at  $[\text{CEIL}(\text{num}) + \text{FLOOR}(\text{num})] / 2$
- Specifying `ASC` or `DESC` in the `WITHIN GROUP` clause affects results as long as the percentile parameter is not `.5`.
- The `MEDIAN()` function is a specific case of `PERCENTILE_CONT()` where the percentile value defaults to `0.5`. For more information, see `MEDIAN()` (page 156).

## Examples

This query computes the median annual income per group for the first 500 customers in Wisconsin and the District of Columbia.

```
=> SELECT customer_state, customer_key, annual_income,
        PERCENTILE_CONT(.5) WITHIN GROUP(ORDER BY annual_income)
        OVER (PARTITION BY customer_state) AS PERCENTILE_CONT
FROM customer_dimension
WHERE customer_state IN ('DC','WI')
AND customer_key < 300
ORDER BY customer_state, customer_key;
```

customer_state	customer_key	annual_income	PERCENTILE_CONT
DC	104	658383	658383

```

DC          |          168 |          417092 |          658383
DC          |          245 |          670205 |          658383
WI          |          106 |          227279 |          458607
WI          |          127 |          703889 |          458607
WI          |          209 |          458607 |          458607
(6 rows)

```

The median value for DC is 65838, and the median value for WI is 458607. Note that with a `%_number` of .5 in the above query, `PERCENTILE_CONT()` returns the same result as `MEDIAN()` in the following query:

```

=> SELECT customer_state, customer_key, annual_income,
       MEDIAN(annual_income)
       OVER (PARTITION BY customer_state) AS MEDIAN
FROM   customer_dimension
WHERE  customer_state IN ('DC','WI')
AND    customer_key < 300
ORDER BY customer_state, customer_key;

```

```

customer_state | customer_key | annual_income | MEDIAN
-----+-----+-----+-----
DC             |          104 |          658383 | 658383
DC             |          168 |          417092 | 658383
DC             |          245 |          670205 | 658383
WI             |          106 |          227279 | 458607
WI             |          127 |          703889 | 458607
WI             |          209 |          458607 | 458607
(6 rows)

```

### See Also

***MEDIAN*** (page 156)

Using SQL Analytics in the Programmer's Guide

## PERCENTILE\_DISC [Analytic]

An inverse distribution function where, for each row, `PERCENTILE_DISC()` returns the value that would fall into the specified percentile among a set of values in each partition within a window. `PERCENTILE_DISC()` assumes a discrete distribution data model. Nulls are ignored.

### Behavior Type

Immutable

### Syntax

```

PERCENTILE_DISC ( %_number ) WITHIN GROUP (
... ORDER BY expression [ ASC | DESC ] ) OVER (
... [ window_partition_clause (page 130) ] )

```

### Parameters

<code>%_number</code>	Is the percentile value, which must be a FLOAT constant ranging from 0 to 1 (inclusive).
-----------------------	--

WITHIN GROUP (ORDER BY <i>expression</i> )	Specifies how the data is sorted within each group. ORDER BY takes only one column/expression that must be INTEGER, FLOAT, INTERVAL, or NUMERIC data type. Nulls are discarded. <b>Note:</b> The WITHIN GROUP (ORDER BY) clause does not guarantee the order of the SQL result. Use the <b>SQL ORDER BY clause</b> (page 757) to guarantee the ordering of the final result set.
ASC   DESC	Specifies the ordering sequence as ascending (default) or descending.
OVER (...)	See <b>Analytic Functions</b> . (page 128)

## Notes

- PERCENTILE\_DISC(%\_number) examines the cumulative distribution values in each group until it finds one that is greater than or equal to %\_number.
- Vertica computes the percentile where, for each row, PERCENTILE\_DISC outputs the first value of the WITHIN GROUP (ORDER BY) column whose CUME\_DIST (cumulative distribution) value is >= the argument FLOAT value (for example, .4). Specifically:  

```
PERCENTILE_DIST(.4) WITHIN GROUP (ORDER BY salary) OVER(PARTITION BY deptno) ...
```

If you write, for example, `SELECT CUME_DIST() OVER(ORDER BY salary) FROM table;` you notice that the smallest CUME\_DIST value that is greater than .4 is also the PERCENTILE\_DISC.

## Examples

This query computes the 20th percentile annual income by group for first 500 customers in Wisconsin and the District of Columbia.

```
=> SELECT customer_state, customer_key, annual_income,
       PERCENTILE_DISC(.2) WITHIN GROUP(ORDER BY annual_income)
       OVER (PARTITION BY customer_state) AS PERCENTILE_DISC
FROM customer_dimension
WHERE customer_state IN ('DC','WI')
AND customer_key < 300
ORDER BY customer_state, customer_key;
```

customer_state	customer_key	annual_income	PERCENTILE_DISC
DC	104	658383	417092
DC	168	417092	417092
DC	245	670205	417092
WI	106	227279	227279
WI	127	703889	227279
WI	209	458607	227279

(6 rows)

## See Also

**CUME\_DIST** (page 140)

**PERCENTILE\_CONT** (page 162)

## Using SQL Analytics in the Programmer's Guide

**RANK [Analytic]**

Assigns a rank to each row returned from a query with respect to the other rows, based on the values of the expressions in the window `ORDER BY` clause. The data within a group is sorted by the `ORDER BY` clause and then a numeric ranking is assigned to each row in turn, starting with 1, and continuing up. Rows with the same values of the `ORDER BY` expressions receive the same rank; however, if two rows receive the same rank (a tie), `RANK()` skips the ties. If, for example, two rows are numbered 1, `RANK()` skips number 2 and assigns 3 to the next row in the group. This is in contrast to `DENSE_RANK()` (page 142), which does not skip values.

**Behavior Type**

Immutable

**Syntax**

```
RANK ( ) OVER (
... [ window_partition_clause (page 130) ]
... window_order_clause (page 131) )
```

**Parameters**

<code>OVER(...)</code>	See <b><i>Analytic Functions</i></b> . (page 128)
------------------------	---

**Notes**

- Ranking functions return a rank value for each row in a result set based on the order specified in the query. For example, a territory sales manager might want to identify the top or bottom ranking sales associates in a department or the highest/lowest-performing sales offices by region.
- `RANK()` requires an `OVER()` clause. The `window_partition_clause` is optional.
- In ranking functions, `OVER()` specifies the measures *expression* on which ranking is done and defines the order in which rows are sorted in each group (or partition). Once the data is sorted within each partition, ranks are given to each row starting from 1.
- The primary difference between `RANK` and `DENSE_RANK` is that `RANK` leaves gaps when ranking records; `DENSE_RANK` leaves no gaps. For example, if more than one record occupies a particular position (a tie), `RANK` places all those records in that position and it places the next record after a gap of the additional records (it skips one). `DENSE_RANK` places all the records in that position only—it does not leave a gap for the next rank.

If there is a tie at the third position with two records having the same value, `RANK` and `DENSE_RANK` place both the records in the third position only, but `RANK` has the next record at the fifth position — leaving a gap of 1 position—while `DENSE_RANK` places the next record at the fourth position (no gap).

- If you omit `NULLS FIRST | LAST | AUTO`, the ordering of the null values depends on the `ASC` or `DESC` arguments. Null values are considered larger than any other values. If the ordering sequence is `ASC`, then nulls appear last; nulls appear first otherwise. Nulls are considered equal to other nulls and, therefore, the order in which nulls are presented is non-deterministic.

## Examples

This example ranks the longest-standing customers in Massachusetts. The query first computes the `customer_since` column by region, and then partitions the results by customers with businesses in `MA`. Then within each region, the query ranks customers over the age of 70.

```
=> SELECT customer_type, customer_name,
        RANK() OVER (PARTITION BY customer_region ORDER BY customer_since) as rank
FROM customer_dimension
WHERE customer_state = 'MA'
AND customer_age > '70';
```

customer_type	customer_name	rank
Company	Virtadata	1
Company	Evergen	2
Company	Infocore	3
Company	Goldtech	4
Company	Veritech	5
Company	Inishop	6
Company	Intracom	7
Company	Virtacom	8
Company	Goldcom	9
Company	Infostar	10
Company	Golddata	11
Company	Everdata	12
Company	Goldcorp	13

(13 rows)

The following example shows the difference between `RANK` and `DENSE_RANK` when ranking customers by their annual income. Notice that `RANK` has a tie at 10 and skips 11, while `DENSE_RANK` leaves no gaps in the ranking sequence:

```
=> SELECT customer_name, SUM(annual_income),
        RANK () OVER (ORDER BY TO_CHAR(SUM(annual_income),'100000') DESC) rank,
        DENSE_RANK () OVER (ORDER BY TO_CHAR(SUM(annual_income),'100000') DESC)
dense_rank
FROM customer_dimension
GROUP BY customer_name
LIMIT 15;
```

customer_name	sum	rank	dense_rank
Brian M. Garnett	99838	1	1
Tanya A. Brown	99834	2	2
Tiffany P. Farmer	99826	3	3
Jose V. Sanchez	99673	4	4
Marcus D. Rodriguez	99631	5	5
Alexander T. Nguyen	99604	6	6
Sarah G. Lewis	99556	7	7
Ruth Q. Vu	99542	8	8

Theodore T. Farmer	99532	9	9
Daniel P. Li	99497	10	10
Seth E. Brown	99497	10	10
Matt X. Gauthier	99402	12	11
Rebecca W. Lewis	99296	13	12
Dean L. Wilson	99276	14	13
Tiffany A. Smith	99257	15	14

(15 rows)

**See Also****DENSE\_RANK** (page 142)

Using SQL Analytics in the Programmer's Guide

**ROW\_NUMBER [Analytic]**

Assigns a unique number, sequentially, starting from 1, to each row in a partition within a window.

**Behavior Type**

Immutable

**Syntax**

```
ROW_NUMBER ( ) OVER (
... [ window_partition_clause (page 130) ]
... window_order_clause (page 131) )
```

**Parameters**

OVER(...)	See <b>Analytic Functions.</b> (page 128)
-----------	---

**Notes**

- `ROW_NUMBER()` is a Vertica extension, not part of the SQL-99 standard. It requires an `OVER()` clause. The `window_partition_clause` is optional.
- You can use the optional partition clause to group data into partitions before operating on it; for example:  

```
SUM OVER (PARTITION BY col1, col2, ...)
```
- You can substitute any `RANK()` example for `ROW_NUMBER()`. The difference is that `ROW_NUMBER` assigns a unique ordinal number, starting with 1, to each row in the ordered set.

**Examples**

The following query first partitions customers in the `customer_dimension` table by occupation and then ranks those customers based on the ordered set specified by the analytic `partition_clause`.

```
=> SELECT occupation, customer_key, customer_since, annual_income,
       ROW_NUMBER() OVER (PARTITION BY occupation) AS customer_since_row_num
FROM public.customer_dimension
ORDER BY occupation, customer_since_row_num;
```

```
occupation      | customer_key | customer_since | annual_income | customer_since_row_num
```

Accountant	19453	1973-11-06	602460	1
Accountant	42989	1967-07-09	850814	2
Accountant	24587	1995-05-18	180295	3
Accountant	26421	2001-10-08	126490	4
Accountant	37783	1993-03-16	790282	5
Accountant	39170	1980-12-21	823917	6
Banker	13882	1998-04-10	15134	1
Banker	14054	1989-03-16	961850	2
Banker	15850	1996-01-19	262267	3
Banker	29611	2004-07-14	739016	4
Doctor	261	1969-05-11	933692	1
Doctor	1264	1981-07-19	593656	2
Psychologist	5189	1999-05-04	397431	1
Psychologist	5729	1965-03-26	339319	2
Software Developer	2513	1996-09-22	920003	1
Software Developer	5927	2001-03-12	633294	2
Software Developer	9125	1971-10-06	198953	3
Software Developer	16097	1968-09-02	748371	4
Software Developer	23137	1988-12-07	92578	5
Software Developer	24495	1989-04-16	149371	6
Software Developer	24548	1994-09-21	743788	7
Software Developer	33744	2005-12-07	735003	8
Software Developer	9684	1970-05-20	246000	9
Software Developer	24278	2001-11-14	122882	10
Software Developer	27122	1994-02-05	810044	11
Stock Broker	5950	1965-01-20	752120	1
Stock Broker	12517	2003-06-13	380102	2
Stock Broker	33010	1984-05-07	384463	3
Stock Broker	46196	1972-11-28	497049	4
Stock Broker	8710	2005-02-11	79387	5
Writer	3149	1998-11-17	643972	1
Writer	17124	1965-01-18	444747	2
Writer	20100	1994-08-13	106097	3
Writer	23317	2003-05-27	511750	4
Writer	42845	1967-10-23	433483	5
Writer	47560	1997-04-23	515647	6

(39 rows)

**See Also****RANK** (page 166)

Using SQL for Analytics in the Programmer's Guide

**STDDEV [Analytic]**

**Note:** The non-standard function `STDDEV()` is provided for compatibility with other databases. It is semantically identical to `STDDEV_SAMP()` (page 171).

Computes the statistical sample standard deviation of the current row with respect to the group within a window. The `STDDEV_SAMP()` return value is the same as the square root of the variance defined for the `VAR_SAMP()` function:

$$\text{STDDEV}(\text{expression}) = \text{SQRT}(\text{VAR\_SAMP}(\text{expression}))$$

When `VAR_SAMP()` returns null, this function returns null.

**Behavior Type**

Immutable

**Syntax**
`STDDEV ( expression ) OVER (`

... [ **window\_partition\_clause** (page 130) ]  
 ... [ **window\_order\_clause** (page 131) ]  
 ... [ **window\_frame\_clause** (page 133) ] )

## Parameters

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
OVER(...)	See <b>Analytic Functions</b> . (page 128)

## Example

The following example returns the standard deviations of salaries in the employee dimension table by job title Assistant Director:

```
=> SELECT employee_last_name, annual_salary,
        STDDEV(annual_salary) OVER (ORDER BY hire_date) as "stddev"
   FROM employee_dimension
  WHERE job_title = 'Assistant Director';
```

employee_last_name	annual_salary	stddev
Goldberg	61859	NaN
Miller	79582	12532.0534829692
Goldberg	74236	9090.97147357388
Campbell	66426	7909.9541665339
Moore	66630	7068.30282316761
Nguyen	53530	9154.14713486005
Harris	74115	8773.54346886142
Lang	59981	8609.60471031374
Farmer	60597	8335.41158418579
Nguyen	78941	8812.87941405456
Smith	55018	9179.7672390773
...		

## See Also

**STDDEV** (page 122) and **STDDEV\_SAMP** (page 123) aggregate functions

**STDDEV\_SAMP** (page 171) analytic function

Using SQL Analytics in the Programmer's Guide

## STDDEV\_POP [Analytic]

Computes the statistical population standard deviation and returns the square root of the population variance within a window. The **STDDEV\_POP()** return value is the same as the square root of the **VAR\_POP()** function:

```
STDDEV_POP(expression) = SQRT(VAR_POP(expression))
```

When **VAR\_POP** returns null, this function returns null.

## Behavior Type

Immutable

## Syntax

```
STDDEV_POP ( expression ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )
```

## Parameters

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
OVER(...)	See <b>Analytic Functions</b> . (page 128)

## Examples

The following example returns the population standard deviations of salaries in the employee dimension table by job title Assistant Director:

```
=> SELECT employee_last_name, annual_salary,
        STDDEV_POP(annual_salary) OVER (ORDER BY hire_date) as "stddev_pop"
FROM employee_dimension WHERE job_title = 'Assistant Director';
employee_last_name | annual_salary | stddev_pop
```

```
-----+-----+-----
Goldberg           |          61859 |          0
Miller             |          79582 |         8861.5
Goldberg           |          74236 | 7422.74712548456
Campbell           |          66426 | 6850.22125098891
Moore              |          66630 | 6322.08223926257
Nguyen             |          53530 | 8356.55480080699
Harris             |          74115 | 8122.72288970008
Lang               |          59981 | 8053.54776538731
Farmer             |          60597 | 7858.70140687825
Nguyen             |          78941 | 8360.63150784682
```

## See Also

**STDDEV\_POP** (page 123) aggregate functions

Using SQL Analytics in the Programmer's Guide

## STDDEV\_SAMP [Analytic]

Computes the statistical sample standard deviation of the current row with respect to the group within a window. The `STDDEV_SAMP()` return value is the same as the square root of the variance defined for the `VAR_SAMP()` function:

```
STDDEV(expression) = SQRT(VAR_SAMP(expression))
```

When `VAR_SAMP()` returns null, this function returns null.

## Behavior Type

Immutable

## Syntax

```
STDDEV_SAMP ( expression ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )
```

## Parameters

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument..
OVER(...)	See <b>Analytic Functions</b> . (page 128)

## Notes

STDDEV\_SAMP() is semantically identical to the non-standard function, STDDEV() (page 122).

## Examples

The following example returns the sample standard deviations of salaries in the employee dimension table by job title Assistant Director:

```
=> SELECT employee_last_name, annual_salary,
        STDDEV(annual_salary) OVER (ORDER BY hire_date) as "stddev_samp"
   FROM employee_dimension WHERE job_title = 'Assistant Director';
```

employee_last_name	annual_salary	stddev_samp
Goldberg	61859	NaN
Miller	79582	12532.0534829692
Goldberg	74236	9090.97147357388
Campbell	66426	7909.9541665339
Moore	66630	7068.30282316761
Nguyen	53530	9154.14713486005
Harris	74115	8773.54346886142
Lang	59981	8609.60471031374
Farmer	60597	8335.41158418579
Nguyen	78941	8812.87941405456
...		

## See Also

**Analytic Functions** (page 128)

**STDDEV** (page 169) analytic function

**STDDEV** (page 122) and **STDDEV\_SAMP** (page 123) aggregate functions

Using SQL Analytics in the Programmer's Guide

## SUM [Analytic]

Computes the sum of an expression over a group of rows within a window. It returns a `DOUBLE PRECISION` value for a floating-point expression. Otherwise, the return value is the same as the expression data type.

### Behavior Type

Immutable

### Syntax

```
SUM ( expression ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )
```

### Parameters

<i>expression</i>	Any <code>NUMERIC</code> <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
OVER(...)	See <b>Analytic Functions</b> . (page 128)

### Notes

- If you encounter data overflow when using `SUM()`, use `SUM_FLOAT()` (page 125) which converts data to a floating point.
- `SUM()` returns the sum of values of an expression.

### Examples

The following query returns the cumulative sum all of the returns made to stores in January:

```
=> SELECT calendar_month_name AS month, transaction_type, sales_quantity,
        SUM(sales_quantity)
        OVER (PARTITION BY calendar_month_name ORDER BY date_dimension.date_key) AS
SUM
FROM store.store_sales_fact JOIN date_dimension
USING(date_key) WHERE calendar_month_name IN ('January')
AND transaction_type= 'return';
```

month	transaction_type	sales_quantity	SUM
January	return	4	2338
January	return	3	2338
January	return	1	2338
January	return	5	2338
January	return	8	2338
January	return	3	2338
January	return	5	2338

```

January | return          |          10 | 2338
January | return          |           9 | 2338
January | return          |          10 | 2338
(10 rows)

```

**See Also**

**SUM** (page 124) aggregate function

**Numeric Data Types** (page 95)

Using SQL Analytics in the Programmer's Guide

**VAR\_POP [Analytic]**

Returns the statistical population variance of a non-null set of numbers (nulls are ignored) in a group within a window. Results are calculated by the sum of squares of the difference of *expression* from the mean of *expression*, divided by the number of rows remaining:

$$\frac{(\text{SUM}(\text{expression} * \text{expression}) - \text{SUM}(\text{expression}) * \text{SUM}(\text{expression})) / \text{COUNT}(\text{expression})}{\text{COUNT}(\text{expression})}$$

**Behavior Type**

Immutable

**Syntax**

```

VAR_POP ( expression ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )

```

**Parameters**

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument
OVER(...)	See <b>Analytic Functions</b> . (page 128)

**Examples**

The following example calculates the cumulative population in the store orders fact table of sales in December 2007:

```

=> SELECT date_ordered,
        VAR_POP(SUM(total_order_cost))
        OVER (ORDER BY date_ordered) "var_pop"
FROM store.store_orders_fact s
WHERE date_ordered BETWEEN '2007-12-01' AND '2007-12-31'
GROUP BY s.date_ordered;

```

```

date_ordered |      var_pop

```

```

-----+-----
2007-12-01 |          0
2007-12-02 |      1129564881
2007-12-03 | 1206008121.55542
2007-12-04 | 26353624176.1875
2007-12-05 | 21315288023.4402
2007-12-06 | 21619271028.3333
2007-12-07 | 19867030477.6328
2007-12-08 |      19197735288.5
2007-12-09 | 19100157155.2097
2007-12-10 | 19369222968.0896
(10 rows)

```

**See Also**

**VAR\_POP** (page 126) aggregate function

Using SQL Analytics in the Programmer's Guide

**VAR\_SAMP [Analytic]**

Returns the sample variance of a non-null set of numbers (nulls in the set are ignored) for each row of the group within a window. Results are calculated by the sum of squares of the difference of *expression* from the mean of *expression*, divided by the number of rows remaining minus 1:

$$\frac{(\text{SUM}(\text{expression} * \text{expression}) - \text{SUM}(\text{expression}) * \text{SUM}(\text{expression}) / \text{COUNT}(\text{expression}))}{(\text{COUNT}(\text{expression}) - 1)}$$
**Behavior Type**

Immutable

**Syntax**

```

VAR_SAMP ( expression ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )

```

**Parameters**

<i>expression</i>	Any NUMERIC <b>data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument
OVER (...)	See <b>Analytic Functions.</b> (page 128)

**Notes**

- VAR\_SAMP () returns the sample variance of a set of numbers after it discards the nulls in the set.
- If the function is applied to an empty set, then it returns null.
- This function is similar to VARIANCE (), except that given an input set of one element, VARIANCE () returns 0 and VAR\_SAMP () returns null.

## Examples

The following example calculates the sample variance in the store orders fact table of sales in December 2007:

```
=> SELECT date_ordered,
        VAR_SAMP(SUM(total_order_cost))
        OVER (ORDER BY date_ordered) "var_samp"
FROM store.store_orders_fact s
WHERE date_ordered BETWEEN '2007-12-01' AND '2007-12-31'
GROUP BY s.date_ordered;
```

date_ordered	var_samp
2007-12-01	NaN
2007-12-02	2259129762
2007-12-03	1809012182.33301
2007-12-04	35138165568.25
2007-12-05	26644110029.3003
2007-12-06	25943125234
2007-12-07	23178202223.9048
2007-12-08	21940268901.1431
2007-12-09	21487676799.6108
2007-12-10	21521358853.4331

(10 rows)

## See Also

**VARIANCE** (page 176) analytic function

**VAR\_SAMP** (page 127) aggregate function

Using SQL Analytics in the Programmer's Guide

## VARIANCE [Analytic]

**Note:** The non-standard function `VARIANCE()` is provided for compatibility with other databases. It is semantically identical to `VAR_SAMP()` (page 175).

Returns the sample variance of a non-null set of numbers (nulls in the set are ignored) for each row of the group within a window. Results are calculated by the sum of squares of the difference of *expression* from the mean of *expression*, divided by the number of rows remaining minus 1:

$$\frac{(\text{SUM}(\text{expression} * \text{expression}) - \text{SUM}(\text{expression}) * \text{SUM}(\text{expression})) / \text{COUNT}(\text{expression})}{(\text{COUNT}(\text{expression}) - 1)}$$

## Behavior Type

Immutable

## Syntax

```
VAR_SAMP ( expression ) OVER (
... [ window_partition_clause (page 130) ]
... [ window_order_clause (page 131) ]
... [ window_frame_clause (page 133) ] )
```

## Parameters

<i>expression</i>	Any <b>NUMERIC data type</b> (page 95) or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
OVER(...)	See <b>Analytic Functions</b> . (page 128)

## Notes

- `VARIANCE()` returns the variance of *expression*.
- The variance of *expression* is calculated as follows:
  - 0 if the number of rows in *expression* = 1
  - `VAR_SAMP()` if the number of rows in *expression* > 1

## Examples

The following example calculates the cumulative variance in the store orders fact table of sales in December 2007:

```
=> SELECT date_ordered,
        VARIANCE(SUM(total_order_cost))
        OVER (ORDER BY date_ordered) "variance"
   FROM store.store_orders_fact s
  WHERE date_ordered BETWEEN '2007-12-01' AND '2007-12-31'
  GROUP BY s.date_ordered;
```

date_ordered	variance
2007-12-01	NaN
2007-12-02	2259129762
2007-12-03	1809012182.33301
2007-12-04	35138165568.25
2007-12-05	26644110029.3003
2007-12-06	25943125234
2007-12-07	23178202223.9048
2007-12-08	21940268901.1431
2007-12-09	21487676799.6108
2007-12-10	21521358853.4331

(10 rows)

## See Also

**VAR\_SAMP** (page 175) analytic function

**VARIANCE** (page 127) and **VAR\_SAMP** (page 127) aggregate functions

Using SQL Analytics in the Programmer's Guide

## Performance Optimization for Analytic Sort Computation

Vertica stores data in projections that is sorted in a specific way. All columns are stored in `ASC` (ascending) order, but the placement of nulls depends on the column's data type.

The analytic `ORDER BY (window_order_clause)` and the SQL `ORDER BY` clause also perform slightly different sort operations:

- The analytic `window_order_clause` sorts data that is used by the analytic function as either ascending (`ASC`) or descending (`DESC`) and specifies where null values appear in the sorted result as either `NULLS FIRST` or `NULLS LAST`. The following is the default sort order:
  - `ASC + NULLS LAST`. Null values are placed at the end of the sorted result
  - `DESC + NULLS FIRST`. Null values are placed at the beginning of the sorted result
- The SQL `ORDER BY` clause specifies only ascending or descending order; however, the following is the default for null placement in Vertica:
  - `NUMERIC, INTEGER, DATE, TIME, TIMESTAMP, and INTERVAL` columns. `NULLS FIRST` (null values are stored at the beginning of a sorted projection).
  - `FLOAT, STRING, and BOOLEAN` columns. `NULLS LAST` (null values are stored at the end of a sorted projection).
  - No matter what the data type, if you specify `NULLS AUTO`, Vertica chooses the most efficient placement of nulls (for example, either `NULLS FIRST` or `NULLS LAST`) based on your query.

If you do not care about null placement in queries that involve analytics computation, or if you know that columns contain no null values, specify `NULLS AUTO`, and Vertica chooses the placement that gives the fastest performance. Otherwise you can specify `NULLS FIRST` or `NULLS LAST`.

You can also carefully formulate queries so Vertica can avoid sorting the data and can process the query more quickly, as illustrated by the following example.

### Example

In the following example, Vertica sorts inputs from table `t` on column `x`, as specified in the `OVER (ORDER BY)` clause. Then it evaluates `RANK()`:

```
=> CREATE TABLE t (  
    x FLOAT,  
    y FLOAT );  
=> CREATE PROJECTION t_p (x, y) AS SELECT * FROM t  
    ORDER BY x, y UNSEGMENTED ALL NODES;  
=> SELECT x, RANK() OVER (ORDER BY x) FROM t;
```

In the above `SELECT` statement, Vertica can eliminate the `ORDER BY` clause and run the query quickly because column `x` is a `FLOAT` data type; thus, the projection sort order matches the analytic default ordering (`ASC + NULLS LAST`). Vertica can also avoid having to sort the data when the underlying projection is already sorted.

Assume, however, that column `x` had been defined as `INTEGER`. Vertica cannot avoid sorting the data because the projection sort order for `INTEGER` data types (`ASC + NULLS FIRST`) does not match default analytic ordering (`ASC + NULLS LAST`). To help Vertica eliminate the sort, specify the placement of nulls to match default ordering:

```
=> SELECT x, RANK() OVER (ORDER BY x NULLS FIRST) FROM t;
```

If column `x` is defined as a `STRING`, the following query would eliminate the sort:

```
=> SELECT x, RANK() OVER (ORDER BY x NULLS LAST) FROM t;
```

Note that omitting `NULLS LAST` in the above query still eliminates the sort because `ASC + NULLS LAST` is the default sort specification for both the analytic `ORDER BY` clause and for string-related columns in Vertica.

### Data Types and their Default Sorting

The following tables summarizes the data types supported by Vertica, as well as the default placement of null values in projections. The Size column is shown in uncompressed bytes.

Type	Size	Description	NULL Sorting
<b>Binary types</b>			
BINARY	1 to 65000	Fixed-length binary string	NULLS LAST
VARBINARY	1 to 65000	Variable-length binary string	NULLS LAST
BYTEA	1 to 65000	Variable-length binary string (synonym for VARBINARY)	NULLS LAST
RAW	1 to 65000	Variable-length binary string (synonym for VARBINARY)	NULLS LAST
<b>Boolean types</b>			
BOOLEAN	1	True or False or NULL	NULLS LAST
<b>Character types</b>			
CHAR	1 to 65000	Fixed-length character string	NULLS LAST
VARCHAR	1 to 65000	Variable-length character string	NULLS LAST
<b>Date/time types</b>			
DATE	8	Represents a month, day, and year	NULLS FIRST
DATETIME	8	Represents a date and time with or without timezone (synonym for TIMESTAMP)	NULLS FIRST
SMALLDATETIME	8	Represents a date and time with or without timezone (synonym for TIMESTAMP)	NULLS FIRST
TIME	8	Represents a time of day without timezone	NULLS FIRST

TIME WITH TIMEZONE	8	Represents a time of day with timezone	NULLS FIRST
TIMESTAMP	8	Represents a date and time without timezone	NULLS FIRST
TIMESTAMP WITH TIMEZONE	8	Represents a date and time with timezone	NULLS FIRST
INTERVAL	8	Measures the difference between two points in time	NULLS FIRST
<b>Approximate numeric types</b>			
DOUBLE PRECISION	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
FLOAT	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
FLOAT (n)	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
FLOAT8	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
REAL	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
<b>Exact numeric types</b>			
INTEGER	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
INT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
BIGINT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
INT8	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
SMALLINT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
TINYINT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
DECIMAL	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST
NUMERIC	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST
NUMBER	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST

MONEY	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST
-------	----	--	-------------

**See Also**

Using SQL Analytics in the Programmer's Guide

**Boolean Functions****BIT\_AND**

Takes the bitwise `AND` of all non-null input values. If the input parameter is `NULL`, the return value is also `NULL`.

**Behavior Type**

Immutable

**Syntax**

```
BIT_AND ( expression )
```

**Parameters**

<i>expression</i>	The [BINARY  VARBINARY] input value to be evaluated. <code>BIT_AND()</code> operates on <code>VARBINARY</code> types explicitly and on <code>BINARY</code> types implicitly through <b>casts</b> (page 107).
-------------------	--

**Notes**

- The function returns the same value as the argument data type.
- For each bit compared, if **all** bits are 1, the function returns 1; otherwise it returns 0.
- If the columns are different lengths, the return values are treated as though they are all equal in length and are right-extended with zero bytes. For example, given a group containing the hex values 'ff', null, and 'f', the function ignores the null value and extends the value 'f' to 'f0'..

**Example**

This examples uses the following schema, which creates table `t` with a single column of `VARBINARY` data type:

```
=> CREATE TABLE t (
      c VARBINARY(2) );
=> INSERT INTO t values(HEX_TO_BINARY('0xFF00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFFFF'));
=> INSERT INTO t values(HEX_TO_BINARY('0xF00F'));
```

Query table `t` to see column `c` output:

```
=> SELECT TO_HEX(c) FROM t;
   TO_HEX
-----
```

```
ff00
ffff
f00f
(3 rows)
```

Query table `t` to get the AND value for column `c`:

```
SELECT TO_HEX(BIT_AND(c)) FROM t;
TO_HEX
-----
f000
(1 row)
```

The function is applied pairwise to all values in the group, resulting in `f000`, which is determined as follows:

- 1 `ff00` (record 1) is compared with `ffff` (record 2), which results in `ff00`.
- 2 The result from the previous comparison is compared with `f00f` (record 3), which results in `f000`.

### See Also

***Binary Data Types*** (page 65)

## BIT\_OR

Takes the bitwise OR of all non-null input values. If the input parameter is `NULL`, the return value is also `NULL`.

### Behavior Type

Immutable

### Syntax

```
BIT_OR ( expression )
```

### Parameters

<i>expression</i>	The [BINARY  VARBINARY] input value to be evaluated. <code>BIT_OR()</code> operates on <code>VARBINARY</code> types explicitly and on <code>BINARY</code> types implicitly through <b>casts</b> (page 107).
-------------------	---

### Notes

- The function returns the same value as the argument data type.
- For each bit compared, if **any** bit is 1, the function returns 1; otherwise it returns 0.
- If the columns are different lengths, the return values are treated as though they are all equal in length and are right-extended with zero bytes. For example, given a group containing the hex values `'ff'`, `null`, and `'f'`, the function ignores the null value and extends the value `'f'` to `'f0'`.

## Example

This examples uses the following schema, which creates table `t` with a single column of `VARBINARY` data type:

```
=> CREATE TABLE t (
      c VARBINARY(2) );
=> INSERT INTO t values(HEX_TO_BINARY('0xFF00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFFFF'));
=> INSERT INTO t values(HEX_TO_BINARY('0xF00F'));
```

Query table `t` to see column `c` output:

```
=> SELECT TO_HEX(c) FROM t;
  TO_HEX
-----
  ff00
  ffff
  f00f
(3 rows)
```

Query table `t` to get the OR value for column `c`:

```
SELECT TO_HEX(BIT_OR(c)) FROM t;
  TO_HEX
-----
  ffff
(1 row)
```

The function is applied pairwise to all values in the group, resulting in `ffff`, which is determined as follows:

- 1 `ff00` (record 1) is compared with `ffff`, which results in `ffff`.
- 2 The `ffff` result from the previous comparison is compared with `f00f` (record 3), which results in `ffff`.

## See Also

***Binary Data Types*** (page 65)

## BIT\_XOR

Takes the bitwise XOR of all non-null input values. If the input parameter is `NULL`, the return value is also `NULL`.

## Behavior Type

Immutable

## Syntax

```
BIT_XOR ( expression )
```

## Parameters

<i>expression</i>	The [BINARY   VARBINARY] input value to be evaluated. <code>BIT_XOR()</code> operates on <code>VARBINARY</code> types explicitly and on
-------------------	---

BINARY types implicitly through <b>casts</b> (page 107).
--

## Notes

- The function returns the same value as the argument data type.
- For each bit compared, if there are an odd number of arguments with set bits, the function returns 1; otherwise it returns 0.
- If the columns are different lengths, the return values are treated as though they are all equal in length and are right-extended with zero bytes. For example, given a group containing the hex values 'ff', null, and 'f', the function ignores the null value and extends the value 'f' to 'f0'.

## Example

First create a sample table and projections with binary columns:

This examples uses the following schema, which creates table `t` with a single column of `VARBINARY` data type:

```
=> CREATE TABLE t (  
    c VARBINARY(2) );  
=> INSERT INTO t values(HEX_TO_BINARY('0xFF00'));  
=> INSERT INTO t values(HEX_TO_BINARY('0xFFFF'));  
=> INSERT INTO t values(HEX_TO_BINARY('0xF00F'));
```

Query table `t` to see column `c` output:

```
=> SELECT TO_HEX(c) FROM t;  
TO_HEX  
-----  
ff00  
ffff  
f00f  
(3 rows)
```

Query table `t` to get the XOR value for column `c`:

```
SELECT TO_HEX(BIT_XOR(c)) FROM t;  
TO_HEX  
-----  
f0f0  
(1 row)
```

## See Also

**Binary Data Types** (page 65)

## Date/Time Functions

Date and time functions perform conversion, extraction, or manipulation operations on date and time data types and can return date and time information.

## Usage

Functions that take `TIME` or `TIMESTAMP` inputs come in two variants:

- `TIME WITH TIME ZONE` or `TIMESTAMP WITH TIME ZONE`
- `TIME WITHOUT TIME ZONE` or `TIMESTAMP WITHOUT TIME ZONE`

For brevity, these variants are not shown separately.

The `+` and `*` operators come in commutative pairs; for example, both `DATE + INTEGER` and `INTEGER + DATE`. We show only one of each such pair.

## Daylight Savings Time Considerations

When adding an `INTERVAL` value to (or subtracting an `INTERVAL` value from) a `TIMESTAMP WITH TIME ZONE` value, the days component advances (or decrements) the date of the `TIMESTAMP WITH TIME ZONE` by the indicated number of days. Across daylight saving time changes (with the session time zone set to a time zone that recognizes DST), this means `INTERVAL '1 day'` does not necessarily equal `INTERVAL '24 hours'`.

For example, with the session time zone set to `CST7CDT`:

```
TIMESTAMP WITH TIME ZONE '2005-04-02 12:00-07' + INTERVAL '1 day'
```

produces

```
TIMESTAMP WITH TIME ZONE '2005-04-03 12:00-06'
```

Adding `INTERVAL '24 hours'` to the same initial `TIMESTAMP WITH TIME ZONE` produces

```
TIMESTAMP WITH TIME ZONE '2005-04-03 13:00-06',
```

as there is a change in daylight saving time at `2005-04-03 02:00` in time zone `CST7CDT`.

## Date/Time Functions in Transactions

`CURRENT_TIMESTAMP()` and related functions return the start time of the current transaction; their values do not change during the transaction. The intent is to allow a single transaction to have a consistent notion of the "current" time, so that multiple modifications within the same transaction bear the same timestamp. However, `TIMEOFDAY()` returns the wall-clock time and advances during transactions.

## See Also

*Template Patterns for Date/Time Formatting* (page 248)

## ADD\_MONTHS

Takes a `DATE`, `TIMESTAMP`, or `TIMESTAMPTZ` argument and a number of months and returns a date. `TIMESTAMPTZ` arguments are implicitly cast to `TIMESTAMP`.

## Behavior Type

Immutable if called with `DATE` or `TIMESTAMP` but stable with `TIMESTAMPTZ` in that its results can change based on `TIMEZONE` settings

## Syntax

```
ADD_MONTHS ( d , n );
```

## Parameters

<i>d</i>	Is the incoming DATE, TIMESTAMP, or TIMESTAMPZ. If the start date falls on the last day of the month, or if the resulting month has fewer days than the given day of the month, then the result is the last day of the resulting month. Otherwise, the result has the same start day.
<i>n</i>	Can be any INTEGER.

## Examples

The following example's results include a leap year:

```
SELECT ADD_MONTHS('31-Jan-08', 1) "Months";
       Months
-----
2008-02-29
(1 row)
```

The next example adds four months to January and returns a date in May:

```
SELECT ADD_MONTHS('31-Jan-08', 4) "Months";
       Months
-----
2008-05-31
(1 row)
```

This example subtracts 4 months from January, returning a date in September:

```
SELECT ADD_MONTHS('31-Jan-08', -4) "Months";
       Months
-----
2007-09-30
(1 row)
```

Because the following example specifies NULL, the result set is empty:

```
SELECT ADD_MONTHS('31-Jan-03', NULL) "Months";
       Months
-----

(1 row)
```

This example provides no date argument, so even though the number of months specified is 1, the result set is empty:

```
SELECT ADD_MONTHS(NULL, 1) "Months";
       Months
-----

(1 row)
```

In this example, the date field defaults to a timestamp, so the PST is ignored. Notice that even though it is already the next day in Pacific time, the result falls on the same date in New York (two years later):

```
SET TIME_ZONE 'America/New_York';
SELECT ADD_MONTHS('2008-02-29 23:30 PST', 24);
  add_months
-----
2010-02-28
(1 row)
```

This example specifies a timestamp with time zone, so the PST is taken into account:

```
SET TIME_ZONE 'America/New_York';
SELECT ADD_MONTHS('2008-02-29 23:30 PST'::TIMESTAMPTZ, 24);
  add_months
-----
2010-03-01
(1 row)
```

## AGE\_IN\_MONTHS

Returns an INTEGER value representing the difference in months between two **TIMESTAMP**, **DATE** or **TIMESTAMPTZ** values.

### Behavior Type

Stable if second argument is omitted or if either argument is **TIMESTAMPTZ**. Immutable otherwise.

### Syntax

```
AGE_IN_MONTHS ( expression1 [ , expression2 ] )
```

### Parameters

<i>expression1</i>	specifies the beginning of the period.
<i>expression2</i>	specifies the end of the period. The default is the <b>CURRENT_DATE</b> (page 190).

### Notes

The inputs can be **TIMESTAMP**, **TIMESTAMPTZ**, or **DATE**.

### Examples

The following example returns the age in months of a person born on March 2, 1972 on the date June 21, 1990, with a time elapse of 18 years, 3 months, and 19 days:

```
SELECT AGE_IN_MONTHS(TIMESTAMP '1990-06-21', TIMESTAMP '1972-03-02');
  AGE_IN_MONTHS
-----
                219
(1 row)
```

The next example shows the age in months of the same person (born March 2, 1972) as of March 16, 2010:

```
SELECT AGE_IN_MONTHS (TIMESTAMP 'March 16, 2010', TIMESTAMP '1972-03-02');
      AGE_IN_MONTHS
-----
                456
(1 row)
```

This example returns the age in months of a person born on November 21, 1939:

```
SELECT AGE_IN_MONTHS (TIMESTAMP '1939-11-21');
      AGE_IN_MONTHS
-----
                844
(1 row)
```

In the above form, the result changes as time goes by.

**See Also**

**AGE\_IN\_YEARS** (page 188)

**INTERVAL** (page 74)

**AGE\_IN\_YEARS**

Returns an INTEGER value representing the difference in years between two **TIMESTAMP**, **DATE** or **TIMESTAMPTZ** values.

**Behavior Type**

Stable if second argument is omitted or if either argument is **TIMESTAMPTZ**. Immutable otherwise.

**Syntax**

```
AGE_IN_YEARS ( expression1 [ , expression2 ] )
```

**Parameters**

<i>expression1</i>	specifies the beginning of the period.
<i>expression2</i>	specifies the end of the period. The default is the <b>CURRENT_DATE</b> (page 190).

**Notes**

- The **AGE\_IN\_YEARS()** function was previously called **AGE**. **AGE()** is not supported.
- Inputs can be **TIMESTAMP**, **TIMESTAMPTZ**, or **DATE**.

**Examples**

The following example returns the age in years of a person born on March 2, 1972 on the date June 21, 1990, with a time elapse of 18 years, 3 months, and 19 days:

```
SELECT AGE_IN_YEARS (TIMESTAMP '1990-06-21', TIMESTAMP '1972-03-02');
      AGE_IN_YEARS
-----
```

```

          18
(1 row)

```

The next example shows the age in years of the same person (born March 2, 1972) as of February 24, 2009:

```

SELECT AGE_IN_YEARS (TIMESTAMP '2009-02-24', TIMESTAMP '1972-03-02');
   AGE_IN_YEARS
-----
          36
(1 row)

```

This example returns the age in years of a person born on November 21, 1939:

```

SELECT AGE_IN_YEARS (TIMESTAMP '1939-11-21');
   AGE_IN_YEARS
-----
          70
(1 row)

```

### See Also

***AGE\_IN\_MONTHS*** (page 187)

***INTERVAL*** (page 74)

## CLOCK\_TIMESTAMP

Returns a value of type `TIMESTAMP WITH TIMEZONE` representing the current system-clock time.

### Behavior Type

Volatile

### Syntax

```
CLOCK_TIMESTAMP()
```

### Notes

This function uses the date and time supplied by the operating system on the server to which you are connected, which should be the same across all servers. The value changes each time you call it.

### Examples

The following command returns the current time on your system:

```

SELECT CLOCK_TIMESTAMP() "Current Time";
   Current Time
-----
2010-09-23 11:41:23.33772-04
(1 row)

```

Each time you call the function, you get a different result. The difference in this example is in microseconds:

```

SELECT CLOCK_TIMESTAMP() "Time 1", CLOCK_TIMESTAMP() "Time 2";
   Time 1 | Time 2
-----

```

```
-----+-----  
2010-09-23 11:41:55.369201-04 | 2010-09-23 11:41:55.369202-04  
(1 row)
```

**See Also****STATEMENT\_TIMESTAMP** (page 223)**TRANSACTION\_TIMESTAMP** (page 235)**CURRENT\_DATE**

Returns the date (date-type value) on which the current transaction started.

**Behavior Type**

Stable

**Syntax**

CURRENT\_DATE

**Notes**

The CURRENT\_DATE function does not require parentheses.

**Examples**

```
SELECT CURRENT_DATE;  
?column?  
-----  
2010-09-23  
(1 row)
```

**CURRENT\_TIME**

Returns a value of type TIME WITH TIMEZONE representing the time of day.

**Behavior Type**

Stable

**Syntax**CURRENT\_TIME [ ( *precision* ) ]**Parameters**

<i>precision</i>	(INTEGER) causes the result to be rounded to the specified number of fractional digits in the seconds field.
------------------	--

**Notes**

- This function returns the start time of the current transaction; the value does not change during the transaction. The intent is to allow a single transaction to have a consistent notion of the current time, so that multiple modifications within the same transaction bear the same timestamp.

- The CURRENT\_TIME function does not require parentheses.

### Examples

```
SELECT CURRENT_TIME "Current Time";
      Current Time
-----
12:45:12.186089-05
(1 row)
```

## CURRENT\_TIMESTAMP

Returns a value of type TIMESTAMP WITH TIME ZONE representing the start of the current transaction.

### Behavior Type

Stable

### Syntax

```
CURRENT_TIMESTAMP [ ( precision ) ]
```

### Parameters

<i>precision</i>	(INTEGER) causes the result to be rounded to the specified number of fractional digits in the seconds field. Range of INTEGER is 0-6.
------------------	---

### Notes

This function returns the start time of the current transaction; the value does not change during the transaction. The intent is to allow a single transaction to have a consistent notion of the "current" time, so that multiple modifications within the same transaction bear the same timestamp.

### Examples

```
SELECT CURRENT_TIMESTAMP;
      ?column?
-----
2010-09-23 11:37:22.354823-04
(1 row)
SELECT CURRENT_TIMESTAMP(2);
      ?column?
-----
2010-09-23 11:37:22.35-04
(1 row)
```

## DATE\_PART

Is modeled on the traditional Ingres equivalent to the SQL-standard function EXTRACT. Internally DATE\_PART is used by the EXTRACT function.

### Behavior Type

Stable when source is of type TIMESTAMPTZ, Immutable otherwise.

## Syntax

DATE\_PART ( *field* , *source* )

## Parameters

<i>field</i>	Is a single-quoted string value that specifies the field to extract. <b>Note:</b> The <i>field</i> parameter values are the same for the <b>EXTRACT</b> (page 207) function.
<i>source</i>	Is a <b>date/time</b> (page 72) expression

## Field Values

CENTURY	<p>The century number.</p> <pre>SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');</pre> <p>Result: 20</p> <pre>SELECT EXTRACT(CENTURY FROM TIMESTAMP '2001-02-16 20:38:40');</pre> <p>Result: 21</p> <p>The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries. There is no century number 0, you go from -1 to 1.</p>
DAY	<p>The day (of the month) field (1 - 31).</p> <pre>SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40');</pre> <p>Result: 16</p> <pre>SELECT EXTRACT(DAY FROM DATE '2001-02-16');</pre> <p>Result: 16</p>
DECADE	<p>The year field divided by 10.</p> <pre>SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40');</pre> <p>Result: 200</p> <pre>SELECT EXTRACT(DECADE FROM DATE '2001-02-16');</pre> <p>Result: 200</p>
DOQ	<p>The day within the current quarter.</p> <pre>SELECT EXTRACT(DOQ FROM CURRENT_DATE);</pre> <p>Result: 89</p> <p>The result is calculated as follows: Current date = June 28, current quarter = 2 (April, May, June). 30 (April) + 31 (May) + 28 (June current day) = 89. DOQ recognizes leap year days.</p>
DOW	<p>The day of the week (0 - 6; Sunday is 0).</p> <pre>SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');</pre> <p>Result: 5</p> <pre>SELECT EXTRACT(DOW FROM DATE '2001-02-16');</pre> <p>Result: 5</p> <p>Note that EXTRACT's day of the week numbering is different from that of the TO_CHAR function.</p>
DOY	<p>The day of the year (1 - 365/366)</p> <pre>SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40');</pre> <p>Result: 47</p> <pre>SELECT EXTRACT(DOY FROM DATE '2001-02-16');</pre> <p>Result: 5</p>
EPOCH	<p>For DATE and TIMESTAMP values, the number of seconds since 1970-01-01 00:00:00-00 (can be negative); for INTERVAL values, the total number of seconds in the interval.</p>

	<pre>SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40-08'); Result: 982384720 SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours'); Result: 442800</pre> <p><b>Here is how you can convert an epoch value back to a timestamp:</b></p> <pre>SELECT TIMESTAMP WITH TIME ZONE 'epoch' + 982384720 * INTERVAL '1 second';</pre>
HOURL	<p><b>The hour field (0 - 23).</b></p> <pre>SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 20 SELECT EXTRACT(HOUR FROM TIME '13:45:59'); Result: 13</pre>
ISODOW	<p><b>The ISO day of the week (1 - 7; Monday is 1).</b></p> <pre>SELECT EXTRACT(ISODOW FROM DATE '2010-09-27'); Result: 1</pre>
ISOYEAR	<p><b>The ISO year, which is 52 or 53 weeks (Monday - Sunday).</b></p> <pre>SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01'); Result: 2005 SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02'); Result: 2006 SELECT EXTRACT(ISOYEAR FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 2001</pre>
MICROSECONDS	<p><b>The seconds field, including fractional parts, multiplied by 1,000,000. This includes full seconds.</b></p> <pre>SELECT EXTRACT(MICROSECONDS FROM TIME '17:12:28.5'); Result: 28500000</pre>
MILLENNIUM	<p><b>The millennium number.</b></p> <pre>SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 3</pre> <p>Years in the 1900s are in the second millennium. The third millennium starts January 1, 2001.</p>
MILLISECONDS	<p><b>The seconds field, including fractional parts, multiplied by 1000. Note that this includes full seconds.</b></p> <pre>SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5'); Result: 28500</pre>
MINUTE	<p><b>The minutes field (0 - 59).</b></p> <pre>SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 38 SELECT EXTRACT(MINUTE FROM TIME '13:45:59'); Result: 45</pre>
MONTH	<p><b>For timestamp values, the number of the month within the year (1 - 12) ; for interval values the number of months, modulo 12 (0 - 11).</b></p> <pre>SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 2 SELECT EXTRACT(MONTH FROM INTERVAL '2 years 3 months'); Result: 3 SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months'); Result: 1</pre>
QUARTER	<p><b>The quarter of the year (1 - 4) that the day is in (for timestamp values only).</b></p> <pre>SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 1</pre>
SECOND	<p><b>The seconds field, including fractional parts (0 - 59) (60 if leap seconds are implemented by the operating system).</b></p> <pre>SELECT EXTRACT(SECOND FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 40</pre>

	<pre>SELECT EXTRACT(SECOND FROM TIME '17:12:28.5'); Result: 28.5</pre>
TIME_ZONE	The time zone offset from UTC, measured in seconds. Positive values correspond to time zones east of UTC, negative values to zones west of UTC.
TIMEZONE_HOUR	The hour component of the time zone offset.
TIMEZONE_MINUTE	The minute component of the time zone offset.
WEEK	<p>The number of the week of the year that the day is in. By definition, the ISO-8601 week starts on Monday, and the first week of a year contains January 4 of that year. In other words, the first Thursday of a year is in week 1 of that year.</p> <p>Because of this, it is possible for early January dates to be part of the 52nd or 53rd week of the previous year. For example, 2005-01-01 is part of the 53rd week of year 2004, and 2006-01-01 is part of the 52nd week of year 2005.</p> <pre>SELECT EXTRACT(WEEK FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 7 SELECT EXTRACT(WEEK FROM DATE '2001-02-16'); Result: 7</pre>
YEAR	<p>The year field. Keep in mind there is no 0 AD, so subtract BC years from AD years with care.</p> <pre>SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 2001</pre>

### Examples

The following example extracts the day value from the input parameters:

```
SELECT DATE_PART('day', TIMESTAMP '2009-02-24 20:38:40') "Day";
Day
-----
 24
(1 row)
```

The following example extracts the month value from the input parameters:

```
SELECT DATE_PART('month', TIMESTAMP '2009-02-24 20:38:40') "Month";
Month
-----
 2
(1 row)
```

The following example extracts the year value from the input parameters:

```
SELECT DATE_PART('year', TIMESTAMP '2009-02-24 20:38:40') "Year";
Year
-----
2009
(1 row)
```

The following example extracts the hours from the input parameters:

```
SELECT DATE_PART('hour', TIMESTAMP '2009-02-24 20:38:40') "Hour";
Hour
-----
 20
(1 row)
```

The following example extracts the minutes from the input parameters:

```
SELECT DATE_PART('minutes', TIMESTAMP '2009-02-24 20:38:40') "Minutes";
Minutes
-----
      38
(1 row)
```

The following example extracts the seconds from the input parameters:

```
SELECT DATE_PART('seconds', TIMESTAMP '2009-02-24 20:38:40') "Seconds";
Seconds
-----
      40
(1 row)
```

The following example extracts the day of quarter (DOQ) from the input parameters:

```
SELECT DATE_PART('DOQ', TIMESTAMP '2009-02-24 20:38:40') "DOQ";
DOQ
-----
    55
(1 row)
```

```
SELECT DATE_PART('day', INTERVAL '29 days 23 hours');
date_part
-----
      29
(1 row)
```

Notice what happens to the above query if you add an hour:

```
SELECT DATE_PART('day', INTERVAL '29 days 24 hours');
date_part
-----
      30
(1 row)
```

The following example returns 0 because an interval in hours is up to 24 only:

```
SELECT DATE_PART('hour', INTERVAL '24 hours 45 minutes');
date_part
-----
      0
(1 row)
```

### See Also

**EXTRACT** (page 207)

## DATE

Converts a `TIMESTAMP`, `TIMESTAMPZ`, `DATE`, or `VARCHAR` to a `DATE`. You can also use this function to convert an `INTEGER` to a `DATE`. In this case, the resulting date reflects the *int* number of days after 0001 AD. (Day 1 is January 1, 0001.)

### Syntax

```
DATE ( d | n )
```

## Behavior type

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

## Parameters

<i>d</i>	Is the TIMESTAMP, TIMESTAMPTZ, VARCHAR, or DATE input value.
<i>n</i>	Is the integer you want to convert to a DATE.

## Example

```
=> SELECT DATE (1);
      DATE
-----
 0001-01-01
(1 row)

=> SELECT DATE (734260);
      DATE
-----
 2011-05-03
(1 row)

=> SELECT DATE ('TODAY');
      DATE
-----
 2011-05-31
(1 row)
```

## DATE\_TRUNC

Is conceptually similar to the **TRUNC** (page 297) function for numbers. The return value is of type TIMESTAMP or INTERVAL with all fields that are less significant than the selected one set to zero (or one, for day and month).

## Behavior Type

Stable when source is of type TIMESTAMPTZ, Immutable otherwise.

## Syntax

```
DATE_TRUNC ( field , source )
```

## Parameters

<i>field</i>	Is a string constant that selects the precision to which truncate the input value. Valid values for <i>field</i> are:  century            milliseconds day                minute decade             month
--------------	---

	hour            second microseconds   week millennium     year
<i>source</i>	Is a value expression of type <code>TIMESTAMP</code> or <code>INTERVAL</code> . Values of type <code>DATE</code> and <code>TIME</code> are cast automatically, to <code>TIMESTAMP</code> or <code>INTERVAL</code> , respectively.

## Examples

The following example returns the hour and truncates the minutes and seconds:

```
SELECT DATE_TRUNC('hour', TIMESTAMP '2009-02-24 13:38:40') AS hour;
      hour
-----
2009-02-24 13:00:00
(1 row)
```

The following example returns the year and defaults month and day to January 1, truncating the rest of the string:

```
SELECT DATE_TRUNC('year', TIMESTAMP '2009-02-24 13:38:40') AS year;
      year
-----
2009-01-01 00:00:00
(1 row)
```

The following example returns the year and month and defaults day of month to 1, truncating the rest of the string:

```
SELECT DATE_TRUNC('month', TIMESTAMP '2009-02-24 13:38:40') AS year;
      year
-----
2009-02-01 00:00:00
(1 row)
```

## DATEDIFF

Returns the difference between two date or time values, based on the specified start and end arguments.

### Behavior Type

Immutable, except for `TIMESTAMPTZ` arguments where it is Stable.

### Syntax 1

```
DATEDIFF ( datepart , startdate , enddate );
```

### Syntax 2

```
DATEDIFF ( datepart , starttime , endtime );
```

## Parameters

<i>datepart</i>	<p>Returns the number of specified datepart boundaries between the specified startdate and enddate.</p> <p>Can be an unquoted identifier, a quoted string, or an expression in parentheses, which evaluates to the datepart as a character string.</p> <p>The following table lists the valid <i>datepart</i> arguments.</p> <table border="1"> <thead> <tr> <th>datepart</th> <th>abbreviation</th> </tr> </thead> <tbody> <tr> <td>year</td> <td>yy, yyyy</td> </tr> <tr> <td>quarter</td> <td>qq, q</td> </tr> <tr> <td>month</td> <td>mm, m</td> </tr> <tr> <td>day</td> <td>dd, d, dy, dayofyear, y</td> </tr> <tr> <td>week</td> <td>wk, ww</td> </tr> <tr> <td>hour</td> <td>hh</td> </tr> <tr> <td>minute</td> <td>mi, n</td> </tr> <tr> <td>second</td> <td>ss, s</td> </tr> <tr> <td>millisecond</td> <td>ms</td> </tr> <tr> <td>microsecond</td> <td>mcs, us</td> </tr> </tbody> </table>	datepart	abbreviation	year	yy, yyyy	quarter	qq, q	month	mm, m	day	dd, d, dy, dayofyear, y	week	wk, ww	hour	hh	minute	mi, n	second	ss, s	millisecond	ms	microsecond	mcs, us
datepart	abbreviation																						
year	yy, yyyy																						
quarter	qq, q																						
month	mm, m																						
day	dd, d, dy, dayofyear, y																						
week	wk, ww																						
hour	hh																						
minute	mi, n																						
second	ss, s																						
millisecond	ms																						
microsecond	mcs, us																						
<i>startdate</i>	<p>Is the start date for the calculation and is an expression that returns a <b>TIMESTAMP</b> (page 90), <b>DATE</b> (page 73), or <b>TIMESTAMPZ</b> value.</p> <p>The <i>startdate</i> value is not included in the count.</p>																						
<i>enddate</i>	<p>Is the end date for the calculation and is an expression that returns a <b>TIMESTAMP</b> (page 90), <b>DATE</b> (page 73), or <b>TIMESTAMPZ</b> value.</p> <p>The <i>enddate</i> value is included in the count.</p>																						
<i>starttime</i>	<p>Is the start time for the calculation and is an expression that returns an <b>INTERVAL</b> (page 74) or <b>TIME</b> (page 88) data type.</p> <ul style="list-style-type: none"> <li>▪ The <i>starttime</i> value is not included in the count.</li> <li>▪ Year, quarter, or month <i>dateparts</i> are not allowed.</li> </ul>																						
<i>endtime</i>	<p>Is the end time for the calculation and is an expression that returns an <b>INTERVAL</b> (page 74) or <b>TIME</b> (page 88) data type.</p> <ul style="list-style-type: none"> <li>▪ The <i>endtime</i> value is included in the count.</li> <li>▪ Year, quarter, or month <i>dateparts</i> are not allowed.</li> </ul>																						

## Notes

- DATEDIFF() is an immutable function with a default type of **TIMESTAMP**. It also takes **DATE**. If **TIMESTAMPZ** is specified, the function is stable.
- Vertica accepts statements written in any of the following forms:
 

```
DATEDIFF(year, s, e);
DATEDIFF('year', s, e);
```

If you use an expression, the expression must be enclosed in parentheses:

```
DATEDIFF((expression), s, e);
```

- Starting arguments are not included in the count, but end arguments are included.

### The datepart boundaries

DATEDIFF calculates results according to ticks—or boundaries—within the date range or time range. Results are calculated based on the specified *datepart*. Let's examine the following statement and its results:

```
SELECT DATEDIFF('year', TO_DATE('01-01-2005', 'MM-DD-YYYY'),
TO_DATE('12-31-2008', 'MM-DD-YYYY'));
datediff
-----
          3
(1 row)
```

In the above example, we specified a *datepart* of year, a *startdate* of January 1, 2005 and an *enddate* of December 31, 2008. DATEDIFF returns 3 by counting the year intervals as follows:

```
[1] January 1, 2006 + [2] January 1, 2007 + [3] January 1, 2008 = 3
```

The function returns 3, and not 4, because *startdate* (January 1, 2005) is not counted in the calculation. DATEDIFF also ignores the months between January 1, 2008 and December 31, 2008 because the *datepart* specified is year and only the start of each year is counted.

Sometimes the *enddate* occurs earlier in the ending year than the *startdate* in the starting year. For example, assume a *datepart* of year, a *startdate* of August 15, 2005, and an *enddate* of January 1, 2009. In this scenario, less than three years have elapsed, but DATEDIFF counts the same way it did in the previous example, returning 3 because it returns the number of January 1s between the limits:

```
[1] January 1, 2006 + [2] January 1, 2007 + [3] January 1, 2008 = 3
```

In the following query, Vertica recognizes the full year 2005 as the starting year and 2009 as the ending year.

```
SELECT DATEDIFF('year', TO_DATE('08-15-2005', 'MM-DD-YYYY'),
TO_DATE('01-01-2009', 'MM-DD-YYYY'));
```

The count occurs as follows:

```
[1] January 1, 2006 + [2] January 1, 2007 + [3] January 1, 2008 + [4] January 1,
2009 = 4
```

Even though August 15 has not yet occurred in the *enddate*, the function counts the entire *enddate* year as one tick or boundary because of the year *datepart*.

### Examples

**Year:** In this example, the *startdate* and *enddate* are adjacent. The difference between the dates is one time boundary (second) of its *datepart*, so the result set is 1.

```
SELECT DATEDIFF('year', TIMESTAMP '2008-12-31 23:59:59',
'2009-01-01 00:00:00');
datediff
-----
          1
(1 row)
```

**Quarters** start on January, April, July, and October.

In the following example, the result is 0 because the difference from January to February in the same calendar year does not span a quarter:

```
SELECT DATEDIFF('qq', TO_DATE('01-01-1995', 'MM-DD-YYYY'),
  TO_DATE('02-02-1995', 'MM-DD-YYYY'));
datediff
-----
          0
(1 row)
```

The next example, however, returns 8 quarters because the difference spans two full years. The extra month is ignored:

```
SELECT DATEDIFF('quarter', TO_DATE('01-01-1993', 'MM-DD-YYYY'),
  TO_DATE('02-02-1995', 'MM-DD-YYYY'));
datediff
-----
          8
(1 row)
```

**Months** are based on real calendar months.

The following statement returns 1 because there is month difference between January and February in the same calendar year:

```
SELECT DATEDIFF('mm', TO_DATE('01-01-2005', 'MM-DD-YYYY'),
  TO_DATE('02-02-2005', 'MM-DD-YYYY'));
datediff
-----
          1
(1 row)
```

The next example returns a negative value of 1:

```
SELECT DATEDIFF('month', TO_DATE('02-02-1995', 'MM-DD-YYYY'),
  TO_DATE('01-01-1995', 'MM-DD-YYYY'));
datediff
-----
         -1
(1 row)
```

And this third example returns 23 because there are 23 months difference between

```
SELECT DATEDIFF('m', TO_DATE('02-02-1993', 'MM-DD-YYYY'),
  TO_DATE('01-01-1995', 'MM-DD-YYYY'));
datediff
-----
         23
(1 row)
```

**Weeks** start on Sunday at midnight.

The first example returns 0 because, even though the week starts on a Sunday, it is not a full calendar week:

```
SELECT DATEDIFF('ww', TO_DATE('02-22-2009', 'MM-DD-YYYY'),
```

```

    TO_DATE('02-28-2009','MM-DD-YYYY');
datediff
-----
          0
(1 row)

```

The following example returns 1 (week); January 1, 2000 fell on a Saturday.

```

SELECT DATEDIFF('week', TO_DATE('01-01-2000','MM-DD-YYYY'),
    TO_DATE('01-02-2000','MM-DD-YYYY'));
datediff
-----
          1
(1 row)

```

In the next example, DATEDIFF() counts the weeks between January 1, 1995 and February 2, 1995 and returns 4 (weeks):

```

SELECT DATEDIFF('wk', TO_DATE('01-01-1995','MM-DD-YYYY'),
    TO_DATE('02-02-1995','MM-DD-YYYY'));
datediff
-----
          4
(1 row)

```

The next example returns a difference of 100 weeks:

```

SELECT DATEDIFF('ww', TO_DATE('02-02-2006','MM-DD-YYYY'),
    TO_DATE('01-01-2008','MM-DD-YYYY'));
datediff
-----
        100
(1 row)

```

**Days** are based on real calendar days.

The first example returns 31, the full number of days in the month of July 2008.

```

SELECT DATEDIFF('day', 'July 1, 2008', 'Aug 1, 2008'::date);
datediff
-----
        31
(1 row)

```

Just over two years of days:

```

SELECT DATEDIFF('d', TO_TIMESTAMP('01-01-1993','MM-DD-YYYY'),
    TO_TIMESTAMP('02-02-1995','MM-DD-YYYY'));
datediff
-----
        762
(1 row)

```

**Hours, minutes, and seconds** are based on clock time.

The first example counts backwards from March 2 to February 14 and returns -384 hours:

```

SELECT DATEDIFF('hour', TO_DATE('03-02-2009','MM-DD-YYYY'),
    TO_DATE('02-14-2009','MM-DD-YYYY'));
datediff
-----

```

```
      -384
(1 row)
```

Another hours example:

```
SELECT DATEDIFF('hh', TO_TIMESTAMP('01-01-1993', 'MM-DD-YYYY'),
  TO_TIMESTAMP('02-02-1995', 'MM-DD-YYYY'));
datediff
-----
      18288
(1 row)
```

This example counts the minutes backwards:

```
SELECT DATEDIFF('mi', TO_TIMESTAMP('01-01-1993 03:00:45', 'MM-DD-YYYY HH:MI:SS'),
  TO_TIMESTAMP('01-01-1993 01:30:21', 'MM-DD-YYYY HH:MI:SS'));
datediff
-----
      -90
(1 row)
```

And this example counts the minutes forward:

```
SELECT DATEDIFF('minute', TO_DATE('01-01-1993', 'MM-DD-YYYY'),
  TO_DATE('02-02-1995', 'MM-DD-YYYY'));
datediff
-----
    1097280
(1 row)
```

In the following example, the query counts the difference in seconds, beginning at a start time of 4:44 and ending at 5:55 with an interval of 2 days:

```
SELECT DATEDIFF('ss', TIME '04:44:42.315786',
  INTERVAL '2 05:55:52.963558');
datediff
-----
    177070
(1 row)
```

## See Also

***Date/Time Expressions*** (page 48)

## DAY

Extracts the day of the month from a `TIMESTAMP`, `TIMESTAMPTZ`, `INTEGER`, `VARCHAR` or `INTERVAL` input value. The return value is of type `INTEGER`.

## Syntax

```
DAY (d )
```

## Behavior type

Immutable, except for `TIMESTAMPTZ` arguments where it is Stable.

**Parameters**

<i>d</i>	Is the <code>TIMESTAMP</code> , <code>TIMESTAMPTZ</code> , <code>INTERVAL</code> , <code>VARCHAR</code> , or <code>INTEGER</code> input value.
----------	--

**Examples**

```
=> SELECT DAY (6);
DAY
-----
    6
(1 row)
=> SELECT DAY(TIMESTAMP 'sep 22, 2011 12:34');
DAY
-----
   22
(1 row)
=> SELECT DAY('sep 22, 2011 12:34');
DAY
-----
   22
(1 row)
=> SELECT DAY(INTERVAL '35 12:34');
DAY
-----
   35
(1 row)
```

**DAYOFMONTH**

Returns an integer representing the day of the month based on a `VARCHAR`, `DATE`, `TIMESTAMP`, OR `TIMESTAMPTZ` input value.

**Syntax**

```
DAYOFMONTH ( d )
```

**Behavior type**

Immutable, except for `TIMESTAMPTZ` arguments where it is Stable.

**Parameters**

<i>d</i>	Is the <code>VARCHAR</code> , <code>DATE</code> , <code>TIMESTAMP</code> , or <code>TIMESTAMPTZ</code> input value.
----------	---

**Example**

```
=> SELECT DAYOFMONTH (TIMESTAMP 'sep 22, 2011 12:34');
DAYOFMONTH
-----
          22
(1 row)
```

## DAYOFWEEK

Returns an INTEGER representing the day of the week based on a `TIMESTAMP`, `TIMESTAMPTZ`, `VARCHAR`, or `DATE` input value. Valid return values are:

Integer	Week Day
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

### Syntax

```
DAYOFWEEK ( d )
```

### Behavior type

Immutable, except for `TIMESTAMPTZ` arguments where it is Stable.

### Parameters

<i>d</i>	Is the <code>TIMESTAMP</code> , <code>TIMESTAMPTZ</code> , <code>VARCHAR</code> , or <code>DATE</code> input value.
----------	---

### Example

```
=> SELECT DAYOFWEEK (TIMESTAMP 'sep 17, 2011 12:34');  
DAYOFWEEK
```

```
-----  
7
```

```
(1 row)
```

## DAYOFWEEK\_ISO

Returns an INTEGER representing the ISO 8061 day of the week based on a `VARCHAR`, `DATE`, `TIMESTAMP` or `TIMESTAMPTZ` input value. Valid return values are:

Integer	Week Day
1	Monday
2	Tuesday
3	Wednesday

4	Thursday
5	Friday
6	Saturday
7	Sunday

**Syntax**

```
DAYOFWEEK_ISO ( d )
```

**Behavior type**

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

**Parameters**

<i>d</i>	Is the VARCHAR, DATE, TIMESTAMP, or TIMESTAMPTZ input value.
----------	--

**Example**

```
=> SELECT DAYOFWEEK_ISO(TIMESTAMP 'Sep 22, 2011 12:34');
   DAYOFWEEK_ISO
-----
                4
(1 row)
```

The following example shows how to combine the DAYOFWEEK\_ISO, WEEK\_ISO, and YEAR\_ISO functions to find the ISO day of the week, week, and year:

```
=> SELECT DAYOFWEEK_ISO('Jan 1, 2000'), WEEK_ISO('Jan 1,
2000'), YEAR_ISO('Jan1,2000');
   DAYOFWEEK_ISO | WEEK_ISO | YEAR_ISO
-----+-----+-----
                6 |         52 |      1999
(1 row)
```

**See Also**

**WEEK\_ISO** (page 238)

**DAYOFWEEK\_ISO** (page 204)

[http://en.wikipedia.org/wiki/ISO\\_8601](http://en.wikipedia.org/wiki/ISO_8601) ([http://en.wikipedia.org/wiki/ISO\\_8601](http://en.wikipedia.org/wiki/ISO_8601))

**DAYOFYEAR**

Returns an INTEGER representing the day of the year based on a TIMESTAMP, TIMESTAMPTZ, VARCHAR, or DATE input value. (January 1 is day 1.)

**Syntax**

```
DAYOFYEAR ( d )
```

**Behavior type**

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

**Parameters**

<i>d</i>	Is the TIMESTAMP, TIMESTAMPTZ, VARCHAR, OR DATE input value.
----------	--

**Example**

```
=> SELECT DAYOFYEAR (TIMESTAMP 'SEPT 22,2011 12:34');
   DAYOFYEAR
-----
          265
(1 row)
```

**DAYS**

Converts a DATE, VARCHAR, TIMESTAMP, or TIMESTAMPTZ to an INTEGER, reflecting the number of days after 0001 AD.

**Syntax**

```
DAYS( DATE d )
```

**Behavior type**

Immutable

**Parameters**

DATE <i>d</i>	Is the VARCHAR, DATE, TIMESTAMP, or TIMESTAMPTZ input value.
---------------	--

**Example**

```
=> SELECT DAYS (DATE '2011-01-22');
   DAYS
-----
 734159
(1 row)

=> SELECT DAYS ('1999-12-31');
   DAYS
-----
 730119
(1 row)
```

## EXTRACT

Retrieves subfields such as year or hour from date/time values and returns values of type **DOUBLE PRECISION** (page 97). EXTRACT is primarily intended for computational processing, rather than for formatting date/time values for display.

Internally EXTRACT uses the DATE\_PART function.

### Behavior Type

Stable when source is of type TIMESTAMPTZ, Immutable otherwise.

### Syntax

```
EXTRACT ( field FROM source )
```

### Parameters

<i>field</i>	Is an identifier or string that selects what field to extract from the source value. <b>Note:</b> The field parameter is the same for the DATE_PART() (page 191) function.
<i>source</i>	Is an expression of type DATE, TIMESTAMP, TIME, or INTERVAL. <b>Note:</b> Expressions of type DATE are cast to TIMESTAMP.

### Field Values

CENTURY	<p>The century number.</p> <pre>SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13'); Result: 20 SELECT EXTRACT(CENTURY FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 21</pre> <p>The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries. There is no century number 0, you go from -1 to 1.</p>
DAY	<p>The day (of the month) field (1 - 31).</p> <pre>SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 16 SELECT EXTRACT(DAY FROM DATE '2001-02-16'); Result: 16</pre>
DECADE	<p>The year field divided by 10.</p> <pre>SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 200 SELECT EXTRACT(DECADE FROM DATE '2001-02-16'); Result: 200</pre>
DOQ	<p>The day within the current quarter.</p> <pre>SELECT EXTRACT(DOQ FROM CURRENT_DATE); Result: 89</pre> <p>The result is calculated as follows: Current date = June 28, current quarter = 2 (April, May, June). 30 (April) + 31 (May) + 28 (June current day) = 89. DOQ recognizes leap year days.</p>
DOW	The day of the week (0 - 6; Sunday is 0).

	<pre>SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 5 SELECT EXTRACT(DOW FROM DATE '2001-02-16'); Result: 5</pre> <p>Note that EXTRACT's day of the week numbering is different from that of the TO_CHAR function.</p>
DOY	<p><b>The day of the year (1 - 365/366)</b></p> <pre>SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 47 SELECT EXTRACT(DOY FROM DATE '2001-02-16'); Result: 5</pre>
EPOCH	<p><b>For DATE and TIMESTAMP values, the number of seconds since 1970-01-01 00:00:00-00 (can be negative); for INTERVAL values, the total number of seconds in the interval.</b></p> <pre>SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40-08'); Result: 982384720 SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours'); Result: 442800</pre> <p>Here is how you can convert an epoch value back to a timestamp:</p> <pre>SELECT TIMESTAMP WITH TIME ZONE 'epoch' + 982384720 * INTERVAL '1 second';</pre>
HOURL	<p><b>The hour field (0 - 23).</b></p> <pre>SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 20 SELECT EXTRACT(HOUR FROM TIME '13:45:59'); Result: 13</pre>
ISODOW	<p><b>The ISO day of the week (1 - 7; Monday is 1).</b></p> <pre>SELECT EXTRACT(ISODOW FROM DATE '2010-09-27'); Result: 1</pre>
ISOYEAR	<p><b>The ISO year, which is 52 or 53 weeks (Monday - Sunday).</b></p> <pre>SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01'); Result: 2005 SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02'); Result: 2006 SELECT EXTRACT(ISOYEAR FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 2001</pre>
MICROSECONDS	<p><b>The seconds field, including fractional parts, multiplied by 1,000,000. This includes full seconds.</b></p> <pre>SELECT EXTRACT(MICROSECONDS FROM TIME '17:12:28.5'); Result: 28500000</pre>
MILLENNIUM	<p><b>The millennium number.</b></p> <pre>SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 3</pre> <p>Years in the 1900s are in the second millennium. The third millennium starts January 1, 2001.</p>
MILLISECONDS	<p><b>The seconds field, including fractional parts, multiplied by 1000. Note that this includes full seconds.</b></p> <pre>SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5'); Result: 28500</pre>
MINUTE	<p><b>The minutes field (0 - 59).</b></p> <pre>SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40'); Result: 38 SELECT EXTRACT(MINUTE FROM TIME '13:45:59');</pre>

	<i>Result: 45</i>
MONTH	<p>For timestamp values, the number of the month within the year (1 - 12) ; for interval values the number of months, modulo 12 (0 - 11).</p> <p>SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40');  <i>Result: 2</i>  SELECT EXTRACT(MONTH FROM INTERVAL '2 years 3 months');  <i>Result: 3</i>  SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months');  <i>Result: 1</i></p>
QUARTER	<p>The quarter of the year (1 - 4) that the day is in (for timestamp values only).</p> <p>SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40');  <i>Result: 1</i></p>
SECOND	<p>The seconds field, including fractional parts (0 - 59) (60 if leap seconds are implemented by the operating system).</p> <p>SELECT EXTRACT(SECOND FROM TIMESTAMP '2001-02-16 20:38:40');  <i>Result: 40</i>  SELECT EXTRACT(SECOND FROM TIME '17:12:28.5');  <i>Result: 28.5</i></p>
TIME_ZONE	The time zone offset from UTC, measured in seconds. Positive values correspond to time zones east of UTC, negative values to zones west of UTC.
TIMEZONE_HOUR	The hour component of the time zone offset.
TIMEZONE_MINUTE	The minute component of the time zone offset.
WEEK	<p>The number of the week of the year that the day is in. By definition, the ISO-8601 week starts on Monday, and the first week of a year contains January 4 of that year. In other words, the first Thursday of a year is in week 1 of that year.</p> <p>Because of this, it is possible for early January dates to be part of the 52nd or 53rd week of the previous year. For example, 2005-01-01 is part of the 53rd week of year 2004, and 2006-01-01 is part of the 52nd week of year 2005.</p> <p>SELECT EXTRACT(WEEK FROM TIMESTAMP '2001-02-16 20:38:40');  <i>Result: 7</i>  SELECT EXTRACT(WEEK FROM DATE '2001-02-16');  <i>Result: 7</i></p>
YEAR	<p>The year field. Keep in mind there is no 0 AD, so subtract BC years from AD years with care.</p> <p>SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40');  <i>Result: 2001</i></p>

## Examples

```
=> SELECT EXTRACT (DAY FROM DATE '2008-12-25');
date_part
-----
          25
(1 row)
=> SELECT EXTRACT (MONTH FROM DATE '2008-12-25');
date_part
-----
          12
(1 row)
SELECT EXTRACT(DOY FROM CURRENT_DATE);
date_part
-----
          89
```

(1 row)

Remember that internally `EXTRACT()` uses the `DATE_PART()` function:

```
=> SELECT EXTRACT(EPOCH FROM AGE_IN_YEARS(TIMESTAMP '2009-02-24',
      TIMESTAMP '1972-03-02')) :: INTERVAL year);
```

```
date_part
-----
1136073600
(1 row)
```

In the above example, `AGE_IN_YEARS` is 36. The UNIX epoch uses 365.25 days per year:

```
=> SELECT 1136073600.0/36/(24*60*60);
?column?
-----
365.25
(1 row)
```

### See Also

***DATE\_PART*** (page 191)

## GETDATE

Returns the current system date and time as a `TIMESTAMP` value.

### Behavior Type

Stable

### Syntax

```
GETDATE();
```

### Notes

- `GETDATE` is a stable function that requires parentheses but accepts no arguments.
- This function uses the date and time supplied by the operating system on the server to which you are connected, which is the same across all servers.
- `GETDATE` internally converts ***STATEMENT\_TIMESTAMP()*** (page 223) from `TIMESTAMPTZ` to `TIMESTAMP`.
- This function is identical to ***SYSDATE()*** (page 224).

### Example

```
=> SELECT GETDATE();
      GETDATE
-----
2011-03-07 13:21:29.497742
(1 row)
```

### See Also

***Date/Time Expressions*** (page 48)

## GETUTCDATE

Returns the current system date and time as a `TIMESTAMP` value relative to UTC.

### Behavior Type

Stable

### Syntax

```
GETUTCDATE ();
```

### Notes

- `GETUTCDATE` is a stable function that requires parentheses but accepts no arguments.
- This function uses the date and time supplied by the operating system on the server to which you are connected, which is the same across all servers.
- `GETUTCDATE` is internally converted to `STATEMENT_TIMESTAMP()` at `TIME_ZONE 'UTC'`.

### Example

```
=> SELECT GETUTCDATE ();
      GETUTCDATE
```

```
-----
2011-03-07 20:20:26.193052
(1 row)
```

### See Also

*Date/Time Expressions* (page 48)

## HOUR

Extracts the hour from a `DATE`, `TIMESTAMP`, `TIMESTAMPZ`, `VARCHAR`, or `INTERVAL` value. The return value is of type `INTEGER`. (Hour 0 is midnight to 1 a.m.)

### Syntax

```
HOUR ( d )
```

### Behavior type

Immutable, except for `TIMESTAMPZ` arguments where it is Stable.

### Parameters

<i>d</i>	Is the incoming <code>DATE</code> , <code>TIMESTAMP</code> , <code>TIMESTAMPZ</code> , <code>VARCHAR</code> , or <code>INTERVAL</code> value.
----------	---

### Example

```
=> SELECT HOUR (TIMESTAMP 'sep 22, 2011 12:34');
      HOUR
-----
12
(1 row)
```

```
=> SELECT HOUR (INTERVAL '35 12:34');
      HOUR
-----
       12
(1 row)
=> SELECT HOUR ('12:34');
      HOUR
-----
       12
(1 row)
```

## ISFINITE

Tests for the special `TIMESTAMP` constant `INFINITY` and returns a value of type `BOOLEAN`.

### Behavior Type

Immutable

### Syntax

```
ISFINITE ( timestamp )
```

### Parameters

<i>timestamp</i>	Is an expression of type <code>TIMESTAMP</code>
------------------	---

### Examples

```
SELECT ISFINITE(TIMESTAMP '2009-02-16 21:28:30');
      isfinite
-----
       t
(1 row)
SELECT ISFINITE(TIMESTAMP 'INFINITY');
      isfinite
-----
       f
(1 row)
```

## JULIAN\_DAY

Returns an `INTEGER` representing the Julian day based on an input `TIMESTAMP`, `TIMESTAMPTZ`, `VARCHAR`, or `DATE` value.

### Syntax

```
JULIAN_DAY ( d )
```

### Behavior type

Immutable, except for `TIMESTAMPTZ` arguments where it is Stable.

## Parameters

<i>d</i>	Is the TIMESTAMP, TIMESTAMPTZ, VARCHAR, or DATE input value.
----------	--

## Example

```
=> SELECT JULIAN_DAY(TIMESTAMP 'sep 22, 2011 12:34');
      JULIAN_DAY
-----
      2455827
(1 row)
```

## LAST\_DAY

Returns the last day of the month based on a `TIMESTAMP`. The `TIMESTAMP` can be supplied as a `DATE` or a `TIMESTAMPTZ` data type.

## Behavior Type

Immutable, unless called with `TIMESTAMPTZ`, in which case it is Stable.

## Syntax

```
LAST_DAY ( date );
```

## Examples

The following example returns the last day of the month, February, as 29 because 2008 was a leap year:

```
SELECT LAST_DAY('2008-02-28 23:30 PST') "Last";
      Last
-----
      2008-02-29
(1 row)
```

The following example returns the last day of the month in March, after converting the string value to the specified `DATE` type:

```
SELECT LAST_DAY('2003/03/15') "Last";
      Last
-----
      2003-03-31
(1 row)
```

The following example returns the last day of February in the specified year (not a leap year):

```
SELECT LAST_DAY('2003/02/03') "Last";
      Last
-----
      2003-02-28
(1 row)
```

## LOCALTIME

Returns a value of type `TIME` representing the time of day.

## Behavior Type

Stable

## Syntax

```
LOCALTIME [ ( precision ) ]
```

## Parameters

<i>precision</i>	Causes the result to be rounded to the specified number of fractional digits in the seconds field.
------------------	--

## Notes

This function returns the start time of the current transaction; the value does not change during the transaction. The intent is to allow a single transaction to have a consistent notion of the "current" time, so that multiple modifications within the same transaction bear the same timestamp.

## Examples

```
SELECT LOCALTIME;  
       time  
-----  
16:16:06.790771  
(1 row)
```

## LOCALTIMESTAMP

Returns a value of type `TIMESTAMP` representing today's date and time of day.

## Behavior Type

Stable

## Syntax

```
LOCALTIMESTAMP [ ( precision ) ]
```

## Parameters

<i>precision</i>	Causes the result to be rounded to the specified number of fractional digits in the seconds field.
------------------	--

## Notes

This function returns the start time of the current transaction; the value does not change during the transaction. The intent is to allow a single transaction to have a consistent notion of the "current" time, so that multiple modifications within the same transaction bear the same timestamp.

## Examples

```
SELECT LOCALTIMESTAMP;  
       timestamp  
-----  
2009-02-24 14:47:48.5951
```

(1 row)

## MICROSECOND

Returns an INTEGER representing the microsecond portion of an input DATE, VARCHAR, TIMESTAMP, TIMESTAMPTZ, or INTERVAL value.

### Syntax

```
MICROSECOND ( d )
```

### Behavior type

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

### Parameters

<i>d</i>	Is the DATE, VARCHAR, TIMESTAMP, TIMESTAMPTZ, or INTERVAL input value.
----------	--

### Example

```
=> SELECT MICROSECOND (TIMESTAMP 'Sep 22, 2011 12:34:01.123456');
   MICROSECOND
-----
          123456
(1 row)
```

## MIDNIGHT\_SECONDS

Returns an INTEGER representing the number of seconds between midnight and the input value. The input value can be of type VARCHAR, TIME, TIMESTAMP, or TIMESTAMPTZ.

### Syntax

```
MIDNIGHT_SECONDS ( d )
```

### Behavior type

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

### Parameters

<i>d</i>	Is the VARCHAR, TIME, TIMESTAMP, or TIMESTAMPTZ input value.
----------	--

### Example

```
=> SELECT MIDNIGHT_SECONDS ('12:34:00.987654');
   MIDNIGHT_SECONDS
-----
          45240
(1 row)
=> SELECT MIDNIGHT_SECONDS (TIME '12:34:00.987654');
```

```

MIDNIGHT_SECONDS
-----
          45240
(1 row)
=> SELECT MIDNIGHT_SECONDS (TIMESTAMP 'sep 22, 2011 12:34');
MIDNIGHT_SECONDS
-----
          45240
(1 row)

```

## MINUTE

Returns an INTEGER representing the minute value of the input value. The input value can be of type VARCHAR, DATE, TIMESTAMP, TIMESTAMPTZ, or INTERVAL.

### Syntax

```
MINUTE ( d )
```

### Behavior type

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

### Parameters

<i>d</i>	Is the VARCHAR, DATE, TIMESTAMP, TIMESTAMPTZ, or INTERVAL input value.
----------	--

### Example

```

=> SELECT MINUTE('12:34:03.456789');
MINUTE
-----
      34
(1 row)
=>SELECT MINUTE (TIMESTAMP 'sep 22, 2011 12:34');
MINUTE
-----
      34
(1 row)
=> SELECT MINUTE (INTERVAL '3512:34:03.456789');
MINUTE
-----
      34
(1 row)

```

## MONTH

Returns an INTEGER representing the month portion of the input value. The input value can be of type VARCHAR, DATE, TIMESTAMP, TIMESTAMPTZ, or INTERVAL.

**Syntax**

```
MONTH( d )
```

**Behavior type**

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

**Parameters**

<i>d</i>	Is the incoming VARCHAR, DATE, TIMESTAMP, TIMESTAMPTZ, or INTERVAL value.
----------	---

**Examples**

```
=> SELECT MONTH('6-9');
      MONTH
-----
          9
(1 row)
=> SELECT MONTH (TIMESTAMP 'sep 22, 2011 12:34');
      MONTH
-----
          9
(1 row)
=> SELECT MONTH(INTERVAL '2-35' year to month);
      MONTH
-----
         11
(1 row)
```

**MONTHS\_BETWEEN**

Returns the number of months between *date1* and *date2* as a FLOAT8. where the input arguments can be of TIMESTAMP, DATE, or TIMESTAMPTZ type.

**Behavior Type**

Immutable for TIMESTAMP and Date, Stable for TIMESTAMPTZ

**Syntax**

```
MONTHS_BETWEEN ( date1 , date2 );
```

**Parameters**

<i>date1, date2</i>	<p>If <i>date1</i> is later than <i>date2</i>, then the result is positive. If <i>date1</i> is earlier than <i>date2</i>, then the result is negative.</p> <p>If <i>date1</i> and <i>date2</i> are either the same days of the month or both are the last days of their respective month, then the result is always an integer. Otherwise MONTHS_BETWEEN returns a FLOAT8 result based on a 31-day month, which considers the difference between <i>date1</i> and <i>date2</i>.</p>
---------------------	---

## Examples

Note the following result is an integral number of days because the dates are on the same day of the month:

```
SELECT MONTHS_BETWEEN('2009-03-07 16:00'::TIMESTAMP, '2009-04-07
15:00'::TIMESTAMP);
  months_between
-----
                -1
(1 row)
```

The result from the following example returns an integral number of days because the days fall on the last day of their respective months:

```
SELECT MONTHS_BETWEEN('29Feb2000', '30Sep2000') "Months";
  Months
-----
        -7
(1 row)
```

In this example, and in the example that immediately follows it, MONTHS\_BETWEEN() returns the number of months between *date1* and *date2* as a fraction because the days do not fall on the same day or on the last day of their respective months:

```
SELECT MONTHS_BETWEEN(TO_DATE('02-02-1995','MM-DD-YYYY'),
                      TO_DATE('01-01-1995','MM-DD-YYYY')) "Months";
  Months
-----
1.03225806451613
(1 row)
SELECT MONTHS_BETWEEN(TO_DATE('2003/01/01','yyyymm/dd'),
                      TO_DATE('2003/03/14','yyyymm/dd')) "Months";
  Months
-----
-2.41935483870968
(1 row)
```

The following two examples use the same *date1* and *date2* strings, but they are cast to a different data types (TIMESTAMP and TIMESTAMPTZ). The result set is the same for both statements:

```
SELECT MONTHS_BETWEEN('2008-04-01'::timestamp, '2008-02-29'::timestamp);
  months_between
-----
1.09677419354839
(1 row)
SELECT MONTHS_BETWEEN('2008-04-01'::timestamptz, '2008-02-29'::timestamptz);
  months_between
-----
1.09677419354839
(1 row)
```

The following two examples show alternate inputs:

```
SELECT MONTHS_BETWEEN('2008-04-01'::date, '2008-02-29'::timestamp);
  months_between
```

```

-----
1.09677419354839
(1 row)
SELECT MONTHS_BETWEEN('2008-02-29'::timestamptz, '2008-04-01'::date);
months_between
-----
-1.09677419354839
(1 row)

```

## NEXT\_DAY

Returns the date of the first instance of a particular day of the week that follows the specified date.

### Behavior Type

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

### Syntax

```
NEXT_DAY( 'date', 'DOW')
```

### Parameters

<i>date</i>	Can be VARCHAR, TIMESTAMP, TIMESTAMPTZ, or DATE. Only standard English day-names and day-name abbreviations are accepted.
<i>DOW</i>	Day of week can be a CHAR/VARCHAR string or a character constant. DOW is not case sensitive and trailing spaces are ignored.

### Examples

The following example returns the date of the next Friday following the specified date. All are variations on the same query, and all return the same result:

```

=> SELECT NEXT_DAY('28-MAR-2011','FRIDAY') "NEXT DAY" FROM DUAL;
NEXT DAY
-----
2011-04-01
(1 row)
=> SELECT NEXT_DAY('March 28 2011','FRI') "NEXT DAY" FROM DUAL;
NEXT DAY
-----
2011-04-01
(1 row)
=> SELECT NEXT_DAY('3-29-11','FRI') "NEXT DAY" FROM DUAL;
NEXT DAY
-----
2011-04-01
(1 row)

```

## NOW [Date/Time]

Returns a value of type `TIMESTAMP WITH TIME ZONE` representing the start of the current transaction. `NOW` is equivalent to ***CURRENT\_TIMESTAMP*** (page 191) except that it does not accept a precision parameter.

### Behavior Type

Stable

### Syntax

```
NOW ( )
```

### Notes

This function returns the start time of the current transaction; the value does not change during the transaction. The intent is to allow a single transaction to have a consistent notion of the "current" time, so that multiple modifications within the same transaction bear the same timestamp.

### Examples

```
SELECT NOW ( ) ;
                NOW
-----
2010-04-01 15:31:12.144584-04
(1 row)
```

### See Also

***CURRENT\_TIMESTAMP*** (page 191)

## OVERLAPS

Returns true when two time periods overlap, false when they do not overlap.

### Behavior Type

Stable when `TIMESTAMP` and `TIMESTAMPTZ` are both used, or when `TIMESTAMPTZ` is used with `INTERVAL`, Immutable otherwise.

### Syntax

```
( start, end ) OVERLAPS ( start, end )
( start, interval ) OVERLAPS ( start, interval )
```

### Parameters

<i>start</i>	Is a <code>DATE</code> , <code>TIME</code> , or <code>TIME STAMP</code> value that specifies the beginning of a time period.
<i>end</i>	Is a <code>DATE</code> , <code>TIME</code> , or <code>TIME STAMP</code> value that specifies the end of a time period.
<i>interval</i>	Is a value that specifies the length of the time period.

## Examples

The first command returns true for an overlap in date range of 2007-02-16 – 2007-12-21 with 2007-10-30 – 2008-10-30.

```
SELECT (DATE '2007-02-16', DATE '2007-12-21')
       OVERLAPS (DATE '2007-10-30', DATE '2008-10-30');
overlaps
-----
t
(1 row)
```

The next command returns false for an overlap in date range of 2007-02-16 – 2007-12-21 with 2008-10-30 – 2008-10-30.

```
SELECT (DATE '2007-02-16', DATE '2007-12-21')
       OVERLAPS (DATE '2008-10-30', DATE '2008-10-30');
overlaps
-----
f
(1 row)
```

The next command returns false for an overlap in date range of 2007-02-16, 22 hours ago with 2007-10-30, 22 hours ago.

```
SELECT (DATE '2007-02-16', INTERVAL '1 12:59:10')
       OVERLAPS (DATE '2007-10-30', INTERVAL '1 12:59:10');
overlaps
-----
f
(1 row)
```

## QUARTER

Returns an INTEGER representing calendar quarter into which the input value falls. The input value can be of type VARCHAR, DATE, TIMESTAMP or TIMESTAMPTZ.

### Syntax

```
QUARTER( d )
```

### Behavior type

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

### Parameters

<i>d</i>	Is the DATE, VARCHAR, TIMESTAMP, or TIMESTAMPTZ input value.
----------	--

### Example

```
=> SELECT QUARTER (TIMESTAMP 'sep 22, 2011 12:34');
QUARTER
-----
3
```

(1 row)

## ROUND [Date/Time]

Rounds a `TIMESTAMP`, `TIMESTAMPTZ`, or `DATE`. The return value is of type `TIMESTAMP`.

### Behavior Type

Immutable, except for `TIMESTAMPTZ` arguments where it is Stable.

### Syntax

```
ROUND([TIMESTAMP | DATE] , format )
```

### Parameters

<code>TIMESTAMP   DATE</code>	Is the <code>TIMESTAMP</code> or <code>DATE</code> input value.																												
<code>format</code>	<p>Is a string constant that selects the precision to which truncate the input value. Valid values for <code>format</code> are:</p> <table border="1"> <thead> <tr> <th>Precision</th> <th>Valid values</th> </tr> </thead> <tbody> <tr> <td>Century</td> <td>CC, SCC</td> </tr> <tr> <td>Year</td> <td>SYYY, YYYY, YEAR, YYY, YY,Y</td> </tr> <tr> <td>ISO Year</td> <td>IYYY, IYY, IY, I</td> </tr> <tr> <td>Quarter</td> <td>Q</td> </tr> <tr> <td>Month</td> <td>MONTH, MON, MM, RM</td> </tr> <tr> <td>Same day of the week as the first day of the year</td> <td>WW</td> </tr> <tr> <td>Same day of the week as the first day of the ISO year</td> <td>IW</td> </tr> <tr> <td>Same day of the week as the first day of the month</td> <td>W</td> </tr> <tr> <td>Day</td> <td>DDD, DD, J</td> </tr> <tr> <td>Starting day of the week</td> <td>DAY, DY, D</td> </tr> <tr> <td>Hour</td> <td>HH, HH12, HH24</td> </tr> <tr> <td>Minute</td> <td>MI</td> </tr> <tr> <td>Second</td> <td>SS</td> </tr> </tbody> </table>	Precision	Valid values	Century	CC, SCC	Year	SYYY, YYYY, YEAR, YYY, YY,Y	ISO Year	IYYY, IYY, IY, I	Quarter	Q	Month	MONTH, MON, MM, RM	Same day of the week as the first day of the year	WW	Same day of the week as the first day of the ISO year	IW	Same day of the week as the first day of the month	W	Day	DDD, DD, J	Starting day of the week	DAY, DY, D	Hour	HH, HH12, HH24	Minute	MI	Second	SS
Precision	Valid values																												
Century	CC, SCC																												
Year	SYYY, YYYY, YEAR, YYY, YY,Y																												
ISO Year	IYYY, IYY, IY, I																												
Quarter	Q																												
Month	MONTH, MON, MM, RM																												
Same day of the week as the first day of the year	WW																												
Same day of the week as the first day of the ISO year	IW																												
Same day of the week as the first day of the month	W																												
Day	DDD, DD, J																												
Starting day of the week	DAY, DY, D																												
Hour	HH, HH12, HH24																												
Minute	MI																												
Second	SS																												

### Examples

```
=> SELECT ROUND(TIMESTAMP 'sep 22, 2011 12:34:00', 'dy');
      ROUND
-----
2011-09-18 00:00:00
(1 row)
```

## SECOND

Returns an INTEGER representing the second portion of the input value. The input value can be of type VARCHAR, TIMESTAMP, TIMESTAMPTZ, or INTERVAL.

### Syntax

```
SECOND( d )
```

### Behavior type

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

### Parameters

<i>d</i>	Is the VARCHAR, TIMESTAMP, TIMESTAMPTZ, or INTERVAL input value.
----------	--

### Examples

```
=> SELECT SECOND ('23:34:03.456789');
      SECOND
-----
          3
(1 row)
=> SELECT SECOND (TIMESTAMP 'sep 22, 2011 12:34');
      SECOND
-----
          0
(1 row)
=> SELECT SECOND (INTERVAL '35 12:34:03.456789');
      SECOND
-----
          3
(1 row)
```

## STATEMENT\_TIMESTAMP

Is similar to *TRANSACTION\_TIMESTAMP* (page 235). It returns a value of type TIMESTAMP WITH TIME ZONE representing the start of the current statement.

### Behavior Type

Stable

## Syntax

```
STATEMENT_TIMESTAMP()
```

## Notes

This function returns the start time of the current statement; the value does not change during the statement. The intent is to allow a single statement to have a consistent notion of the "current" time, so that multiple modifications within the same statement bear the same timestamp.

## Examples

```
SELECT STATEMENT_TIMESTAMP();
       STATEMENT_TIMESTAMP
-----
2010-04-01 15:40:42.223736-04
(1 row)
```

## See Also

**CLOCK\_TIMESTAMP** (page 189)

**TRANSACTION\_TIMESTAMP** (page 235)

## SYSDATE

Returns the current system date and time as a **TIMESTAMP** value.

## Behavior Type

Stable

## Syntax

```
SYSDATE();
```

## Notes

- **SYSDATE** is a stable function (called once per statement) that requires no arguments. Parentheses are optional.
- This function uses the date and time supplied by the operating system on the server to which you are connected, which must be the same across all servers.
- In implementation, **SYSDATE** converts **STATEMENT\_TIMESTAMP** (page 223) from **TIMESTAMPZ** to **TIMESTAMP**.
- This function is identical to **GETDATE()** (page 210).

## Examples

```
=> SELECT SYSDATE();
       sysdate
-----
2011-03-07 13:22:28.295802
(1 row)
```

**See Also*****Date/Time Expressions*** (page 48)**TIME\_SLICE**

Aggregates data by different fixed-time intervals and returns a rounded-up input `TIMESTAMP` value to a value that corresponds with the start or end of the time slice interval.

Given an input `TIMESTAMP` value, such as '2000-10-28 00:00:01', the start time of a 3-second time slice interval is '2000-10-28 00:00:00', and the end time of the same time slice is '2000-10-28 00:00:03'.

**Behavior Type**

Immutable

**Syntax**

```
TIME_SLICE(expression, slice_length,
  [ time_unit = 'SECOND' ],
  [ start_or_end = 'START' ] )
```

**Parameters**

<i>expression</i>	Is evaluated on each row. Can be either a column of type <code>TIMESTAMP</code> or a (string) constant that can be parsed into a <code>TIMESTAMP</code> value, such as '2004-10-19 10:23:54'.
<i>slice_length</i>	Is the length of the slice specified in integers. Input must be a positive integer.
<i>time_unit</i>	Is the time unit of the slice with a default of <code>SECOND</code> . Domain of possible values: { <code>HOURL</code> , <code>MINUTE</code> , <code>SECOND</code> , <code>MILLISECOND</code> , <code>MICROSECOND</code> }.
<i>start_or_end</i>	Indicates whether the returned value corresponds to the start or end time of the time slice interval. The default is <code>START</code> . Domain of possible values: { <code>START</code> , <code>END</code> }.

**Notes**

- The returned value's data type is `TIMESTAMP`.
- The corresponding SQL data type for `TIMESTAMP` is `TIMESTAMP WITHOUT TIME ZONE`. Vertica supports `TIMESTAMP` for `TIME_SLICE` instead of `DATE` and `TIME` data types.
- `TIME_SLICE` exhibits the following behavior around nulls:
  - The system returns an error when any one of *slice\_length*, *time\_unit*, or *start\_or\_end* parameters is null.
  - When *slice\_length*, *time\_unit*, and *start\_or\_end* contain legal values, and *expression* is null, the system returns a `NULL` value, instead of an error.

**Usage**

The following command returns the (default) start time of a 3-second time slice:

```
SELECT TIME_SLICE('2009-09-19 00:00:01', 3);
       time_slice
-----
2009-09-19 00:00:00
(1 row)
```

The following command returns the end time of a 3-second time slice:

```
SELECT TIME_SLICE('2009-09-19 00:00:01', 3, 'SECOND', 'END');
       time_slice
-----
2009-09-19 00:00:03
(1 row)
```

This command returns results in milliseconds, using a 3-second time slice:

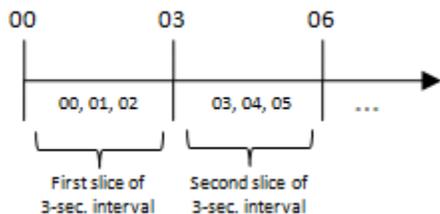
```
SELECT TIME_SLICE('2009-09-19 00:00:01', 3, 'ms');
       time_slice
-----
2009-09-19 00:00:00.999
(1 row)
```

This command returns results in microseconds, using a 9-second time slice:

```
SELECT TIME_SLICE('2009-09-19 00:00:01', 3, 'us');
       time_slice
-----
2009-09-19 00:00:00.999999
(1 row)
```

The next example uses a 3-second interval with an input value of '00:00:01'. To focus specifically on seconds, the example omits date, though all values are implied as being part of the timestamp with a given input of '00:00:01':

- '00:00:00' is the start of the 3-second time slice
- '00:00:03' is the end of the 3-second time slice.
- '00:00:03' is also the start of the *second* 3-second time slice. In time slice boundaries, the end value of a time slice does not belong to that time slice; it starts the next one.

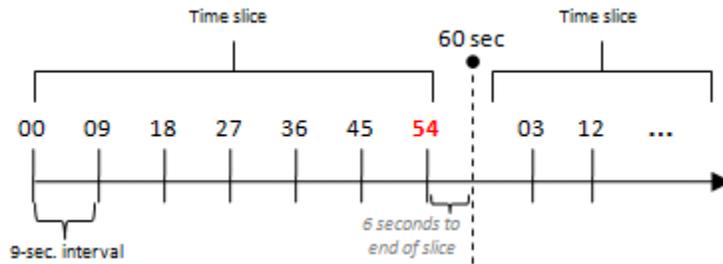


When the time slice interval is not a factor of 60 seconds, such as a given slice length of 9 in the following example, the slice does not always start or end on 00 seconds:

```
SELECT TIME_SLICE('2009-02-14 20:13:01', 9);
       time_slice
-----
2009-02-14 20:12:54
(1 row)
```

This is expected behavior, as the following properties are true for all time slices:

- Equal in length
- Consecutive (no gaps between them)
- Non-overlapping



To force the above example ('2009-02-14 20:13:01') to start at '2009-02-14 20:13:00', adjust the output timestamp values so that the remainder of 54 counts up to 60:

```
SELECT TIME_SLICE('2009-02-14 20:13:01', 9) + '6 seconds'::INTERVAL AS time;
      time
-----
2009-02-14 20:13:00
(1 row)
```

Alternatively, you could use a different slice length, which is divisible by 60, such as 5:

```
SELECT TIME_SLICE('2009-02-14 20:13:01', 5);
      time_slice
-----
2009-02-14 20:13:00
(1 row)
```

A `TIMESTAMPZ` value is implicitly cast to `TIMESTAMP`. For example, the following two statements have the same effect.

```
SELECT TIME_SLICE('2009-09-23 11:12:01'::timestampz, 3);
      TIME_SLICE
-----
2009-09-23 11:12:00
(1 row)
SELECT TIME_SLICE('2009-09-23 11:12:01'::timestampz::timestamp, 3);
      TIME_SLICE
-----
2009-09-23 11:12:00
(1 row)
```

## Examples

You can use the SQL analytic functions `FIRST_VALUE` and `LAST_VALUE` to find the first/last price within each time slice group (set of rows belonging to the same time slice). This structure could be useful if you want to sample input data by choosing one row from each time slice group.

```
SELECT date_key, transaction_time, sales_dollar_amount,
       TIME_SLICE(DATE '2000-01-01' + date_key + transaction_time, 3),
       FIRST_VALUE(sales_dollar_amount)
```

```
OVER (PARTITION BY TIME_SLICE (DATE '2000-01-01' + date_key + transaction_time, 3)
      ORDER BY DATE '2000-01-01' + date_key + transaction_time) AS first_value
FROM store.store_sales_fact
LIMIT 20;
```

date_key	transaction_time	sales_dollar_amount	time_slice	first_value
1	00:41:16	164	2000-01-02 00:41:15	164
1	00:41:33	310	2000-01-02 00:41:33	310
1	15:32:51	271	2000-01-02 15:32:51	271
1	15:33:15	419	2000-01-02 15:33:15	419
1	15:33:44	193	2000-01-02 15:33:42	193
1	16:36:29	466	2000-01-02 16:36:27	466
1	16:36:44	250	2000-01-02 16:36:42	250
2	03:11:28	39	2000-01-03 03:11:27	39
3	03:55:15	375	2000-01-04 03:55:15	375
3	11:58:05	369	2000-01-04 11:58:03	369
3	11:58:24	174	2000-01-04 11:58:24	174
3	11:58:52	449	2000-01-04 11:58:51	449
3	19:01:21	201	2000-01-04 19:01:21	201
3	22:15:05	156	2000-01-04 22:15:03	156
4	13:36:57	-125	2000-01-05 13:36:57	-125
4	13:37:24	-251	2000-01-05 13:37:24	-251
4	13:37:54	353	2000-01-05 13:37:54	353
4	13:38:04	426	2000-01-05 13:38:03	426
4	13:38:31	209	2000-01-05 13:38:30	209
5	10:21:24	488	2000-01-06 10:21:24	488

(20 rows)

Notice how `TIME_SLICE` rounds the transaction time to the 3-second slice length.

The following example returns the last trading price (the last row ordered by `TickTime`) in each 3-second time slice partition:

```
SELECT DISTINCT TIME_SLICE(TickTime, 3), LAST_VALUE(price)
OVER (PARTITION BY TIME_SLICE(TickTime, 3)
      ORDER BY TickTime ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING);
```

While the above example is the most intuitive way to express the query, Vertica does not currently support the windowing clause and `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`.

**Note:** If you omit the windowing clause from an analytic clause, `LAST_VALUE` defaults to `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`. Results can seem non-intuitive, because instead of returning the value from the bottom of the current partition, the function returns the bottom of the *window*, which continues to change along with the current input row that is being processed.

You can rewrite the query so Vertica supports it. For example, below `FIRST_VALUE` is evaluated once for each input record and the data is sorted by ascending values. Use `SELECT DISTINCT` to remove the duplicates and return only one output record per `TIME_SLICE`:

```
SELECT DISTINCT TIME_SLICE(TickTime, 3), FIRST_VALUE(price)
OVER (PARTITION BY TIME_SLICE(TickTime, 3)
      ORDER BY TickTime ASC)
FROM tick_store;
```

TIME_SLICE	?column?
2009-09-21 00:00:06	20.00
2009-09-21 00:00:09	30.00
2009-09-21 00:00:00	10.00

(3 rows)

The information output by the above query can also return MIN, MAX, and AVG of the trading prices within each time slice. Note that the following example is shown for illustration purposes only, as Vertica currently supports simple SQL aggregates only.

```
SELECT DISTINCT TIME_SLICE(TickTime, 3),
FIRST_VALUE(Price) OVER (PARTITION BY TIME_SLICE(TickTime, 3)
ORDER BY TickTime ASC),
  MIN(price) OVER (PARTITION BY TIME_SLICE(TickTime, 3)),
  MAX(price) OVER (PARTITION BY TIME_SLICE(TickTime, 3)),
  AVG(price) OVER (PARTITION BY TIME_SLICE(TickTime, 3))
FROM tick_store;
```

Rewrite query as follows:

```
SELECT fact.ts, fstvalP, minP, maxP, avgP
FROM
  (SELECT DISTINCT TIME_SLICE(TickTime, 3) ts,
  FIRST_VALUE(Price) OVER (PARTITION BY TIME_SLICE(TickTime, 3)
ORDER BY TickTime ASC) fstvalP
FROM tick_store) fact
JOIN
  (SELECT TIME_SLICE(TickTime, 3) ts,
  MIN(Price) minP, MAX(Price) maxP, AVG(Price) avgP
FROM tick_store
GROUP BY TIME_SLICE(TickTime, 3)) dim
ON fact.ts=dim.ts;
```

ts	fstvalP	minP	maxP	avgP
2009-09-21 00:00:00	10.00	10.00	11.10	10.55
2009-09-21 00:00:06	20.00	20.00	21.10	20.55
2009-09-21 00:00:09	30.00	30.00	31.10	30.55

(3 rows)

The query first sort the records within each time slice by TickTime. It next picks the subset of records with the largest TickTime value in that slice, and then it evaluates the minimum price on that subset. If no multiple records exist with the same `ts` value in the input, the output is deterministic. Otherwise, "finding the last value within each slice" is inherently nondeterministic.

**See Also**

**Aggregate Functions** (page 110)

**FIRST\_VALUE** (page 146), **LAST\_VALUE** (page 152), **TIMESERIES Clause** (page 761), **TS\_FIRST\_VALUE** (page 471), and **TS\_LAST\_VALUE** (page 472)

Using Time Series Analytics and Using SQL Analytics in the Programmer's Guide

Using Time Zones with Vertica in the Administrator's Guide

## TIMEOFDAY

Returns a text string representing the time of day.

### Behavior Type

Volatile

### Syntax

```
TIMEOFDAY ( )
```

### Notes

TIMEOFDAY ( ) returns the wall-clock time and advances during transactions.

### Examples

```
SELECT TIMEOFDAY ( ) ;
           TIMEOFDAY
-----
Thu Apr 01 15:42:04.483766 2010 EDT
(1 row)
```

## TIMESTAMPADD

Adds a specified number of intervals to a `TIMESTAMP` or `TIMESTAMPZ`. The return value depends on the input, as follows:

- If *starttimestamp* is of `TIMESTAMP`, the return value is of type `TIMESTAMP`.
- If *starttimestamp* is of `TIMESTAMPZ`, the return value is of type `TIMESTAMPZ`.

### Behavior Type

Immutable, except for `TIMESTAMPZ` arguments where it is Stable.

### Syntax 1

```
TIMESTAMPADD ( datepart , interval , starttimestamp );
```

## Parameters

<i>datepart</i>	<p>(VARCHAR) Returns the number of specified datepart boundaries between the specified startdate and enddate.</p> <p>Can be an unquoted identifier, a quoted string, or an expression in parentheses, which evaluates to the datepart as a character string.</p> <p>The following table lists the valid <i>datepart</i> arguments.</p> <table border="0"> <thead> <tr> <th><i>datepart</i>*</th> <th>abbreviation</th> </tr> <tr> <th>-----</th> <th>-----</th> </tr> </thead> <tbody> <tr> <td>YEAR</td> <td>YY, YYYY</td> </tr> <tr> <td>QUARTER</td> <td>qq, q</td> </tr> <tr> <td>MONTH</td> <td>mm, m</td> </tr> <tr> <td>DAY</td> <td>dd, d, dy, dayofyear, y</td> </tr> <tr> <td>WEEK</td> <td>wk, ww</td> </tr> <tr> <td>HOUR</td> <td>hh</td> </tr> <tr> <td>MINUTE</td> <td>mi, n</td> </tr> <tr> <td>SECOND</td> <td>ss, s</td> </tr> <tr> <td>MILLISECOND</td> <td>ms</td> </tr> <tr> <td>MICROSECOND</td> <td>mcs, us</td> </tr> </tbody> </table> <p>* Each of these dateparts can be prefixed with SQL_TSI_ (i.e. SQL_TSI_YEAR, SQL_TSI_DAY, and so forth.)</p>	<i>datepart</i> *	abbreviation	-----	-----	YEAR	YY, YYYY	QUARTER	qq, q	MONTH	mm, m	DAY	dd, d, dy, dayofyear, y	WEEK	wk, ww	HOUR	hh	MINUTE	mi, n	SECOND	ss, s	MILLISECOND	ms	MICROSECOND	mcs, us
<i>datepart</i> *	abbreviation																								
-----	-----																								
YEAR	YY, YYYY																								
QUARTER	qq, q																								
MONTH	mm, m																								
DAY	dd, d, dy, dayofyear, y																								
WEEK	wk, ww																								
HOUR	hh																								
MINUTE	mi, n																								
SECOND	ss, s																								
MILLISECOND	ms																								
MICROSECOND	mcs, us																								
<i>starttimestamp</i>	Is the start <b>TIMESTAMP</b> or <b>TIMESTAMPTZ</b> for the calculation.																								
<i>endtimestamp</i>	Is the end <b>TIMESTAMP</b> for the calculation.																								

## Notes

- **TIMESTAMPDIFF()** is an immutable function with a default type of **TIMESTAMP**. If **TIMESTAMPTZ** is specified, the function is stable.
- Vertica accepts statements written in any of the following forms:
 

```
TIMESTAMPDIFF(year, s, e);
TIMESTAMPDIFF('year', s, e);
```

If you use an expression, the expression must be enclosed in parentheses:

```
DATEDIFF((expression), s, e);
```
- Starting arguments are not included in the count, but end arguments are included.

## Example

```
=> SELECT TIMESTAMPADD (SQL_TSI_MONTH, 2, ('jan 1, 2006'));
      timestampadd
```

2006-03-01 00:00:00-05  
(1 row)

**See Also**

*Date/Time Expressions* (page 48)

**TIMESTAMPDIFF**

Returns the difference between two **TIMESTAMP** or **TIMESTAMPTZ** values, based on the specified start and end arguments.

**Behavior Type**

Immutable, except for **TIMESTAMPTZ** arguments where it is **Stable**.

**Syntax 1**

`TIMESTAMPDIFF ( datepart , starttimestamp , endtimestamp );`

**Parameters**

<i>datepart</i>	<p>(<b>VARCHAR</b>) Returns the number of specified <i>datepart</i> boundaries between the specified <i>startdate</i> and <i>enddate</i>.</p> <p>Can be an unquoted identifier, a quoted string, or an expression in parentheses, which evaluates to the <i>datepart</i> as a character string. The following table lists the valid <i>datepart</i> arguments.</p> <table style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;"><i>datepart</i></th> <th style="text-align: left;">abbreviation</th> </tr> <tr> <th style="text-align: left;">-----</th> <th style="text-align: left;">-----</th> </tr> </thead> <tbody> <tr> <td>year</td> <td>YY, YYYY</td> </tr> <tr> <td>quarter</td> <td>qq, q</td> </tr> <tr> <td>month</td> <td>mm, m</td> </tr> <tr> <td>day</td> <td>dd, d, dy, dayofyear, y</td> </tr> <tr> <td>week</td> <td>wk, ww</td> </tr> <tr> <td>hour</td> <td>hh</td> </tr> <tr> <td>minute</td> <td>mi, n</td> </tr> <tr> <td>second</td> <td>ss, s</td> </tr> <tr> <td>millisecond</td> <td>ms</td> </tr> <tr> <td>microsecond</td> <td>mcs, us</td> </tr> </tbody> </table>	<i>datepart</i>	abbreviation	-----	-----	year	YY, YYYY	quarter	qq, q	month	mm, m	day	dd, d, dy, dayofyear, y	week	wk, ww	hour	hh	minute	mi, n	second	ss, s	millisecond	ms	microsecond	mcs, us
<i>datepart</i>	abbreviation																								
-----	-----																								
year	YY, YYYY																								
quarter	qq, q																								
month	mm, m																								
day	dd, d, dy, dayofyear, y																								
week	wk, ww																								
hour	hh																								
minute	mi, n																								
second	ss, s																								
millisecond	ms																								
microsecond	mcs, us																								
<i>starttimestamp</i>	Is the start <b>TIMESTAMP</b> for the calculation.																								
<i>endtimestamp</i>	Is the end <b>TIMESTAMP</b> for the calculation.																								

## Notes

- `TIMESTAMPDIFF()` is an immutable function with a default type of `TIMESTAMP`. If `TIMESTAMPTZ` is specified, the function is stable.
- Vertica accepts statements written in any of the following forms:
 

```
TIMESTAMPDIFF(year, s, e);
TIMESTAMPDIFF('year', s, e);
```

 If you use an expression, the expression must be enclosed in parentheses:
 

```
TIMESTAMPDIFF((expression), s, e);
```
- Starting arguments are not included in the count, but end arguments are included.

## Example

```
=> SELECT TIMESTAMPDIFF ('YEAR', ('jan 1, 2006 12:34:00'), ('jan 1, 2008
12:34:00'));
 timestampdiff
-----
                2
(1 row)
```

## See Also

*Date/Time Expressions* (page 48)

## TIMESTAMP\_ROUND

Rounds a `TIMESTAMP` to a specified *format*. The return value is of type `TIMESTAMP`.

### Behavior Type

Immutable, except for `TIMESTAMPTZ` arguments where it is Stable.

### Syntax

```
TIMESTAMP_ROUND ( timestamp, format )
```

### Parameters

<code>timestamp</code>	Is the <code>TIMESTAMP</code> or <code>TIMESTAMPTZ</code> input value.												
<code>format</code>	Is a string constant that selects the precision to which truncate the input value. Valid values for <code>format</code> are: <table border="1" data-bbox="365 1501 1120 1764"> <thead> <tr> <th>Precision</th> <th>Valid values</th> </tr> </thead> <tbody> <tr> <td>Century</td> <td>CC, SCC</td> </tr> <tr> <td>Year</td> <td>SYYY, YYYY, YEAR, YYY, YY,Y</td> </tr> <tr> <td>ISO Year</td> <td>IYYY, IYY, IY, I</td> </tr> <tr> <td>Quarter</td> <td>Q</td> </tr> <tr> <td>Month</td> <td>MONTH, MON, MM, RM</td> </tr> </tbody> </table>	Precision	Valid values	Century	CC, SCC	Year	SYYY, YYYY, YEAR, YYY, YY,Y	ISO Year	IYYY, IYY, IY, I	Quarter	Q	Month	MONTH, MON, MM, RM
Precision	Valid values												
Century	CC, SCC												
Year	SYYY, YYYY, YEAR, YYY, YY,Y												
ISO Year	IYYY, IYY, IY, I												
Quarter	Q												
Month	MONTH, MON, MM, RM												

Same day of the week as the first day of the year	WW
Same day of the week as the first day of the ISO year	IW
Same day of the week as the first day of the month	W
Day	DDD, DD, J
Starting day of the week	DAY, DY, D
Hour	HH, HH12, HH24
Minute	MI
Second	SS

### Examples

```
b=> SELECT TIMESTAMP_ROUND('sep 22, 2011 12:34:00', 'dy');
       TIMESTAMP_ROUND
-----
2011-09-18 00:00:00
(1 row)
```

## TIMESTAMP\_TRUNC

Truncates a `TIMESTAMP`. The return value is of type `TIMESTAMP`.

### Behavior Type

Immutable, except for `TIMESTAMPTZ` arguments where it is Stable.

### Syntax

```
TIMESTAMP_TRUNC ( timestamp, format )
```

### Parameters

<code>timestamp</code>	Is the <code>TIMESTAMP</code> or <code>TIMESTAMPTZ</code> input value.								
<code>format</code>	Is a string constant that selects the precision to which truncate the input value. Valid values for <code>format</code> are: <table border="1" data-bbox="365 1669 1123 1841"> <thead> <tr> <th>Precision</th> <th>Valid values</th> </tr> </thead> <tbody> <tr> <td>Century</td> <td>CC, SCC</td> </tr> <tr> <td>Year</td> <td>SYYY, YYYY, YEAR, YYY, YY,Y</td> </tr> <tr> <td>ISO Year</td> <td>IYYY, IYY, IY, I</td> </tr> </tbody> </table>	Precision	Valid values	Century	CC, SCC	Year	SYYY, YYYY, YEAR, YYY, YY,Y	ISO Year	IYYY, IYY, IY, I
Precision	Valid values								
Century	CC, SCC								
Year	SYYY, YYYY, YEAR, YYY, YY,Y								
ISO Year	IYYY, IYY, IY, I								

Quarter	Q
Month	MONTH, MON, MM, RM
Same day of the week as the first day of the year	WW
Same day of the week as the first day of the ISO year	IW
Same day of the week as the first day of the month	W
Day	DDD, DD, J
Starting day of the week	DAY, DY, D
Hour	HH, HH12, HH24
Minute	MI
Second	SS

## Examples

```
=> SELECT TIMESTAMP_TRUNC('sep 22, 2011 12:34:00');
TIMESTAMP_TRUNC
-----
2011-09-22 00:00:00
(1 row)
=> SELECT TIMESTAMP_TRUNC('sep 22, 2011 12:34:00', 'dy');
TIMESTAMP_TRUNC
-----
2011-09-18 00:00:00
(1 row)
```

## TRANSACTION\_TIMESTAMP

Returns a value of type `TIMESTAMP WITH TIME ZONE` representing the start of the current transaction. `TRANSACTION_TIMESTAMP` is equivalent to ***CURRENT\_TIMESTAMP*** (page 191) except that it does not accept a precision parameter.

### Behavior Type

Stable

### Syntax

```
TRANSACTION_TIMESTAMP()
```

## Notes

This function returns the start time of the current transaction; the value does not change during the transaction. The intent is to allow a single transaction to have a consistent notion of the "current" time, so that multiple modifications within the same transaction bear the same timestamp.

## Examples

```
SELECT TRANSACTION_TIMESTAMP();
      TRANSACTION_TIMESTAMP
-----
2010-04-01 15:31:12.144584-04
(1 row)
```

## See Also

**CLOCK\_TIMESTAMP** (page 189) and **STATEMENT\_TIMESTAMP** (page 223)

## TRUNC [Date/Time]

Truncates a **TIMESTAMP**, **TIMESTAMPTZ**, or **DATE**. The return value is of type **TIMESTAMP**.

## Behavior Type

Immutable, except for **TIMESTAMPTZ** arguments where it is Stable.

## Syntax

```
TRUNC ([TIMESTAMP | DATE] , format )
```

## Parameters

<b>TIMESTAMP   DATE</b>	Is the <b>TIMESTAMP</b> or <b>DATE</b> input value.																
<i>format</i>	Is a string constant that selects the precision to which truncate the input value. Valid values for <i>format</i> are: <table border="1" data-bbox="446 1302 1122 1814"> <thead> <tr> <th>Precision</th> <th>Valid values</th> </tr> </thead> <tbody> <tr> <td>Century</td> <td>CC, SCC</td> </tr> <tr> <td>Year</td> <td>SYYY, YYYY, YEAR, YYY, YY,Y</td> </tr> <tr> <td>ISO Year</td> <td>IYYY, IYY, IY, I</td> </tr> <tr> <td>Quarter</td> <td>Q</td> </tr> <tr> <td>Month</td> <td>MONTH, MON, MM, RM</td> </tr> <tr> <td>Same day of the week as the first day of the year</td> <td>WW</td> </tr> <tr> <td>Same day of the week as the first day of the ISO year</td> <td>IW</td> </tr> </tbody> </table>	Precision	Valid values	Century	CC, SCC	Year	SYYY, YYYY, YEAR, YYY, YY,Y	ISO Year	IYYY, IYY, IY, I	Quarter	Q	Month	MONTH, MON, MM, RM	Same day of the week as the first day of the year	WW	Same day of the week as the first day of the ISO year	IW
Precision	Valid values																
Century	CC, SCC																
Year	SYYY, YYYY, YEAR, YYY, YY,Y																
ISO Year	IYYY, IYY, IY, I																
Quarter	Q																
Month	MONTH, MON, MM, RM																
Same day of the week as the first day of the year	WW																
Same day of the week as the first day of the ISO year	IW																

Same day of the week as the first day of the month	W
Day	DDD, DD, J
Starting day of the week	DAY, DY, D
Hour	HH, HH12, HH24
Minute	MI
Second	SS

## Examples

```
=> SELECT TRUNC(TIMESTAMP 'sep 22, 2011 12:34:00', 'dy');
      TRUNC
-----
2011-09-18 00:00:00
(1 row)
```

## WEEK

Returns an INTEGER representing the week of the year into which the input value falls. A week starts on Sunday. January 1 is always in the first week of the year.

The input is of type VARCHAR, DATE, TIMESTAMP, or TIMESTAMPTZ.

### Syntax

```
WEEK ( d )
```

### Behavior type

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

### Parameters

<i>d</i>	Is the VARCHAR, DATE, TIMESTAMP, TIMESTAMPTZ input value.
----------	---

### Example

```
=> SELECT WEEK (TIMESTAMP 'sep 22, 2011 12:34');
      WEEK
-----
      39
(1 row)
```

## WEEK\_ISO

Returns an INTEGER from 1 - 53 representing the week of the year into which the input value falls. The return value is based on the ISO 8061 standard. The input is of VARCHAR, DATE, TIMESTAMP, or TIMESTAMPTZ.

The ISO week consists of 7 days starting on Monday and ending on Sunday. The first week of the year is the week that contains January 4.

### Syntax

```
WEEK_ISO ( d )
```

### Behavior type

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

### Parameters

<i>d</i>	Is the VARCHAR, DATE, TIMESTAMP, or TIMESTAMPTZ input value.
----------	--

### Example

The following examples illustrate the different results returned by WEEK\_ISO. The first shows that December 28, 2011 falls within week 52 of the ISO calendar:

```
=> SELECT WEEK_ISO (TIMESTAMP 'Dec 28, 2011 10:00:00');
WEEK_ISO
-----
        52
(1 row)
```

The second example shows WEEK\_ISO results for January 1, 2012. Note that, since this date falls on a Sunday, it falls within week 52 of the ISO year:

```
=> SELECT WEEK_ISO (TIMESTAMP 'Jan 1, 2012 10:00:00');
WEEK_ISO
-----
        52
(1 row)
```

The third example shows WEEK\_ISO results for January 2, 2012, which occurs on a Monday. This is the first week of the year that contains a Thursday and contains January 4. The function returns week 1.

```
=> SELECT WEEK_ISO (TIMESTAMP 'Jan 2, 2012 10:00:00');
WEEK_ISO
-----
         1
```

The last example shows how to combine the DAYOFWEEK\_ISO, WEEK\_ISO, and YEAR\_ISO functions to find the ISO day of the week, week, and year:

```
=> SELECT DAYOFWEEK_ISO('Jan 1, 2000'), WEEK_ISO('Jan 1,
2000'), YEAR_ISO('Jan1,2000');
 DAYOFWEEK_ISO | WEEK_ISO | YEAR_ISO
-----+-----+-----
                6 |         52 |    1999
(1 row)
```

**See Also****YEAR\_ISO** (page 240)**DAYOFWEEK\_ISO** (page 204)[http://en.wikipedia.org/wiki/ISO\\_8601](http://en.wikipedia.org/wiki/ISO_8601) ([http://en.wikipedia.org/wiki/ISO\\_8601](http://en.wikipedia.org/wiki/ISO_8601))**YEAR**

Returns an INTEGER representing the year portion of the input value. The input value can be of type VARCHAR, TIMESTAMP, TIMESTAMPTZ, or INTERVAL.

**Syntax**YEAR( *d* )**Behavior type**

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

**Parameters**

<i>d</i>	Is the VARCHAR, TIMESTAMP, TIMESTAMPTZ, or INTERVAL input value.
----------	--

**Example**

```
=> SELECT YEAR ('6-9');
 YEAR
-----
    6
(1 row)
=> SELECT YEAR (TIMESTAMP 'sep 22, 2011 12:34');
 YEAR
-----
 2011
(1 row)

=> SELECT YEAR (INTERVAL '2-35' year to month);
 YEAR
-----
    4
(1 row)
```

## YEAR\_ISO

Returns an INTEGER representing the year portion of the input value. The return value is based on the ISO 8061 standard. The input value can be of type VARCHAR, DATE, TIMESTAMP, or TIMESTAMPTZ.

The first week of the ISO year is the week that contains January 4.

### Syntax

```
YEAR_ISO( d )
```

### Behavior type

Immutable, except for TIMESTAMPTZ arguments where it is Stable.

### Parameters

<i>d</i>	Is the VARCHAR, DATE, TIMESTAMP, or TIMESTAMPTZ input value.
----------	--

### Example

```
=> SELECT YEAR_ISO (TIMESTAMP 'sep 22, 2011 12:34');
YEAR_ISO
-----
      2011
(1 row)
```

The following example shows how to combine the DAYOFWEEK\_ISO, WEEK\_ISO, and YEAR\_ISO functions to find the ISO day of the week, week, and year:

```
=> SELECT DAYOFWEEK_ISO('Jan 1, 2000'), WEEK_ISO('Jan 1,
2000'), YEAR_ISO('Jan1,2000');
DAYOFWEEK_ISO | WEEK_ISO | YEAR_ISO
-----+-----+-----
              6 |       52 |    1999
(1 row)
```

### See also

**WEEK\_ISO** (page 238)

**DAYOFWEEK\_ISO** (page 204)

[http://en.wikipedia.org/wiki/ISO\\_8601](http://en.wikipedia.org/wiki/ISO_8601) ([http://en.wikipedia.org/wiki/ISO\\_8601](http://en.wikipedia.org/wiki/ISO_8601))

## Formatting Functions

Formatting functions provide a powerful tool set for converting various data types (DATE/TIME, INTEGER, FLOATING POINT) to formatted strings and for converting from formatted strings to specific data types.

These functions all follow a common calling convention:

- The first argument is the value to be formatted.
- The second argument is a template that defines the output or input format.

**Exception:** The `TO_TIMESTAMP` function can take a single double precision argument.

### TO\_BITSTRING

Returns a VARCHAR that represents the given VARBINARY value in bitstring format

#### Behavior Type

Immutable

#### Syntax

```
TO_BITSTRING ( expression )
```

#### Parameters

<i>expression</i>	(VARCHAR) is the string to return.
-------------------	------------------------------------

#### Notes

`VARCHAR TO_BITSTRING(VARBINARY)` converts data from binary type to character type (where the character representation is the bitstring format). This function is the inverse of `BITSTRING_TO_BINARY`:

```
TO_BITSTRING(BITSTRING_TO_BINARY(x)) = x
BITSTRING_TO_BINARY(TO_BITSTRING(x)) = x
```

#### Examples

```
SELECT TO_BITSTRING('ab'::BINARY(2));
   to_bitstring
-----
0110000101100010
(1 row)
SELECT TO_BITSTRING(HEX_TO_BINARY('0x10'));
   to_bitstring
-----
00010000
(1 row)
```

```
SELECT TO_BITSTRING(HEX_TO_BINARY('0xF0'));
to_bitstring
-----
11110000
(1 row)
```

**See Also**

**BITCOUNT** (page 316) and **BITSTRING\_TO\_BINARY** (page 316)

**TO\_CHAR**

Converts various date/time and numeric values into text strings.

**Behavior Type**

Stable

**Syntax**

```
TO_CHAR ( expression [, pattern ] )
```

**Parameters**

<i>expression</i>	(TIMESTAMP, INTERVAL, INTEGER, DOUBLE PRECISION) specifies the value to convert.
<i>pattern</i>	[Optional] (CHAR or VARCHAR) specifies an output pattern string using the <b>Template Patterns for Date/Time Formatting</b> (page 248) and and/or <b>Template Patterns for Numeric Formatting</b> (page 250).

**Notes**

- TO\_CHAR(any) casts any type, except BINARY/VARBINARY, to VARCHAR.  
The following example returns an error if you attempt to cast TO\_CHAR to a binary data type:  
=> SELECT TO\_CHAR('abc'::VARBINARY);  
ERROR: cannot cast type varbinary to varchar
- Ordinary text is allowed in to\_char templates and is output literally. You can put a substring in double quotes to force it to be interpreted as literal text even if it contains pattern key words. For example, in '"Hello Year "YYYY"', the YYYY is replaced by the year data, but the single Y in Year is not.
- The TO\_CHAR function's day-of-the-week numbering (see the 'D' **template pattern** (page 248)) is different from that of the **EXTRACT** (page 207) function.
- Given an INTERVAL type, TO\_CHAR formats HH and HH12 as hours in a single day, while HH24 can output hours exceeding a single day, for example, >24.
- To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: '\\\\"YYYY Month\\"'
- TO\_CHAR does not support the use of V combined with a decimal point. For example: 99.9V99 is not allowed.

## Examples

Expression	Result
SELECT TO_CHAR(CURRENT_TIMESTAMP, 'Day, DD HH12:MI:SS');	'Tuesday , 06 05:39:18'
SELECT TO_CHAR(CURRENT_TIMESTAMP, 'FMDay, FMDD HH12:MI:SS');	'Tuesday, 6 05:39:18'
SELECT TO_CHAR(-0.1, '99.99');	' -.10'
SELECT TO_CHAR(-0.1, 'FM9.99');	'-.1'
SELECT TO_CHAR(0.1, '0.9');	' 0.1'
SELECT TO_CHAR(12, '9990999.9');	' 0012.0'
SELECT TO_CHAR(12, 'FM9990999.9');	'0012.'
SELECT TO_CHAR(485, '999');	' 485'
SELECT TO_CHAR(-485, '999');	'-485'
SELECT TO_CHAR(485, '9 9 9');	' 4 8 5'
SELECT TO_CHAR(1485, '9,999');	' 1,485'
SELECT TO_CHAR(1485, '9G999');	' 1 485'
SELECT TO_CHAR(148.5, '999.999');	' 148.500'
SELECT TO_CHAR(148.5, 'FM999.999');	'148.5'
SELECT TO_CHAR(148.5, 'FM999.990');	'148.500'
SELECT TO_CHAR(148.5, '999D999');	' 148,500'
SELECT TO_CHAR(3148.5, '9G999D999');	' 3 148,500'
SELECT TO_CHAR(-485, '999S');	'485-'
SELECT TO_CHAR(-485, '999MI');	'485-'
SELECT TO_CHAR(485, '999MI');	'485 '
SELECT TO_CHAR(485, 'FM999MI');	'485'
SELECT TO_CHAR(485, 'PL999');	'+485'
SELECT TO_CHAR(485, 'SG999');	'+485'
SELECT TO_CHAR(-485, 'SG999');	'-485'
SELECT TO_CHAR(-485, '9SG99');	'4-85'
SELECT TO_CHAR(-485, '999PR');	'<485>'
SELECT TO_CHAR(485, 'L999');	'DM 485'
SELECT TO_CHAR(485, 'RN');	' CDLXXXV'
SELECT TO_CHAR(485, 'FMRN');	'CDLXXXV'
SELECT TO_CHAR(5.2, 'FMRN');	'V'
SELECT TO_CHAR(482, '999th');	' 482nd'
SELECT TO_CHAR(485, '"Good number:"999');	'Good number: 485'
SELECT TO_CHAR(485.8, '"Pre:"999" Post:" .999');	'Pre: 485 Post: .800'
SELECT TO_CHAR(12, '99V999');	' 12000'
SELECT TO_CHAR(12.4, '99V999');	' 12400'
SELECT TO_CHAR(12.45, '99V9');	' 125'
SELECT TO_CHAR(-1234.567);	-1234.567
SELECT TO_CHAR('1999-12-25'::DATE);	1999-12-25
SELECT TO_CHAR('1999-12-25 11:31'::TIMESTAMP);	1999-12-25 11:31:00
SELECT TO_CHAR('1999-12-25 11:31 EST'::TIMESTAMPTZ);	1999-12-25 11:31:00-05
SELECT TO_CHAR('3 days 1000.333 secs'::INTERVAL);	3 days 00:16:40.333

## TO\_DATE

Converts a string value to a DATE type.

## Behavior Type

Stable

## Syntax

```
TO_DATE ( expression , pattern )
```

## Parameters

<i>expression</i>	(CHAR or VARCHAR) specifies the value to convert
<i>pattern</i>	(CHAR or VARCHAR) specifies an output pattern string using the <i>Template Patterns for Date/Time Formatting</i> (page 248) and/or <i>Template Patterns for Numeric Formatting</i> (page 250).

## Notes

- To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: `'\\"YYYY Month\\"'`
- TO\_TIMESTAMP and TO\_DATE skip multiple blank spaces in the input string if the FX option is not used. FX must be specified as the first item in the template. For example:
  - For example `TO_TIMESTAMP('2000 JUN', 'YYYY MON')` is correct.
  - `TO_TIMESTAMP('2000 JUN', 'FXYYYY MON')` returns an error, because TO\_TIMESTAMP expects one space only.
- The YYYY conversion from string to TIMESTAMP or DATE has a restriction if you use a year with more than four digits. You must use a non-digit character or template after YYYY, otherwise the year is always interpreted as four digits. For example (with the year 20000):  
`TO_DATE('200001131', 'YYYYMMDD')` is interpreted as a four-digit year  
 Instead, use a non-digit separator after the year, such as `TO_DATE('20000-1131', 'YYYY-MMDD')` or `TO_DATE('20000Nov31', 'YYYYMonDD')`.
- In conversions from string to TIMESTAMP or DATE, the CC field is ignored if there is a YYY, YYYY or Y,YYY field. If CC is used with YY or Y then the year is computed as  $(CC-1)*100+YY$ .

## Examples

```
SELECT TO_DATE('13 Feb 2000', 'DD Mon YYYY');
       to_date
-----
2000-02-13
(1 row)
```

## See Also

*Template Pattern Modifiers for Date/Time Formatting* (page 249)

## TO\_HEX

Returns a VARCHAR or VARBINARY representing the hexadecimal equivalent of a number.

## Behavior Type

Immutable

## Syntax

```
TO_HEX ( number )
```

## Parameters

<i>number</i>	(INTEGER) is the number to convert to hexadecimal
---------------	---

## Notes

VARCHAR TO\_HEX(INTEGER) and VARCHAR TO\_HEX(VARBINARY) are similar. The function converts data from binary type to character type (where the character representation is in hexadecimal format). This function is the inverse of HEX\_TO\_BINARY.

```
TO_HEX(HEX_TO_BINARY(x)) = x).
HEX_TO_BINARY(TO_HEX(x)) = x).
```

## Examples

```
SELECT TO_HEX(123456789);
   to_hex
-----
   75bcd15
(1 row)
```

For VARBINARY inputs, the returned value is not preceded by "0x". For example:

```
SELECT TO_HEX('ab'::binary(2));
   to_hex
-----
   6162
(1 row)
```

## TO\_TIMESTAMP

Converts a string value or a UNIX/POSIX epoch value to a `TIMESTAMP WITH TIME ZONE` type.

## Behavior Type

Immutable if single argument form, Stable otherwise.

## Syntax

```
TO_TIMESTAMP ( expression, pattern )
TO_TIMESTAMP ( unix-epoch )
```

## Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to convert
<i>pattern</i>	(CHAR or VARCHAR) specifies an output pattern string using the <b>Template Patterns for Date/Time Formatting</b> (page 248) and/or

	<b>Template Patterns for Numeric Formatting</b> (page 250).
<code>unix-epoch</code>	(DOUBLE PRECISION) specifies some number of seconds elapsed since midnight UTC of January 1, 1970, not counting leap seconds. INTEGER values are implicitly cast to DOUBLE PRECISION.

## Notes

- For more information about UNIX/POSIX time, see *Wikipedia* [http://en.wikipedia.org/wiki/Unix\\_time](http://en.wikipedia.org/wiki/Unix_time).
- Millisecond (MS) and microsecond (US) values in a conversion from string to `TIMESTAMP` are used as part of the seconds after the decimal point. For example `TO_TIMESTAMP('12:3', 'SS:MS')` is not 3 milliseconds, but 300, because the conversion counts it as 12 + 0.3 seconds. This means for the format `SS:MS`, the input values 12:3, 12:30, and 12:300 specify the same number of milliseconds. To get three milliseconds, use 12:003, which the conversion counts as 12 + 0.003 = 12.003 seconds.

Here is a more complex example: `TO_TIMESTAMP('15:12:02.020.001230', 'HH:MI:SS.MS.US')` is 15 hours, 12 minutes, and 2 seconds + 20 milliseconds + 1230 microseconds = 2.021230 seconds.

- To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: `'\\"YYYY Month\\"'`
- `TZ/tz` are not supported patterns for the `TO_TIMESTAMP` function; for example, the following command returns an error:

```
SELECT TO_TIMESTAMP('01-01-01 01:01:01+03:00', 'DD-MM-YY
HH24:MI:SStz');
ERROR: "TZ"/"tz" not supported
```
- `TO_TIMESTAMP` and `TO_DATE` skip multiple blank spaces in the input string if the `FX` option is not used. `FX` must be specified as the first item in the template. For example:
  - For example `TO_TIMESTAMP('2000 JUN', 'YYYY MON')` is correct.
  - `TO_TIMESTAMP('2000 JUN', 'FXYYYY MON')` returns an error, because `TO_TIMESTAMP` expects one space only.
- The `YYYY` conversion from string to `TIMESTAMP` or `DATE` has a restriction if you use a year with more than four digits. You must use a non-digit character or template after `YYYY`, otherwise the year is always interpreted as four digits. For example (with the year 20000):

```
TO_DATE('200001131', 'YYYYMMDD')
```

is interpreted as a four-digit year

Instead, use a non-digit separator after the year, such as `TO_DATE('20000-1131', 'YYYY-MMDD')` or `TO_DATE('20000Nov31', 'YYYYMonDD')`.
- In conversions from string to `TIMESTAMP` or `DATE`, the `CC` field is ignored if there is a `YYY`, `YYYY` or `Y,YYY` field. If `CC` is used with `YY` or `Y` then the year is computed as  $(CC-1)*100+YY$ .

## Examples

```
=> SELECT TO_TIMESTAMP('13 Feb 2009', 'DD Mon YYYY');
       to_timestamp
-----
2009-02-13 00:00:00-05
```

```
(1 row)
=> SELECT TO_TIMESTAMP(200120400);
       to_timestamp
-----
1976-05-05 01:00:00-04
(1 row)
```

**See Also**

**Template Pattern Modifiers for Date/Time Formatting** (page 249)

**TO\_NUMBER**

Converts a string value to DOUBLE PRECISION.

**Behavior Type**

Stable

**Syntax**

```
TO_NUMBER ( expression, [ pattern ] )
```

**Parameters**

<i>expression</i>	(CHAR or VARCHAR) specifies the string to convert.
<i>pattern</i>	(CHAR or VARCHAR) Optional parameter specifies an output pattern string using the <b>Template Patterns for Date/Time Formatting</b> (page 248) and/or <b>Template Patterns for Numeric Formatting</b> (page 250). If omitted, function returns a floating point.

**Notes**

To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: '\\ "YYYY Month\\ "'

**Examples**

```
SELECT TO_CHAR(2009, 'rn'), TO_NUMBER('mmix', 'rn');
       to_char | to_number
-----+-----
          mmix |          2009
(1 row)
```

It the `pattern` parameter is omitted, the function returns a floating point.

```
SELECT TO_NUMBER('-123.456e-01');
       to_number
-----
-12.3456
```

## Template Patterns for Date/Time Formatting

In an output template string (for `TO_CHAR`), there are certain patterns that are recognized and replaced with appropriately-formatted data from the value to be formatted. Any text that is not a template pattern is copied verbatim. Similarly, in an input template string (for anything other than `TO_CHAR`), template patterns identify the parts of the input data string to be looked at and the values to be found there.

**Note:** Vertica uses the ISO 8601:2004 style for date/time fields in Vertica \*.log files. For example,

```
2008-09-16 14:40:59.123 TM Moveout:0x2aaaac002180 [Txn] <INFO>
```

Certain modifiers can be applied to any template pattern to alter its behavior as described in *Template Pattern Modifiers for Date/Time Formatting* (page 249).

Pattern	Description
HH	Hour of day (00-23)
HH12	Hour of day (01-12)
HH24	Hour of day (00-23)
MI	Minute (00-59)
SS	Second (00-59)
MS	Millisecond (000-999)
US	Microsecond (000000-999999)
SSSS	Seconds past midnight (0-86399)
AM or A.M. or PM or P.M.	Meridian indicator (uppercase)
am or a.m. or pm or p.m.	Meridian indicator (lowercase)
Y,YYY	Year (4 and more digits) with comma
YYYY	Year (4 and more digits)
YYY	Last 3 digits of year
YY	Last 2 digits of year
Y	Last digit of year
IYYY	ISO year (4 and more digits)
IYY	Last 3 digits of ISO year
IY	Last 2 digits of ISO year
I	Last digits of ISO year
BC or B.C. or AD or A.D.	Era indicator (uppercase)
bc or b.c. or ad or a.d.	Era indicator (lowercase)
MONTH	Full uppercase month name (blank-padded to 9 chars)

Month	Full mixed-case month name (blank-padded to 9 chars)
month	Full lowercase month name (blank-padded to 9 chars)
MON	Abbreviated uppercase month name (3 chars)
Mon	Abbreviated mixed-case month name (3 chars)
mon	Abbreviated lowercase month name (3 chars)
MM	Month number (01-12)
DAY	Full uppercase day name (blank-padded to 9 chars)
Day	Full mixed-case day name (blank-padded to 9 chars)
day	full lowercase day name (blank-padded to 9 chars)
DY	Abbreviated uppercase day name (3 chars)
Dy	Abbreviated mixed-case day name (3 chars)
dy	Abbreviated lowercase day name (3 chars)
DDD	Day of year (001-366)
DD	Day of month (01-31) for TIMESTAMP <b>Note:</b> For INTERVAL, DD is day of year (001-366) because day of month is undefined.
D	Day of week (1-7; Sunday is 1)
W	Week of month (1-5) (The first week starts on the first day of the month.)
WW	Week number of year (1-53) (The first week starts on the first day of the year.)
IW	ISO week number of year (The first Thursday of the new year is in week 1.)
CC	Century (2 digits)
J	Julian Day (days since January 1, 4712 BC)
Q	Quarter
RM	Month in Roman numerals (I-XII; I=January) (uppercase)
rm	Month in Roman numerals (i-xii; i=January) (lowercase)
TZ	Time-zone name (uppercase)
tz	Time-zone name (lowercase)

### Template Pattern Modifiers for Date/Time Formatting

Certain modifiers can be applied to any template pattern to alter its behavior. For example, `FMMonth` is the `Month` pattern with the `FM` modifier.

Modifier	Description
AM	Time is before 12:00
AT	Ignored
JULIAN, JD, J	Next field is Julian Day
FM prefix	Fill mode (suppress padding blanks and zeros) For example: FMMonth <b>Note:</b> The FM modifier suppresses leading zeros and trailing blanks that would otherwise be added to make the output of a pattern fixed width.
FX prefix	Fixed format global option For example: FX Month DD Day
ON	Ignored
PM	Time is on or after 12:00
T	Next field is time
TH suffix	Uppercase ordinal number suffix For example: DDTH
th suffix	Lowercase ordinal number suffix For example: DDth
TM prefix	Translation mode (print localized day and month names based on lc_messages). For example: TMMonth

## Template Patterns for Numeric Formatting

Pattern	Description
9	Value with the specified number of digits
0	Value with leading zeros
. (period)	Decimal point
, (comma)	Group (thousand) separator
PR	Negative value in angle brackets
S	Sign anchored to number (uses locale)
L	Currency symbol (uses locale)
D	Decimal point (uses locale)
G	Group separator (uses locale)
MI	Minus sign in specified position (if number < 0)

PL	Plus sign in specified position (if number > 0)
SG	Plus/minus sign in specified position
RN	Roman numeral (input between 1 and 3999)
TH or th	Ordinal number suffix
V	Shift specified number of digits (see notes)
EEEE	Scientific notation (not implemented yet)

### Usage

- A sign formatted using SG, PL, or MI is not anchored to the number; for example:
  - `TO_CHAR(-12, 'S9999')` produces ' -12'
  - `TO_CHAR(-12, 'MI9999')` produces '- 12'
- 9 results in a value with the same number of digits as there are 9s. If a digit is not available it outputs a space.
- TH does not convert values less than zero and does not convert fractional numbers.
- V effectively multiplies the input values by  $10^n$ , where  $n$  is the number of digits following V. `TO_CHAR` does not support the use of V combined with a decimal point. For example: `99.9V99` is not allowed.

## Geospatial Package SQL Functions

The Vertica Geospatial package contains a suite of geospatial SQL functions you can install to report on and analyze geographic location data.

### Contents of the Geospatial package

When you installed Vertica, the RPM placed the Geospatial package files here:

```
/opt/vertica/packages/geospatial
```

This directory contains these files:

<code>install.sh</code>	Installs the Geospatial package.
<code>readme.txt</code>	Contains instructions for installing the package.

This directory also contains these directories:

<code>/src</code>	<p>Contains this files</p> <ul style="list-style-type: none"> <li>▪ <code>geospatial.sql</code>—This file contains all of the functions that will be installed with the package. The file describes the calculations used for each, and provides examples. This file also contains links to helpful sites you can visit for more information about standards and calculations.</li> </ul>
<code>/examples</code>	Contains this file:

- |  |  |
|--|--|
|  | <ul style="list-style-type: none"><li>▪ <code>regions_demo.sql</code>--This file is a demo, intended to illustrate a simple use case: determine the New England state in which a given point lies.</li></ul> |
|--|--|

## Using Geospatial Package SQL Functions

For high-level descriptions of all of the functions included in the package, see **Geospatial SQL Functions** (page 252). For more detailed information about each function and for links to other useful information, see `/opt/vertica/packages/geospatial/src/geospatial.sql`.

## Using built-in Vertica functions for Geospatial analysis

Four mathematical functions, automatically installed with Vertica, let you perform geospatial operations:

- **DEGREES** (page 283)
- **DISTANCE** (page 284)
- **DISTANCEV** (page 285)
- **RADIANS** (page 292)

These functions are not part of the Geospatial package, but are built-in with Vertica.

## Geospatial SQL Functions

With the Geospatial package, Vertica provides SQL functions that let you find geographic constants you can use in your calculations and analysis. These functions appear in the file `/opt/vertica/packages/geospatial/src/geospatial.sql`.

You can use these functions as they are supplied; you can also edit the `geospatial.sql` file to change the calculations according to your needs. If you do modify these, be sure to save a copy of your changes in a private location so that your changes are not lost if you upgrade your Vertica installation. Note that an upgrade will not overwrite any functions already loaded in your database; only the `.sql` file containing the function definitions will be overwritten.

These functions measure distances in kilometers and angles in fractional degrees, unless stated otherwise.

Of the several possible definitions of latitude, the geodetic latitude is most commonly used, and this is what is used in the Vertica Geospatial package. Latitude goes from +90 degrees at the North Pole to -90 at the South Pole. Longitude 0 is near Greenwich, England. It increases going east to +180 degrees, and decreases going west to -180 degrees. True bearings are measured clockwise from north. For more information, see: <http://en.wikipedia.org/wiki/Latitude> (<http://en.wikipedia.org/wiki/Latitude>)

## WGS-84 SQL Functions

The following functions return constants determined by the World Geodetic System (WGS) standard, WGS-84.

- **WGS84\_a()** (page 269)
- **WGS84\_b()** (page 269)

- **WGS84\_e2()** (page 270)
- **WGS84\_f()** (page 270)
- **WGS84\_if()** (page 271)

### Earth Radius, Radius of Curvature, and Bearing SQL Functions

These functions return the earth's radius, radius of curvature, and bearing values.

- **RADIUS\_r(lat)** (page 265)
- **WGS84\_r1()** (page 271)
- **RADIUS\_SI()** (page 268)
- **RADIUS\_M(lat)** (page 264)
- **RADIUS\_N(lat)** (page 265)
- **RADIUS\_Ra(lat)**
- **RADIUS\_Rv(lat)** (page 267)
- **RADIUS\_Rc(lat, bearing)** (page 266)
- **BEARING (lat1,lon1,lat2,lon2)** (page 255)
- **RADIUS\_LON(lat)** (page 264)

### ECEF Conversion SQL Functions

The following functions convert values to Earth-Centered, Earth Fixed (ECEF) values. The ECEF system represents positions on X, Y, and Z axes in meters. (0,0,0) is the center of the earth; X is toward latitude 0, longitude 0; Y is toward latitude 0, longitude 90 degrees; Z is toward the North Pole. The height above mean sea level (h) is also in meters.

- **ECEF\_x(lat,lon,h)** (page 257)
- **ECEF\_y(lat,lon,h)** (page 258)
- **ECEF\_z(lat,lon,h)** (page 258)
- **ECEF\_chord(lat1,lon1,h1,lat2,lon2,h2)** (page 257)
- **CHORD\_TO\_ARC(chord)** (page 255)

### Bounding Box SQL Functions

These functions determine whether points are within a bounding box, a rectangular area whose edges are latitude and longitude lines. Bounding box methods allow you to narrow your focus, and work best on Vertica projections that are sorted by latitude, or by region (such as State) and then by latitude. These methods also work on projections sorted by longitude.

- **BB\_WITHIN (lat,lon, llat, llon, ulat, rlon)** (page 254)
- **LAT\_WITHIN (lat, lat0, d)** (page 260)
- **LON\_WITHIN (lon, lat0, lon0, d)** (page 262)
- **LL\_WITHIN (lat,lon, lat0, lon0, d)** (page 261)
- **DWITHIN (lat,lon,lat0,lon0, d)** (page 256)
- **LLD\_WITHINn(lat,lon,lat0,lon0, d)** (page 262)
- **ISLEFT (x0,y0, x1,y1,x2,y2)** (page 259)
- **RAYCROSSING (x0,y0, x1,y1, x2,y2)** (page 268)

## Miles/Kilometer Conversion SQL Functions

These functions convert miles to kilometers and kilometers to miles.

- **MILES2KM** (*miles*) (page 263)
- **KM2MILES** (*km*) (page 260)

## BB\_WITHIN

Determines whether a point (*lat*, *lon*) falls within a bounding box defined by its lower-left and upper-right corners. The return value has the type **BOOLEAN**.

This function is available only if you install the Vertica Geospatial Package package. See **Geospatial Package SQL Functions** (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
BB_WITHIN (lat,lon, llat, llon, ulat, rlon)
```

### Parameters

<i>lat</i>	A value of type <b>DOUBLE PRECISION</b> indicating the latitude of a given point.
<i>lon</i>	A value of type <b>DOUBLE PRECISION</b> indicating the longitude of a given point.
<i>llat</i>	A value of type <b>DOUBLE PRECISION</b> indicating the latitude used in defining the lower left corner of the bounding box.
<i>llon</i>	A value of type <b>DOUBLE PRECISION</b> indicating the longitude used in defining the lower left corner of the bounding box.
<i>ulat</i>	A value of type <b>DOUBLE PRECISION</b> indicating the latitude used in defining the upper right corner of the bounding box.
<i>rlon</i>	A value of type <b>DOUBLE PRECISION</b> indicating the longitude used in defining the upper right corner of the bounding box.

### Example

```
=> SELECT BB_WITHIN(14, 30, 23.0, 45, 13, 37);  
BB_WITHIN  
-----  
f  
(1 row)
```

## BEARING

Returns the approximate bearing from a starting point to an ending point, in degrees. It assumes a flat earth and is useful only for short distances. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
BEARING (lat1, lon1, lat2, lon2)
```

### Parameters

<i>lat1</i>	A value of type DOUBLE PRECISION indicating latitude of the starting point.
<i>lon1</i>	A value of type DOUBLE PRECISION indicating longitude of the starting point.
<i>lat2</i>	A value of type DOUBLE PRECISION indicating latitude of the ending point.
<i>lon2</i>	A value of type DOUBLE PRECISION indicating longitude of the ending point.

### Example

```
=>SELECT BEARING (90, 0, 0, 0) ;
    BEARING
-----
         180
(1 row)
```

## CHORD\_TO\_ARC

Returns the length of the arc (shortest flying distance) between the two points. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
CHORD_TO_ARC (chord)
```

## Parameters

<i>chord</i>	A value of type DOUBLE PRECISION indicating chord length (in meters)
--------------	--

## Example

```
=>SELECT CHORD_TO_ARC(12000);
      CHORD_TO_ARC
-----
12000.0017738474
(1 row)
```

## DWITHIN

Determines whether a point (lat,lon) is within a circle of radius *d* kilometers centered at a given point (lat0,lon0). The return value has the type BOOLEAN.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

## Behavior Type

Immutable

## Syntax

```
DWITHIN (lat, lon, lat0, lon0, d)
```

## Parameters

<i>lat</i>	A value of type DOUBLE PRECISION indicating a given latitude.
<i>lon</i>	A value of type DOUBLE PRECISION indicating a given longitude.
<i>lat0</i>	A value of type DOUBLE PRECISION indicating the latitude of the center point of a circle.
<i>lon0</i>	A value of type DOUBLE PRECISION indicating the longitude of the center point of a circle.
<i>d</i>	A value of type DOUBLE PRECISION indicating the radius of the circle (in kilometers).

## Example

```
=>SELECT DWITHIN(13.6, 43.5, 48.5, 45.5, 1500);
      DWITHIN
-----
f
(1 row)
```

## ECEF\_CHORD

Converts the specified latitude, longitude, and height into an ECEF chord, measured in meters. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
ECEF_CHORD(lat1,lon1,h1, lat2, lon2, h2)
```

### Parameters

<i>lat</i>	A value of type DOUBLE PRECISION indicating the latitude of one end point of the line.
<i>lon1</i>	A value of type DOUBLE PRECISION indicating the longitude of one end point of the line.
<i>h1</i>	A value of type DOUBLE PRECISION indicating the height above sea level (in meters) of one end point of the line.
<i>lat2</i>	A value of type DOUBLE PRECISION indicating the latitude of one end point of the line.
<i>lon2</i>	A value of type DOUBLE PRECISION indicating the longitude of one end point of the line.
<i>h2</i>	A value of type DOUBLE PRECISION indicating the height of one end point of the line.

### Example

```
=> SELECT ECEF_chord (-12,10.0,14,12,-10,17);
      ECEF_chord
-----
 3411479.93992789
(1 row)
```

## ECEF\_x

Converts a specified latitude, longitude, and height into the ECEF x coordinate. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

### Behavior Type

Immutable

## Syntax

`ECEF_x(lat,lon,h)`

## Parameters

<i>lat</i>	A value of type DOUBLE PRECISION indicating latitude.
<i>lon</i>	A value of type DOUBLE PRECISION indicating longitude.
<i>h</i>	A value of type DOUBLE PRECISION indicating height.

## Example

```
=> SELECT ECEF_x(-12,13.2,0);
      ECEF_x
-----
6074803.56179976
(1 row)
```

## ECEF\_y

Converts a specified latitude, longitude, and height into the ECEF y coordinate. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See **Geospatial Package SQL Functions** (page 252) for information on installing the package.

## Behavior Type

Immutable

## Syntax

`ECEF_X(lat,lon,h)`

## Parameters

<i>lat</i>	A value of type DOUBLE PRECISION indicating latitude.
<i>lon</i>	A value of type DOUBLE PRECISION indicating longitude.
<i>h</i>	A value of type DOUBLE PRECISION indicating height.

## Example

```
=> SELECT ECEF_y(12.0,-14.2,12);
      ECEF_y
-----
-1530638.12327962
(1 row)
```

## ECEF\_z

Converts a specified latitude, longitude, and height into the ECEF z coordinate. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
ECEF_Z(lat,lon,h)
```

### Parameters

<i>lat</i>	A value of type DOUBLE PRECISION indicating latitude.
<i>lon</i>	A value of type DOUBLE PRECISION indicating longitude.
<i>h</i>	A value of type DOUBLE PRECISION indicating height.

### Example

```
=> SELECT ECEF_Z(12.0, -14.2, 12);
       ECEF_Z
-----
1317405.02616989
(1 row)
```

## ISLEFT

Determines whether a point (*x2*, *y2*) is anywhere to the left of a directed line which goes through point (*x0*,*y0*), then through point (*x1*,*y1*).The return value has the type BOOLEAN.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
ISLEFT(x0,y0, x1,y1,x2,y2)
```

### Parameters

<i>x0</i>	A value of type DOUBLE PRECISION indicating the latitude of the first point through which the directed line passes.
<i>y0</i>	A value of type DOUBLE PRECISION indicating the longitude of the the first point through which the directed line passes.
<i>x1</i>	A value of type DOUBLE PRECISION indicating the latitude of the second point through which the directed line passes.
<i>y1</i>	A value of type DOUBLE PRECISION indicating the longitude of the the second point through which the directed line passes.
<i>x2</i>	A value of type DOUBLE PRECISION indicating the latitude of the

	point whose position you are evaluating.
<i>y2</i>	A value of type DOUBLE PRECISION indicating the longitude of a whose position you are evaluating.

**Example**

```
=> SELECT ISLEFT(1,1,2,3,0,0);
      ISLEFT
-----
           1
(1 row)
```

**KM2MILES**

Converts a value from kilometers to miles. The return value is of type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**

KM2MILES (*km*)

**Parameters**

<i>km</i>	A value of type DOUBLE PRECISION indicating the number of kilometers you want to convert.
-----------	---

**Example**

```
=>SELECT KM2MILES(1.0);
      KM2MILES
-----
0.621371192237334
(1 row)
```

**LAT\_WITHIN**

Determines whether a latitude (*lat*) is within *d* kilometers of a given latitude point (*lat0*), independent of longitude. The return value has the type BOOLEAN.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**

LAT\_WITHIN(*lat, lat0, d*);

**Parameters**

<i>lat</i>	A value of type DOUBLE PRECISION indicating a given latitude.
<i>lat0</i>	A value of type DOUBLE PRECISION indicating the latitude of the point to which you are comparing the first latitude.
<i>d</i>	A value of type DOUBLE PRECISION indicating the number of kilometers that determines the range you are evaluating.

**Example**

```
=>SELECT LAT_WITHIN(12,14.0,1000);
LAT_WITHIN
-----
t
(1 row)
```

**LL\_WITHIN**

Determines whether a point (*lat*, *lon*) is within a bounding box whose sides are  $2d$  kilometers long, centered at a given point (*lat0*, *lon0*). The return value has the type BOOLEAN.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**

```
LL_WITHIN (lat,lon, lat0, lon0,d);
```

**Parameters**

<i>lat</i>	A value of type DOUBLE PRECISION indicating a given latitude.
<i>lon</i>	A value of type DOUBLE PRECISION indicating a given longitude.
<i>lat0</i>	A value of type DOUBLE PRECISION indicating the latitude of the center point of the bounding box.
<i>lon0</i>	A value of type DOUBLE PRECISION indicating the longitude of the center point of the bounding box.
<i>d</i>	A value of type DOUBLE PRECISION indicating the length of half the side of the box.

**Example**

```
=> SELECT LL_WITHIN(16,15,12,13.0,16);
LL_WITHIN
-----
f
(1 row)
```

## LLD\_WITHIN

Determines whether a point (lat,lon) is within a circle of radius *d* kilometers centered at a given point (lat0,lon0). LLD\_WITHIN is a faster, but less accurate version of **DWITHIN** (page 256). The return value has the type BOOLEAN.

This function is available only if you install the Vertica Geospatial Package package. See **Geospatial Package SQL Functions** (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
DWITHIN(lat, lon, lat0, lon0, d)
```

### Parameters

<i>lat</i>	A value of type DOUBLE PRECISION indicating a given latitude.
<i>lon</i>	A value of type DOUBLE PRECISION indicating a given longitude.
<i>lat0</i>	A value of type DOUBLE PRECISION indicating the latitude of the center point of a circle.
<i>lon0</i>	A value of type DOUBLE PRECISION indicating the longitude of the center point of a circle.
<i>d</i>	A value of type DOUBLE PRECISION indicating the radius of the circle (in kilometers).

### Example

```
=>SELECT LLD_WITHIN(13.6, 43.5, 48.5, 45.5, 1500) ;
  LLD_WITHIN
-----
  f
(1 row)
```

## LON\_WITHIN

Determines whether a longitude (lon) is within *d* kilometers of a given point (lat0, lon0). The return value has the type BOOLEAN.

This function is available only if you install the Vertica Geospatial Package package. See **Geospatial Package SQL Functions** (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**

```
lon_within(lon,lat0,lon0, d)
```

**Parameters**

<i>lon</i>	A value of type DOUBLE PRECISION indicating a given longitude.
<i>lat0</i>	A value of type DOUBLE PRECISION indicating the latitude of the point to which you want to compare the <i>lon</i> value.
<i>lon0</i>	A value of type DOUBLE PRECISION indicating the longitude of the point to which you want to compare the <i>lon</i> value.
<i>d</i>	A value of type DOUBLE PRECISION indicating the distance, in kilometers, that defines your range.

**Example**

```
=> SELECT LON_WITHIN(15,16,0,10);
   LON_WITHIN
-----
    f
(1 row)
```

**MILES2KM**

Converts a value from miles to kilometers. The return value is of type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**

```
MILES2KM (miles)
```

**Parameters**

<i>miles</i>	A value of type DOUBLE PRECISION indicating the number of miles you want to convert.
--------------	--

**Example**

```
=> SELECT MILES2KM(1.0);
   MILES2KM
-----
    1.609344
(1 row)
```

## RADIUS\_LON

Returns the radius of the circle of longitude at the given latitude. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
RADIUS_LON(lat)
```

### Parameters

<i>lat</i>	A value of type DOUBLE PRECISION indicating latitude at which you want to measure the radius.
------------	---

### Example

```
=> SELECT RADIUS_LON(-90);
       RADIUS_LON
-----
3.9053535478748e-13
(1 row)
```

## RADIUS\_M

Returns the earth's radius of curvature along the meridian. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
RADIUS_M(lat)
```

### Parameters

<i>lat</i>	A value of type DOUBLE PRECISION indicating latitude at which you want to measure the radius of curvature.
------------	--

### Example

```
=> SELECT RADIUS_M(-90);
```

```

RADIUS_M
-----
6399.5936257585
(1 row)

```

## RADIUS\_N

Returns the earth's radius of curvature normal to the meridian. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
RADIUS_N(lat)
```

### Parameters

<i>lat</i>	A value of type DOUBLE PRECISION indicating latitude at which you want to measure the radius of curvature.
------------	--

### Example

```

=> SELECT RADIUS_N(-90);
       RADIUS_N
-----
6399.59362575849
(1 row)

```

## radius\_r

Returns the WGS-84 radius of the earth (to the center of mass) at a given latitude. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
RADIUS_R(lat)
```

### Parameters

<i>lat</i>	A value of type DOUBLE PRECISION indicating latitude at which you want to measure the earth's radius.
------------	---

**Example**

```
=> SELECT RADIUS_R(-90);
      radius_R
-----
6356.75231424518
```

**RADIUS\_Ra**

Returns the earth's average radius of curvature at the given latitude. This function is the geometric mean of radius\_M and radius\_N. (**RADIUS\_Rv** (page 267) is a faster approximation of this function.)

The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See **Geospatial Package SQL Functions** (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**

RADIUS\_Ra (*lat*)

**Parameters**

<i>lat</i>	A value of type DOUBLE PRECISION indicating latitude at which you want to measure the radius of curvature.
------------	--

**Example**

```
=> SELECT RADIUS_Ra (-90) ;
      RADIUS_Ra
-----
6399.59362575849
(1 row)
```

**RADIUS\_Rc**

Returns the radius of curvature at a given bearing measured clockwise from north. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See **Geospatial Package SQL Functions** (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**RADIUS\_Rc(*lat*, *bearing*)**Parameters**

<i>lat</i>	A value of type DOUBLE PRECISION indicating latitude at which you want to measure the radius of curvature.
<i>bearing</i>	A value of type DOUBLE PRECISION indicating a given bearing.

**Example**

```
=> SELECT RADIUS_Rc(45,45);
      RADIUS_Rc
-----
6378.09200754445
(1 row)
```

**RADIUS\_Rv**

Returns the earth's average radius of curvature at the given latitude. This function is the geometric mean of radius\_M and radius\_N. This function is a fast approximation of **RADIUS\_Ra**. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See **Geospatial Package SQL Functions** (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**RADIUS\_Rv(*lat*)**Parameters**

<i>lat</i>	A value of type DOUBLE PRECISION indicating latitude at which you want to measure the radius of curvature.
------------	--

**Example**

```
=> SELECT RADIUS_Rv(-90);
      RADIUS_Rv
-----
6399.59362575849
(1 row)
```

## RADIUS\_SI

Returns the International System of Units (SI) radius based on the nautical mile. The return value has the type NUMERIC.

This function is available only if you install the Vertica Geospatial Package package. See *Geospatial Package SQL Functions* (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
RADIUS_SI()
```

### Example

```
=> SELECT RADIUS_SI();
      RADIUS_SI
-----
6366.70701949370750
(1 row)
```

## RAYCROSSING

Determines whether a ray traveling to the right from point (x2,y2) intersects the directed line segment that starts at point (x0,y0) and ends at point (x1,y1). If not, this function returns 0; if so, the function returns 1 if y1 is above y0, but -1 if y1 is below y0. The return value has the type DOUBLE PRECISION.

This function is available only if you install the Vertica Geospatial Package package. See *Geospatial Package SQL Functions* (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

```
RAYCROSSING (x0,y0, x1,y1, x2,y2)
```

### Parameters

<i>x0</i>	A value of type DOUBLE PRECISION indicating the latitude of the starting point of the line segment.
<i>y0</i>	A value of type DOUBLE PRECISION indicating the longitude of the starting point of the line segment
<i>x1</i>	A value of type DOUBLE PRECISION indicating the latitude of the ending point of the line segment.
<i>y1</i>	A value of type DOUBLE PRECISION indicating the longitude of

	the the ending point of the line segment.
x2	A value of type DOUBLE PRECISION indicating the latitude of the point whose position you are evaluating.
y2	A value of type DOUBLE PRECISION indicating the longitude of the point whose position you are evaluating.

**Example**

```
=> SELECT RAYCROSSING(1,1,2,3,0,0);
```

```
RAYCROSSING
```

```
-----
                0
```

```
(1 row)
```

```
=> SELECT RAYCROSSING(1,1,2,3,0,2);
```

```
RAYCROSSING
```

```
-----
                1
```

```
(1 row)
```

**WGS84\_a**

Returns the length, in kilometers, of the earth's semi-major axis. The return value is of type NUMERIC.

This function is available only if you install the Vertica Geospatial Package package. See *Geospatial Package SQL Functions* (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**

```
WGS84_a()
```

**Example**

```
=>SELECT WGS84_a();
```

```
      wgs84_a
```

```
-----
6378.137000
```

```
(1 row)
```

**WGS84\_b**

Returns the WGS-84 semi-minor axis length value in kilometers. The return value is of type NUMERIC.

This function is available only if you install the Vertica Geospatial Package package. See *Geospatial Package SQL Functions* (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**`WGS84_b()`**Example**

```
=> SELECT WGS84_b();
      WGS84_b
-----
6356.75231424517950
(1 row)
```

**WGS84\_e2**

Returns the WGS-84 eccentricity squared value. The return value is of type NUMERIC.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**`WGS84_e2()`**Example**

```
=> SELECT WGS84_e2();
      WGS84_e2
-----
.00669437999014131700
(1 row)
```

**WGS84\_f**

Returns the WGS-84 flattening value. The return value is of type NUMERIC.

This function is available only if you install the Vertica Geospatial Package package. See ***Geospatial Package SQL Functions*** (page 252) for information on installing the package.

**Behavior Type**

Immutable

**Syntax**`WGS84_f()`**Example**

```
=> SELECT WGS84_f();
      WGS84_f
```

```
-----
.00335281066474748072
(1 row)
```

## WGS84\_if

Returns the WGS-84 inverse flattening value. The return value is of type NUMERIC.

This function is available only if you install the Vertica Geospatial Package package. See *Geospatial Package SQL Functions* (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

WGS84\_if()

### Example

```
=> SELECT WGS84_if();
      WGS84_if
-----
298.257223563
(1 row)
```

## WGS84\_r1

Returns the International Union of Geodesy and Geophysics (IUGG) mean radius of the earth, in kilometers. The return value is of type NUMERIC.

This function is available only if you install the Vertica Geospatial Package package. See *Geospatial Package SQL Functions* (page 252) for information on installing the package.

### Behavior Type

Immutable

### Syntax

WGS84\_r1()

### Example

```
=> SELECT WGS84_r1();
      WGS84_r1
-----
6371.00877141505983
(1 row)
```

## IP Conversion Functions

IP functions perform conversion, calculation, and manipulation operations on IP, network, and subnet addresses.

## INET\_ATON

Returns an integer that represents the value of the address in host byte order, given the dotted-quad representation of a network address as a string.

### Behavior Type

Immutable

### Syntax

```
INET_ATON ( expression )
```

### Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

### Notes

The following syntax converts an IPv4 address represented as the string A to an integer I.

INET\_ATON trims any spaces from the right of A, calls the Linux function *inet\_pton* [http://www.opengroup.org/onlinepubs/000095399/functions/inet\\_pton.html](http://www.opengroup.org/onlinepubs/000095399/functions/inet_pton.html), and converts the result from network byte order to host byte order using *ntohl* <http://opengroup.org/onlinepubs/007908775/xns/ntohl.html>.

```
INET_ATON(VARCHAR A) -> INT8 I
```

If A is NULL, too long, or *inet\_pton* returns an error, the result is NULL.

### Examples

The generated number is always in host byte order. In the following example, the number is calculated as  $209 \times 256^3 + 207 \times 256^2 + 224 \times 256 + 40$ .

```
SELECT INET_ATON('209.207.224.40');
inet_aton
-----
3520061480
(1 row)
SELECT INET_ATON('1.2.3.4');
inet_aton
-----
16909060
(1 row)
SELECT TO_HEX(INET_ATON('1.2.3.4'));
to_hex
-----
1020304
(1 row)
```

### See Also

**INET\_NTOA** (page 273)

## INET\_NTOA

Returns the dotted-quad representation of the address as a VARCHAR, given a network address as an integer in network byte order.

### Behavior Type

Immutable

### Syntax

```
INET_NTOA ( expression )
```

### Parameters

<i>expression</i>	(INTEGER) is the network address to convert.
-------------------	--

### Notes

The following syntax converts an IPv4 address represented as integer I to a string A.

INET\_NTOA converts I from host byte order to network byte order using *htonl* <http://opengroup.org/onlinepubs/007908775/xns/htonl.html>, and calls the Linux function *inet\_ntop* [http://www.opengroup.org/onlinepubs/000095399/functions/inet\\_ntop.html](http://www.opengroup.org/onlinepubs/000095399/functions/inet_ntop.html).

```
INET_NTOA(INT8 I) -> VARCHAR A
```

If I is NULL, greater than 2<sup>32</sup> or negative, the result is NULL.

### Examples

```
SELECT INET_NTOA(16909060);
  inet_ntoa
-----
  1.2.3.4
(1 row)
SELECT INET_NTOA(03021962);
  inet_ntoa
-----
  0.46.28.138
(1 row)
```

### See Also

**INET\_ATON** (page 272)

## V6\_ATON

Converts an IPv6 address represented as a character string to a binary string.

### Behavior Type

Immutable

### Syntax

```
V6_ATON ( expression )
```

## Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

## Notes

The following syntax converts an IPv6 address represented as the character string A to a binary string B.

V6\_ATON trims any spaces from the right of A and calls the Linux function *inet\_pton* [http://www.opengroup.org/onlinepubs/000095399/functions/inet\\_pton.html](http://www.opengroup.org/onlinepubs/000095399/functions/inet_pton.html).

V6\_ATON (VARCHAR A) -> VARBINARY (16) B

If A has no colons it is prepended with '::inet\_pton returns an error, the result is NULL.

## Examples

```
SELECT V6_ATON('2001:DB8::8:800:200C:417A');
           v6_aton
-----
\001\015\270\000\000\000\000\000\010\010\000 \014Az
(1 row)
```

```
SELECT TO_HEX(V6_ATON('2001:DB8::8:800:200C:417A'));
           to_hex
-----
20010db8000000000000080800200c417a
(1 row)
```

```
SELECT V6_ATON('1.2.3.4');
           v6_aton
-----
\000\000\000\000\000\000\000\000\000\000\000\000\377\377\001\002\003\004
(1 row)
```

```
SELECT V6_ATON('::1.2.3.4');
           v6_aton
-----
\000\000\000\000\000\000\000\000\000\000\000\000\000\001\002\003\004
(1 row)
```

## See Also

**V6\_NTOA** (page 275)

## V6\_NTOA

Converts an IPv6 address represented as varbinary to a character string.

### Behavior Type

Immutable

### Syntax

```
V6_NTOA ( expression )
```

### Parameters

<i>expression</i>	(VARBINARY) is the binary string to convert.
-------------------	--

### Notes

The following syntax converts an IPv6 address represented as VARBINARY B to a string A.

V6\_NTOA right-pads B to 16 bytes with zeros, if necessary, and calls the Linux function *inet\_ntop* [http://www.opengroup.org/onlinepubs/000095399/functions/inet\\_ntop.html](http://www.opengroup.org/onlinepubs/000095399/functions/inet_ntop.html).

```
V6_NTOA(VARBINARY B) -> VARCHAR A
```

If B is NULL or longer than 16 bytes, the result is NULL.

Vertica automatically converts the form '::ffff:1.2.3.4' to '1.2.3.4'.

### Examples

```
SELECT V6_NTOA(' \001\015\270\000\000\000\000\000\010\010\000 \014Az');
      v6_ntoa
-----
2001:db8::8:800:200c:417a
(1 row)
SELECT V6_NTOA(V6_ATON('1.2.3.4'));
      v6_ntoa
-----
1.2.3.4
(1 row)
SELECT V6_NTOA(V6_ATON('::1.2.3.4'));
      v6_ntoa
-----
::1.2.3.4
(1 row)
```

### See Also

**N6\_ATON** (page 273)

## V6\_SUBNETA

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a binary or alphanumeric IPv6 address.

## Behavior Type

Immutable

## Syntax

V6\_SUBNETA ( *expression1*, *expression2* )

## Parameters

<i>expression1</i>	(VARBINARY or VARCHAR) is the string to calculate.
<i>expression2</i>	(INTEGER) is the size of the subnet.

## Notes

The following syntax calculates a subnet address in CIDR format from a binary or varchar IPv6 address.

V6\_SUBNETA masks a binary IPv6 address B so that the N leftmost bits form a subnet address, while the remaining rightmost bits are cleared. It then converts to an alphanumeric IPv6 address, appending a slash and N.

V6\_SUBNETA(BINARY B, INT8 N) -> VARCHAR C

The following syntax calculates a subnet address in CIDR format from an alphanumeric IPv6 address.

V6\_SUBNETA(VARCHAR A, INT8 N) -> V6\_SUBNETA(V6\_ATON(A), N) -> VARCHAR C

## Examples

```
SELECT V6_SUBNETA(V6_ATON('2001:db8::8:800:200c:417a'), 28);
   v6_subneta
-----
2001:db0::/28
(1 row)
```

## See Also

**V6\_SUBNETN** (page 276)

## V6\_SUBNETN

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a varbinary or alphanumeric IPv6 address.

## Behavior Type

Immutable

## Syntax

V6\_SUBNETN ( *expression1*, *expression2* )



## Notes

V6\_TYPE(VARBINARY B) returns INT8 T.

V6\_TYPE(VARCHAR A) -> V6\_TYPE(V6\_ATON(A)) -> INT8 T

The IPv6 types are defined in the Network Working Group's *IP Version 6 Addressing Architecture memo* <http://www.ietf.org/rfc/rfc4291.txt>.

GLOBAL = 0	Global unicast addresses
LINKLOCAL = 1	Link-Local unicast (and Private-Use) addresses
LOOPBACK = 2	Loopback
UNSPECIFIED = 3	Unspecified
MULTICAST = 4	Multicast

IPv4-mapped and IPv4-compatible IPv6 addresses are also interpreted, as specified in *IPv4 Global Unicast Address Assignments* <http://www.iana.org/assignments/ipv4-address-space>.

- For IPv4, Private-Use is grouped with Link-Local.
- If B is VARBINARY, it is right-padded to 16 bytes with zeros, if necessary.
- If B is NULL or longer than 16 bytes, the result is NULL.

## Details

IPv4 (either kind):

0.0.0.0/8	UNSPECIFIED
10.0.0.0/8	LINKLOCAL
127.0.0.0/8	LOOPBACK
169.254.0.0/16	LINKLOCAL
172.16.0.0/12	LINKLOCAL
192.168.0.0/16	LINKLOCAL
224.0.0.0/4	MULTICAST
others	GLOBAL

IPv6:

::0/128	UNSPECIFIED
::1/128	LOOPBACK
fe80::/10	LINKLOCAL
ff00::/8	MULTICAST
others	GLOBAL

## Examples

```
SELECT V6_TYPE(V6_ATON('192.168.2.10'));
 v6_type
-----
          1
(1 row)
SELECT V6_TYPE(V6_ATON('2001:db8::8:800:200c:417a'));
 v6_type
-----
          0
(1 row)
```

**See Also**

**INET\_ATON** (page 272)

**IP Version 6 Addressing Architecture** <http://www.ietf.org/rfc/rfc4291.txt>

**IPv4 Global Unicast Address Assignments**

<http://www.iana.org/assignments/ipv4-address-space>

## Mathematical Functions

Some of these functions are provided in multiple forms with different argument types. Except where noted, any given form of a function returns the same data type as its argument. The functions working with `DOUBLE PRECISION` (page 97) data could vary in accuracy and behavior in boundary cases depending on the host system.

**See Also**

**Template Patterns for Numeric Formatting** (page 249)

### ABS

Returns the absolute value of the argument. The return value has the same data type as the argument..

**Behavior Type**

Immutable

**Syntax**

```
ABS ( expression )
```

**Parameters**

<i>expression</i>	Is a value of type INTEGER or DOUBLE PRECISION
-------------------	--

**Examples**

```
SELECT ABS(-28.7);
abs
-----
28.7
(1 row)
```

### ACOS

Returns a `DOUBLE PRECISION` value representing the trigonometric inverse cosine of the argument.

**Behavior Type**

Immutable

**Syntax**

```
ACOS ( expression )
```

**Parameters**

<i>expression</i>	Is a value of type DOUBLE PRECISION
-------------------	-------------------------------------

**Example**

```
SELECT ACOS (1);
      acos
-----
       0
(1 row)
```

**ASIN**

Returns a DOUBLE PRECISION value representing the trigonometric inverse sine of the argument.

**Behavior Type**

Immutable

**Syntax**

```
ASIN ( expression )
```

**Parameters**

<i>expression</i>	Is a value of type DOUBLE PRECISION
-------------------	-------------------------------------

**Example**

```
SELECT ASIN(1);
      asin
-----
1.5707963267949
(1 row)
```

**ATAN**

Returns a DOUBLE PRECISION value representing the trigonometric inverse tangent of the argument.

**Behavior Type**

Immutable

**Syntax**

```
ATAN ( expression )
```

**Parameters**

<i>expression</i>	Is a value of type DOUBLE PRECISION
-------------------	-------------------------------------

**Example**

```
SELECT ATAN(1);
       atan
```

```
-----
0.785398163397448
(1 row)
```

**ATAN2**

Returns a DOUBLE PRECISION value representing the trigonometric inverse tangent of the arithmetic dividend of the arguments.

**Behavior Type**

Immutable

**Syntax**

```
ATAN2 ( quotient, divisor )
```

**Parameters**

<i>quotient</i>	Is an expression of type DOUBLE PRECISION representing the quotient
<i>divisor</i>	Is an expression of type DOUBLE PRECISION representing the divisor

**Example**

```
SELECT ATAN2(2,1);
       atan2
```

```
-----
1.10714871779409
(1 row)
```

**CBRT**

Returns the cube root of the argument. The return value has the type DOUBLE PRECISION.

**Behavior Type**

Immutable

**Syntax**

```
CBRT ( expression )
```

**Parameters**

<i>expression</i>	Is a value of type DOUBLE PRECISION
-------------------	-------------------------------------

## Examples

```
SELECT CBRT(27.0);
  cbrt
-----
      3
(1 row)
```

## CEILING (CEIL)

Rounds the returned value up to the next whole number. Any expression that contains even a slight decimal is rounded up.

### Behavior Type

Immutable

### Syntax

```
CEILING ( expression )
CEIL ( expression )
```

### Parameters

<i>expression</i>	Is a value of type INTEGER or DOUBLE PRECISION
-------------------	--

### Notes

CEILING is the opposite of **FLOOR** (page 286), which rounds the returned value down:

```
=> SELECT CEIL(48.01) AS ceiling, FLOOR(48.01) AS floor;
  ceiling | floor
-----+-----
        49 |      48
(1 row)
```

### Examples

```
=> SELECT CEIL(-42.8);
  CEIL
-----
   -42
(1 row)
SELECT CEIL(48.01);
  CEIL
-----
     49
(1 row)
```

## COS

Returns a DOUBLE PRECISION value representing the trigonometric cosine of the argument.

**Behavior Type**

Immutable

**Syntax**COS ( *expression* )**Parameters**

<i>expression</i>	Is a value of type DOUBLE PRECISION
-------------------	-------------------------------------

**Example**

```
SELECT COS(-1);
       cos
```

```
-----
0.54030230586814
(1 row)
```

**COT**

Returns a DOUBLE PRECISION value representing the trigonometric cotangent of the argument.

**Behavior Type**

Immutable

**Syntax**COT ( *expression* )**Parameters**

<i>expression</i>	Is a value of type DOUBLE PRECISION
-------------------	-------------------------------------

**Example**

```
SELECT COT(1);
       cot
```

```
-----
0.642092615934331
(1 row)
```

**DEGREES**

Converts an expression from *radians* (page 292) to fractional degrees, or from degrees, minutes, and seconds to fractional degrees. The return value has the type DOUBLE PRECISION.

**Behavior Type**

Immutable

**Syntax**DEGREES ( *radians* )

## Syntax2

DEGREES (*degrees, minutes, seconds*)

### Parameters

<i>radians</i>	A unit of angular measure, $2\pi$ radians is equal to a full rotation.
<i>degrees</i>	A unit of angular measure, equal to 1/360 of a full rotation.
<i>minutes</i>	A unit of angular measurement, representing 1/60 of a degree.
<i>seconds</i>	A unit of angular measurement, representing 1/60 of a minute.

### Examples

```
SELECT DEGREES(0.5);
      DEGREES
-----
 28.6478897565412
(1 row)
```

```
SELECT DEGREES(1,2,3);
      DEGREES
-----
 1.034166666666667
(1 row)
```

## DISTANCE

Returns the distance (in kilometers) between two points. You specify the latitude and longitude of both the starting point and the ending point. You can also specify the radius of curvature for greater accuracy when using an ellipsoidal model.

### Behavior type

Immutable

### Syntax

DISTANCE ( *lat0, lon0, lat1, lon1, radius\_of\_curvature* )

### Parameters

<i>lat0</i>	Specifies the latitude of the starting point.
<i>lon0</i>	Specifies the longitude of the starting point.
<i>lat1</i>	Specifies the latitude of the ending point
<i>lon1</i>	Specifies the longitude of the ending point.
<i>radius_of_curvature</i>	Specifies the radius of the curvature of the earth at the midpoint between the starting and ending points. This parameter allows for greater accuracy when using an ellipsoidal earth model. If you do not specify this parameter, it defaults to the WGS-84 average $r_1$ radius, about 6371.009 km.

## Examples

This example finds the distance in kilometers for 1 degree of longitude at latitude 45 degrees, assuming earth is spherical.

```
SELECT DISTANCE(45,0, 45,1);
       distance
-----
 78.6262959272162
(1 row)
```

## DISTANCEV

Returns the distance (in kilometers) between two points using the Vincenty formula. Because the Vincenty formula includes the parameters of the WGS-84 ellipsoid model, you need not specify a radius of curvature. You specify the latitude and longitude of both the starting point and the ending point. This function is more accurate, but will be slower, than the DISTANCE function.

### Behavior type

Immutable

### Syntax

```
DISTANCEV (lat0, lon0, lat1, lon1);
```

### Parameter

<i>lat0</i>	Specifies the latitude of the starting point.
<i>lon0</i>	Specifies the longitude of the starting point.
<i>lat1</i>	Specifies the latitude of the ending point
<i>lon1</i>	Specifies the longitude of the ending point.

## Examples

This example finds the distance in kilometers for 1 degree of longitude at latitude 45 degrees, assuming earth is ellipsoidal.

```
SELECT DISTANCEV(45,0, 45,1);
       distanceV
-----
 78.8463347095916
(1 row)
```

## EXP

Returns the exponential function, e to the power of a number. The return value has the same data type as the argument.

## Behavior Type

Immutable

## Syntax

```
EXP ( exponent )
```

## Parameters

<i>exponent</i>	Is an expression of type INTEGER or DOUBLE PRECISION
-----------------	--

## Example

```
SELECT EXP(1.0);
       exp
-----
 2.71828182845905
(1 row)
```

## FLOOR

Rounds the returned value down to the next whole number. Any expression that contains even a slight decimal is rounded down.

## Behavior Type

Immutable

## Syntax

```
FLOOR ( expression )
```

## Parameters

<i>expression</i>	Is an expression of type INTEGER or DOUBLE PRECISION.
-------------------	---

## Notes

FLOOR is the opposite of **CEILING** (page 282), which rounds the returned value up:

```
=> SELECT FLOOR(48.01) AS floor, CEIL(48.01) AS ceiling;
       floor | ceiling
-----+-----
         48 |         49
(1 row)
```

## Examples

```
=> SELECT FLOOR((TIMESTAMP '2005-01-17 10:00' - TIMESTAMP '2005-01-01') / INTERVAL '7');
       floor
-----
         2
(1 row)
=> SELECT FLOOR(-42.8);
```

```

floor
-----
-43
(1 row)
=> SELECT FLOOR(42.8);
floor
-----
42
(1 row)

```

Although the following example looks like an INTEGER, the number on the left is 2<sup>49</sup> as an INTEGER, but the number on the right is a FLOAT:

```

=> SELECT 1<<49, FLOOR(1 << 49);
?column? | floor
-----+-----
562949953421312 | 562949953421312
(1 row)

```

Compare the above example to:

```

=> SELECT 1<<50, FLOOR(1 << 50);
?column? | floor
-----+-----
1125899906842624 | 1.12589990684262e+15
(1 row)

```

## HASH

Calculates a hash value over its arguments, producing a value in the range  $0 \leq x < 2^{63}$  (two to the sixty-third power or 2<sup>63</sup>).

### Behavior Type

Immutable

### Syntax

```
HASH ( expression [ ,... ] )
```

### Parameters

<i>expression</i>	Is an expression of any data type. For the purpose of hash segmentation, each expression is a <b>column reference</b> (see " <b>Column References</b> " on page 46).
-------------------	--

### Notes

- The HASH() function is used to provide projection segmentation over a set of nodes in a cluster and takes up to 32 arguments, usually column names, and selects a specific node for each row based on the values of the columns for that row. HASH (Col1, Col2).
- If your data is fairly regular and you want more even distribution than you get with HASH, consider using **MODULARHASH** (page 290)() for project segmentation.

### Examples

```
SELECT HASH(product_price, product_cost)
```

```
FROM product_dimension
WHERE product_price = '11';
      hash
```

```
-----
4157497907121511878
1799398249227328285
3250220637492749639
(3 rows)
```

**See Also**

**MODULARHASH** (page 290)

**LN**

Returns the natural logarithm of the argument. The return data type is the same as the argument.

**Behavior Type**

Immutable

**Syntax**

LN ( *expression* )

**Parameters**

<i>expression</i>	Is an expression of type INTEGER or DOUBLE PRECISION
-------------------	--

**Examples**

```
SELECT LN(2);
      ln
```

```
-----
0.693147180559945
(1 row)
```

**LOG**

Returns the logarithm to the specified base of the argument. The return data type is the same as the argument.

**Behavior Type**

Immutable

**Syntax**

LOG ( [ *base*, ] *expression* )

**Parameters**

<i>base</i>	Specifies the base (default is base 10)
<i>expression</i>	Is an expression of type INTEGER or DOUBLE PRECISION

## Examples

```

SELECT LOG(2.0, 64);
  log
-----
    6
(1 row)
SELECT LOG(100);
  log
-----
    2
(1 row)

```

## MOD

Returns the remainder of a division operation. MOD is also called `modulo`.

### Behavior Type

Immutable

### Syntax

```
MOD( expression1, expression2 )
```

### Parameters

<i>expression1</i>	Specifies the dividend (INTEGER, NUMERIC, or FLOAT)
<i>expression2</i>	Specifies the divisor (type same as dividend)

### Notes

When computing `mod(N,M)`, the following rules apply:

- If either N or M is the null value, then the result is the null value.
- If M is zero, then an exception condition is raised: data exception — division by zero.
- Otherwise, the result is the unique exact numeric value R with scale 0 (zero) such that all of the following are true:
  - R has the same sign as N.
  - The absolute value of R is less than the absolute value of M.
  - $N = M * K + R$  for some exact numeric value K with scale 0 (zero).

### Examples

```

SELECT MOD(9, 4);
  mod
-----
    1
(1 row)
SELECT MOD(10, 3);
  mod
-----
    1

```

```
(1 row)
SELECT MOD(-10,3);
mod
-----
-1
(1 row)
SELECT MOD(-10,-3);
mod
-----
-1
(1 row)
SELECT MOD(10,-3);
mod
-----
1
(1 row)
```

MOD(<float>, 0) gives an error:

```
=> SELECT MOD(6.2,0);
ERROR: numeric division by zero
```

## MODULARHASH

Calculates a hash value over its arguments for the purpose of projection segmentation. In all other uses, returns 0.

If you can hash segment your data using a column with a regular pattern, such as a sequential unique identifier, MODULARHASH distributes the data more evenly than HASH, which distributes data using a normal statistical distribution.

### Behavior Type

Immutable

### Syntax

```
MODULARHASH ( expression [ ,... ] )
```

### Parameters

<i>expression</i>	Is a <b>column reference</b> (see " <b>Column References</b> " on page 46) of any data type.
-------------------	--

### Notes

The MODULARHASH() function takes up to 32 arguments, usually column names, and selects a specific node for each row based on the values of the columns for that row.

### Examples

```
CREATE PROJECTION fact_ts_2 (f_price, f_cid, f_tid, f_cost, f_date)
AS (SELECT price, cid, tid, cost, dwdate
    FROM fact)
    SEGMENTED BY MODULARHASH(dwdate)
    ALL NODES OFFSET 2;
```

**See Also****HASH** (page 287)**PI**

Returns the constant pi ( $\Pi$ ), the ratio of any circle's circumference to its diameter in Euclidean geometry. The return type is DOUBLE PRECISION.

**Behavior Type**

Immutable

**Syntax**

PI ( )

**Examples**

```
SELECT PI ( );
       pi
-----
 3.14159265358979
(1 row)
```

**POWER**

Returns a DOUBLE PRECISION value representing one number raised to the power of another number.

**Behavior Type**

Immutable

**Syntax**POWER ( *expression1*, *expression2* )**Parameters**

<i>expression1</i>	Is an expression of type DOUBLE PRECISION that represents the base
<i>expression2</i>	Is an expression of type DOUBLE PRECISION that represents the exponent

**Examples**

```
SELECT POWER(9.0, 3.0);
       power
-----
        729
(1 row)
```

## RADIANS

Returns a DOUBLE PRECISION value representing an angle expressed in radians. You can express the input angle in **degrees** (page 283), and optionally include minutes and seconds.

### Behavior Type

Immutable

### Syntax

```
RADIANS (degrees [, minutes, seconds])
```

### Parameters

<i>degrees</i>	A unit of angular measurement, representing 1/360 of a full rotation.
<i>minutes</i>	A unit of angular measurement, representing 1/60 of a degree.
<i>seconds</i>	A unit of angular measurement, representing 1/60 of a minute.

### Examples

```
SELECT RADIANS(45);
       RADIANS
-----
0.785398163397448
(1 row)
```

```
SELECT RADIANS (1,2,3);
       RADIANS
-----
0.018049613347708
(1 row)
```

## RANDOM

Returns a uniformly-distributed random number  $x$ , where  $0 \leq x < 1$ .

### Behavior Type

Volatile

### Syntax

```
RANDOM()
```

## Parameters

RANDOM has no arguments. Its result is a FLOAT8 data type (also called **DOUBLE PRECISION** (page 97)).

## Notes

Typical pseudo-random generators accept a seed, which is set to generate a reproducible pseudo-random sequence. Vertica, however, distributes SQL processing over a cluster of nodes, where each node generates its own independent random sequence.

Results depending on RANDOM are not reproducible because the work might be divided differently across nodes. Therefore, Vertica automatically generates truly random seeds for each node each time a request is executed and does not provide a mechanism for forcing a specific seed.

## Examples

In the following example, the result is a float, which is  $\geq 0$  and  $< 1.0$ :

```
SELECT RANDOM();
      random
-----
0.211625560652465
(1 row)
```

## RANDOMINT

Returns a uniformly-distributed integer I, where  $0 \leq I < N$ , where  $N \leq \text{MAX\_INT8}$ . That is, RANDOMINT(N) returns one of the N integers from 0 through N-1.

## Behavior Type

Volatile

## Syntax

```
RANDOMINT ( N )
```

## Example

In the following example, the result is an INT8, which is  $\geq 0$  and  $< N$ . In this case, INT8 is randomly chosen from the set {0,1,2,3,4}.

```
SELECT RANDOMINT(5);
      randomint
-----
3
(1 row)
```

## ROUND

Rounds a value to a specified number of decimal places, retaining the original scale and precision. Fractions greater than or equal to .5 are rounded up. Fractions less than .5 are rounded down (truncated).

## Behavior Type

Immutable

## Syntax

```
ROUND ( expression [ , decimal-places ] )
```

## Parameters

<i>expression</i>	Is an expression of type NUMERIC.
<i>decimal-places</i>	If positive, specifies the number of decimal places to display to the right of the decimal point; if negative, specifies the number of decimal places to display to the left of the decimal point.

## Notes

NUMERIC ROUND() returns NUMERIC, retaining the original scale and precision:

```
=> SELECT ROUND(3.5);
      ROUND
-----
       4.0
(1 row)
```

The internal floating point representation used to compute the ROUND function causes the fraction to be evaluated as 3.5, which is rounded up.

## Examples

```
SELECT ROUND(2.0, 1.0 ) FROM dual;
      round
-----
         2
(1 row)
SELECT ROUND(12.345, 2.0 );
      round
-----
      12.35
(1 row)

SELECT ROUND(3.4444444444444444);
      ROUND
-----
3.0000000000000000
(1 row)
SELECT ROUND(3.14159, 3);
      ROUND
-----
      3.14200
(1 row)
SELECT ROUND(1234567, -3);
      round
-----
     1235000
(1 row)
```

```

SELECT ROUND(3.4999, -1);
  ROUND
-----
.0000
(1 row)
SELECT employee_last_name, ROUND(annual_salary,4) FROM
employee_dimension;
 employee_last_name | ROUND
-----+-----
Li                  | 1880
Rodriguez           | 1704
Goldberg            | 2282
Meyer               | 1628
Pavlov              | 3168
McNulty             | 1516
Dobisz              | 3006
Pavlov              | 2142
Goldberg            | 2268
Pavlov              | 1918
Robinson            | 2366
...

```

## SIGN

Returns a DOUBLE PRECISION value of -1, 0, or 1 representing the arithmetic sign of the argument.

### Behavior Type

Immutable

### Syntax

```
SIGN ( expression )
```

### Parameters

<i>expression</i>	Is an expression of type DOUBLE PRECISION
-------------------	---

### Examples

```

SELECT SIGN(-8.4);
  sign
-----
-1
(1 row)

```

## SIN

Returns a DOUBLE PRECISION value representing the trigonometric sine of the argument.

### Behavior Type

Immutable

**Syntax**

```
SIN ( expression )
```

**Parameters**

<i>expression</i>	Is an expression of type DOUBLE PRECISION
-------------------	---

**Example**

```
SELECT SIN(30 * 2 * 3.14159 / 360);
      sin
-----
0.4999999616987256
(1 row)
```

**SQRT**

Returns a DOUBLE PRECISION value representing the arithmetic square root of the argument.

**Behavior Type**

Immutable

**Syntax**

```
SQRT ( expression )
```

**Parameters**

<i>expression</i>	Is an expression of type DOUBLE PRECISION
-------------------	---

**Examples**

```
SELECT SQRT(2);
      sqrt
-----
1.4142135623731
(1 row)
```

**TAN**

Returns a DOUBLE PRECISION value representing the trigonometric tangent of the argument.

**Behavior Type**

Immutable

**Syntax**

```
TAN ( expression )
```

**Parameters**

<i>expression</i>	Is an expression of type DOUBLE PRECISION
-------------------	---

**Example**

```
SELECT TAN(30);
      tan
-----
-6.40533119664628
(1 row)
```

**TRUNC**

Returns a value representing the argument fully truncated (toward zero) or truncated to a specific number of decimal places, retaining the original scale and precision.

**Behavior Type**

Immutable

**Syntax**

```
TRUNC ( expression [ , places ]
```

**Parameters**

<i>expression</i>	Is an expression of type INTEGER or DOUBLE PRECISION that represents the number to truncate
<i>places</i>	Is an expression of type INTEGER that specifies the number of decimal places to return

**Notes**

NUMERIC TRUNC() returns NUMERIC, retaining the original scale and precision:

```
=> SELECT TRUNC(3.5);
      TRUNC
-----
      3.0
(1 row)
```

**Examples**

```
=>SELECT TRUNC(42.8);
      TRUNC
-----
      42.0
(1 row)
=>SELECT TRUNC(42.4382, 2);
      TRUNC
-----
      42.4300
(1 row)
```

## WIDTH\_BUCKET

Constructs equiwidth histograms, in which the histogram range is divided into intervals (buckets) of identical sizes. In addition, values below the low bucket return 0, and values above the high bucket return bucket\_count +1. Returns an integer value.

### Behavior Type

Immutable

### Syntax

```
WIDTH_BUCKET ( expression, hist_min, hist_max, bucket_count )
```

### Parameters

<i>expression</i>	Is the expression for which the histogram is created. This expression must evaluate to a numeric or datetime value or to a value that can be implicitly converted to a numeric or datetime value. If <i>expression</i> evaluates to null, then the <i>expression</i> returns null.
<i>hist_min</i>	Is an expression that resolves to the low boundary of bucket 1. Must also evaluate to numeric or datetime values and cannot evaluate to null.
<i>hist_max</i>	Is an expression that resolves to the high boundary of bucket bucket_count. Must also evaluate to a numeric or datetime value and cannot evaluate to null.
<i>bucket_count</i>	Is an expression that resolves to a constant, indicating the number of buckets. This expression always evaluates to a positive INTEGER.

### Notes

- WIDTH\_BUCKET divides a data set into buckets of equal width. For example, Age = 0-20, 20-40, 40-60, 60-80. This is known as an equiwidth histogram.
- When using WIDTH\_BUCKET pay attention to the minimum and maximum boundary values. Each bucket contains values equal to or greater than the base value of that bucket, so that age ranges of 0-20, 20-40, and so on, are actually 0-19.99 and 20-39.999.
- WIDTH\_BUCKET accepts the following data types: (FLOAT and/or INT), (TIMESTAMP and/or DATE and/or TIMESTAMPTZ), or (INTERVAL and/or TIME).

### Examples

The following example returns five possible values and has three buckets: 0 [Up to 100), 1 [100-300), 2 [300-500), 3 [500-700), and 4 [700 and up):

```
SELECT product_description, product_cost,  
WIDTH_BUCKET(product_cost, 100, 700, 3);
```

The following example creates a nine-bucket histogram on the `annual_income` column for customers in Connecticut who are female doctors. The results return the bucket number to an "Income" column, divided into eleven buckets, including an underflow and an overflow. Note that if customers had an annual incomes greater than the maximum value, they would be assigned to an overflow bucket, 10:

```
SELECT customer_name, annual_income,
WIDTH_BUCKET (annual_income, 100000, 1000000, 9) AS "Income"
FROM public.customer_dimension WHERE customer_state='CT'
AND title='Dr.' AND customer_gender='Female' AND household_id < '1000'
ORDER BY "Income";
```

In the following result set, the reason there is a bucket 0 is because buckets are numbered from 1 to `bucket_count`. Anything less than the given value of `hist_min` goes in bucket 0, and anything greater than the given value of `hist_max` goes in the bucket `bucket_count+1`. In this example, bucket 9 is empty, and there is no overflow. The value 12,283 is less than 100,000, so it goes into the underflow bucket.

customer_name	annual_income	Income
Joanna A. Nguyen	12283	0
Amy I. Nguyen	109806	1
Juanita L. Taylor	219002	2
Carla E. Brown	240872	2
Kim U. Overstreet	284011	2
Tiffany N. Reyes	323213	3
Rebecca V. Martin	324493	3
Betty . Roy	476055	4
Midori B. Young	462587	4
Martha T. Brown	687810	6
Julie D. Miller	616509	6
Julie Y. Nielson	894910	8
Sarah B. Weaver	896260	8
Jessica C. Nielson	861066	8

(14 rows)

## See Also

**NTILE** (page 159)

## NULL-handling Functions

NULL-handling functions take arguments of any type, and their return type is based on their argument types.

### COALESCE

Returns the value of the first non-null expression in the list. If all expressions evaluate to null, then the COALESCE function returns null.

#### Behavior Type

Immutable

## Syntax

```
COALESCE ( expression1, expression2 );  
COALESCE ( expression1, expression2, ... expression-n );
```

## Parameters

- COALESCE (expression1, expression2) is equivalent to the following CASE expression:  
CASE WHEN expression1 IS NOT NULL THEN expression1 ELSE expression2 END;
- COALESCE (expression1, expression2, ... expression-n), for n >= 3, is equivalent to the following CASE expression:  
CASE WHEN expression1 IS NOT NULL THEN expression1  
ELSE COALESCE (expression2, . . . , expression-n) END;

## Notes

COALESCE is an ANSI standard function (SQL-92).

## Example

```
SELECT product_description, COALESCE(lowest_competitor_price,  
highest_competitor_price, average_competitor_price) AS price  
FROM product_dimension;
```

product_description	price
Brand #54109 kidney beans	264
Brand #53364 veal	139
Brand #50720 ice cream sandwiches	127
Brand #48820 coffee cake	174
Brand #48151 halibut	353
Brand #47165 canned olives	250
Brand #39509 lamb	306
Brand #36228 tuna	245
Brand #34156 blueberry muffins	183
Brand #31207 clams	163

(10 rows)

## See Also

**Case Expressions** (page 45)

**ISNULL** (page 301)

## IFNULL

Returns the value of the first non-null expression in the list.

IFNULL is an alias of **NVL** (page 304).

## Behavior Type

Immutable

**Syntax**

```
IFNULL ( expression1 , expression2 );
```

**Parameters**

- If *expression1* is null, then IFNULL returns *expression2*.
- If *expression1* is not null, then IFNULL returns *expression1*.

**Notes**

- **COALESCE** (page 299) is the more standard, more general function.
- IFNULL is equivalent to ISNULL.
- IFNULL is equivalent to COALESCE except that IFNULL is called with only two arguments.
- ISNULL(a,b) is different from x IS NULL.
- The arguments can have any data type supported by Vertica.
- Implementation is equivalent to the CASE expression. For example:  
CASE WHEN expression1 IS NULL THEN expression2 ELSE expression1 END;
- The following statement returns the value 140:  
SELECT IFNULL(NULL, 140) FROM employee\_dimension;
- The following statement returns the value 60:  
SELECT IFNULL(60, 90) FROM employee\_dimension;

**Examples**

```
=> SELECT IFNULL (SCORE, 0.0) FROM TESTING;
IFNULL
-----
 100.0
   87.0
    .0
    .0
    .0
(5 rows)
```

**See Also**

**Case Expressions** (page 45)

**COALESCE** (page 299)

**NVL** (page 304)

**ISNULL** (page 301)

**ISNULL**

Returns the value of the first non-null expression in the list.

ISNULL is an alias of **NVL** (page 304).

## Behavior Type

Immutable

## Syntax

```
ISNULL ( expression1 , expression2 );
```

## Parameters

- If *expression1* is null, then ISNULL returns *expression2*.
- If *expression1* is not null, then ISNULL returns *expression1*.

## Notes

- **COALESCE** (page 299) is the more standard, more general function.
- ISNULL is equivalent to COALESCE except that ISNULL is called with only two arguments.
- ISNULL(a, b) is different from `x IS NULL`.
- The arguments can have any data type supported by Vertica.
- Implementation is equivalent to the CASE expression. For example:  
CASE WHEN *expression1* IS NULL THEN *expression2* ELSE *expression1* END;
- The following statement returns the value 140:  
SELECT ISNULL(NULL, 140) FROM employee\_dimension;
- The following statement returns the value 60:  
SELECT ISNULL(60, 90) FROM employee\_dimension;

## Examples

```
SELECT product_description, product_price, ISNULL(product_cost, 0.0) AS cost FROM  
product_dimension;
```

product_description	product_price	cost
Brand #59957 wheat bread	405	207
Brand #59052 blueberry muffins	211	140
Brand #59004 english muffins	399	240
Brand #53222 wheat bread	323	94
Brand #52951 croissants	367	121
Brand #50658 croissants	100	94
Brand #49398 white bread	318	25
Brand #46099 wheat bread	242	3
Brand #45283 wheat bread	111	105
Brand #43503 jelly donuts	259	19

(10 rows)

## See Also

**Case Expressions** (page 45)

**COALESCE** (page 299)

**NVL** (page 304)

## NULLIF

Compares two expressions. If the expressions are not equal, the function returns the first expression (*expression1*). If the expressions are equal, the function returns null.

### Behavior Type

Immutable

### Syntax

```
NULLIF( expression1, expression2 )
```

### Parameters

<i>expression1</i>	Is a value of any data type.
<i>expression2</i>	Must have the same data type as <i>expr1</i> or a type that can be implicitly cast to match <i>expression1</i> . The result has the same type as <i>expression1</i> .

### Examples

The following series of statements illustrates one simple use of the NULLIF function.

Creates a single-column table *t* and insert some values:

```
CREATE TABLE t (x TIMESTAMPTZ);
INSERT INTO t VALUES('2009-09-04 09:14:00-04');
INSERT INTO t VALUES('2010-09-04 09:14:00-04');
```

Issue a select statement:

```
SELECT x, NULLIF(x, '2009-09-04 09:14:00 EDT') FROM t;
```

```

      x                |          nullif
-----+-----
2009-09-04 09:14:00-04 |
2010-09-04 09:14:00-04 | 2010-09-04 09:14:00-04
```

```
SELECT NULLIF(1, 2);
```

```
NULLIF
```

```
-----
          1
```

```
(1 row)
```

```
SELECT NULLIF(1, 1);
```

```
NULLIF
```

```
-----
```

```
(1 row)
```

```
SELECT NULLIF(20.45, 50.80);
```

```
NULLIF
```

```
-----
      20.45
```

```
(1 row)
```

## NULLIFZERO

Evaluates to NULL if the value in the column is 0.

### Syntax

```
NULLIFZERO(expression)
```

### Parameters

<i>expression</i>	(INTEGER, DOUBLE PRECISION, INTERVAL, or NUMERIC) Is the string to evaluate for 0 values.
-------------------	---

### Example

The TESTING table below shows the test scores for 5 students. Note that test scores are missing for S. Robinson and K. Johnson (NULL values appear in the Score column.)

```
=> select * from TESTING;
   Name      | Score
-----+-----
 J. Doe      |    100
 R. Smith    |     87
 L. White    |     0
 S. Robinson |
 K. Johnson  |
(5 rows)
```

The SELECT statement below specifies that Vertica should return any 0 values in the Score column as Null. In the results, you can see that Vertica returns L. White's 0 score as Null.

```
=> SELECT Name, NULLIFZERO(Score) FROM TESTING;

   Name      | NULLIFZERO
-----+-----
 J. Doe      |          100
 R. Smith    |           87
 L. White    |
 S. Robinson |
 K. Johnson  |
(5 rows)
```

## NVL

Returns the value of the first non-null expression in the list.

### Behavior Type

Immutable

**Syntax**

```
NVL ( expression1 , expression2 );
```

**Parameters**

- If *expression1* is null, then NVL returns *expression2*.
- If *expression1* is not null, then NVL returns *expression1*.

**Notes**

- **COALESCE** (page 299) is the more standard, more general function.
- NVL is equivalent to COALESCE except that NVL is called with only two arguments.
- The arguments can have any data type supported by Vertica.
- Implementation is equivalent to the CASE expression:  

```
CASE WHEN expression1 IS NULL THEN expression2 ELSE expression1 END;
```

**Examples**

expression1 is not null, so NVL returns expression1:

```
SELECT NVL('fast', 'database');
   nvl
-----
   fast
(1 row)
```

expression1 is null, so NVL returns expression2:

```
SELECT NVL(null, 'database');
   nvl
-----
 database
(1 row)
```

expression2 is null, so NVL returns expression1:

```
SELECT NVL('fast', null);
   nvl
-----
   fast
(1 row)
```

In the following example, expression1 (title) contains nulls, so NVL returns expression2 and substitutes 'Withheld' for the unknown values:

```
SELECT customer_name,
       NVL(title, 'Withheld') as title
FROM customer_dimension
ORDER BY title;
   customer_name      | title
-----+-----
 Alexander I. Lang    | Dr.
 Steve S. Harris      | Dr.
 Daniel R. King       | Dr.
 Luigi I. Sanchez     | Dr.
 Duncan U. Carcetti   | Dr.
```

Meghan K. Li	Dr.
Laura B. Perkins	Dr.
Samantha V. Robinson	Dr.
Joseph P. Wilson	Mr.
Kevin R. Miller	Mr.
Lauren D. Nguyen	Mrs.
Emily E. Goldberg	Mrs.
Darlene K. Harris	Ms.
Meghan J. Farmer	Ms.
Bettercare	Withheld
Ameristar	Withheld
Initech	Withheld

(17 rows)

### See Also

**Case Expressions** (page 45)

**COALESCE** (page 299)

**ISNULL** (page 301)

**NVL2** (page 306)

## NVL2

Takes three arguments. If the first argument is not NULL, it returns the second argument, otherwise it returns the third argument. The data types of the second and third arguments are implicitly cast to a common type if they don't agree, similar to **COALESCE** (page 299).

### Behavior Type

Immutable

### Syntax

```
NVL2 ( expression1 , expression2 , expression3 );
```

### Parameters

- If *expression1* is not null, then NVL2 returns *expression2*.
- If *expression1* is null, then NVL2 returns *expression3*.

### Notes

Arguments two and three can have any data type supported by Vertica.

Implementation is equivalent to the CASE expression:

```
CASE WHEN expression1 IS NOT NULL THEN expression2 ELSE expression3 END;
```

### Examples

In this example, *expression1* is not null, so NVL2 returns *expression2*:

```
SELECT NVL2('very', 'fast', 'database');  
nv12
```

```
-----
fast
(1 row)
```

In this example, expression1 is null, so NVL2 returns expression3:

```
SELECT NVL2(null, 'fast', 'database');
       nvl2
```

```
-----
database
(1 row)
```

In the following example, expression1 (title) contains nulls, so NVL2 returns expression3 ('Withheld') and also substitutes the non-null values with the expression 'Known':

```
SELECT customer_name,
       NVL2(title, 'Known', 'Withheld') as title
FROM   customer_dimension
ORDER BY title;
```

customer_name	title
Alexander I. Lang	Known
Steve S. Harris	Known
Daniel R. King	Known
Luigi I. Sanchez	Known
Duncan U. Carcetti	Known
Meghan K. Li	Known
Laura B. Perkins	Known
Samantha V. Robinson	Known
Joseph P. Wilson	Known
Kevin R. Miller	Known
Lauren D. Nguyen	Known
Emily E. Goldberg	Known
Darlene K. Harris	Known
Meghan J. Farmer	Known
Bettercare	Withheld
Ameristar	Withheld
Initech	Withheld

(17 rows)

## See Also

**Case Expressions** (page 45)

**COALESCE** (page 299)

**NVL** (page 299)

## ZEROIFNULL

Evaluates to 0 if the column is NULL.

### Syntax

```
ZEROIFNULL(expression)
```

## Parameters

<i>expression</i>	(INTEGER, DOUBLE PRECISION, INTERVAL, or NUMERIC) Is the string to evaluate for NULL values.
-------------------	--

## Example

The TESTING table below shows the test scores for 5 students. Note that L. White's score is 0, and that scores are missing for S. Robinson and K. Johnson.

```
=> select * from TESTING;
      Name      | Score
-----+-----
 J. Doe        |    100
 R. Smith      |     87
 L. White      |     0
 S. Robinson   |
 K. Johnson    |
(5 rows)
```

The SELECT statement below specifies that Vertica should return any Null values in the Score column as 0s. In the results, you can see that Vertica returns a 0 score for S. Robinson and K. Johnson.

```
=> SELECT Name, ZEROIFNULL (Score) FROM TESTING;
      Name      | ZEROIFNULL
-----+-----
 J. Doe        |    100
 R. Smith      |     87
 L. White      |     0
 S. Robinson   |     0
 K. Johnson    |     0
(5 rows)
```

## Sequence Functions

The sequence functions provide simple, multiuser-safe methods for obtaining successive sequence values from sequence objects.

### NEXTVAL

Advances the return of a new sequence value. A positive value is incremented for ascending sequences and a negative value is decremented for descending sequences.

### Behavior Type

Volatile

## Syntax

```
<sequence_name>.NEXTVAL
NEXTVAL('sequence_name')
```

## Parameters

<i>sequence_name</i>	Identifies the sequence for which to determine the next value.
----------------------	--

## Notes

- NEXTVAL is used in INSERT, COPY, and SELECT statements to create unique values.
- The first time NEXTVAL is called, it generates the starting number for the sequence. Thereafter, it increments this number.
- While executing a SQL statement, if NEXTVAL is called on two different nodes, each node creates and maintains its own cache of values per session. Thus, you need a Global Catalog Lock (X) to obtain a cache of values from a sequence.
- NEXTVAL is evaluated on a per-row basis. Thus, in the following example, both calls to NEXTVAL yield same result:

```
SELECT NEXTVAL('seq1'), NEXTVAL('seq1') FROM vendor_key;
```

## Examples

The following example creates an ascending sequence called my\_seq, starting at 101:

```
CREATE SEQUENCE sequential START 101;
```

The following command generates the first number in the sequence:

```
SELECT NEXTVAL('my_seq');
 nextval
-----
      101
(1 row)
```

The following command generates the next number in the sequence:

```
SELECT NEXTVAL('my_seq');
 nextval
-----
      102
(1 row)
```

The following example shows how to use NEXTVAL in a table SELECT statement. Notice that the nextval column incremented by (1) again:

```
SELECT NEXTVAL('my_seq'), lname FROM customer;
 nextval | lname
-----+-----
      103 | Carr
(1 row)
```

## See Also

**ALTER SEQUENCE** (page 586)

**CREATE SEQUENCE** (page 658)

**CURRVAL** (page 310)

**DROP SEQUENCE** (page 708)

Using Sequences and Sequence Privileges in the Administrator's Guide

## CURRVAL

For a sequence generator, returns the LAST value across all nodes returned by a previous invocation of **NEXTVAL** (page 308) in the same session. If there were no calls to NEXTVAL, an error is returned.

### Behavior Type

Volatile

### Syntax

```
<sequence_name>.CURRVAL
```

### Parameters

<i>sequence_name</i>	Identifies the sequence for which to return the current value.
----------------------	--

### Notes

NEXTVAL is executed before anything else. Therefore, the following statement succeeds even though CURRVAL appears before NEXTVAL in the statement:

```
SELECT CURRVAL('seq1'), NEXTVAL('seq1') FROM vendor_key;
```

### Examples

The following example creates an ascending sequence called sequential, starting at 101:

```
CREATE SEQUENCE seq2 START 101;
```

You cannot call CURRVAL until after you have initiated the sequence with NEXTVAL or the system returns an error:

```
SELECT CURRVAL('seq2');  
ERROR: Sequence seq2 has not been accessed in the session
```

Use the NEXTVAL function to generate the first number for this sequence:

```
SELECT NEXTVAL('seq2');  
nextval  
-----  
101  
(1 row)
```

Now you can use CURRVAL to return the current number from this sequence:

```
SELECT CURRVAL('seq2');  
currval  
-----  
101
```

```
(1 row)
```

The following command shows how to use CURRVAL in a SELECT statement:

```
CREATE TABLE customer3 (
  lname VARCHAR(25),
  fname VARCHAR(25),
  membership_card INTEGER,
  ID INTEGER
);
INSERT INTO customer3 VALUES ('Brown' , 'Sabra', 072753, CURRVAL('my_seq'));
SELECT CURRVAL('seq2'), lname FROM customer3;
  CURRVAL | lname
-----+-----
        101 | Brown
(1 row)
```

### See Also

**ALTER SEQUENCE** (page 586)

**CREATE SEQUENCE** (page 658)

**DROP SEQUENCE** (page 708)

**NEXTVAL** (page 308)

Using Sequences and Sequence Privileges in the Administrator's Guide

## LAST\_INSERT\_ID

Returns the last value of a column whose value is automatically incremented through the AUTO\_INCREMENT or IDENTITY **column-constraint** (page 674). If multiple sessions concurrently load the same table, the returned value is the last value generated for an AUTO\_INCREMENT column by an insert in that session.

### Behavior Type

Volatile

### Syntax

```
LAST_INSERT_ID()
```

### Notes

- This function works only with AUTO\_INCREMENT and IDENTITY columns. See **column-constraints** (page 674) for the **CREATE TABLE** (page 664) statement.
- LAST\_INSERT\_ID does not work with sequence generators created through the **CREATE SEQUENCE** (page 658) statement.

### Examples

Create a sample table called `customer4`.

```
=> CREATE TABLE customer4(
  ID IDENTITY(2,2),
  lname VARCHAR(25),
```

```

        fname VARCHAR(25),
        membership_card INTEGER
    );
=> INSERT INTO customer4(lname, fname, membership_card)
    VALUES ('Gupta', 'Saleem', 475987);

```

Notice that the **IDENTITY** column has a seed of 2, which specifies the value for the first row loaded into the table, and an increment of 2, which specifies the value that is added to the **IDENTITY** value of the previous row.

Query the table you just created:

```

=> SELECT * FROM customer4;
  ID | lname | fname | membership_card
-----+-----+-----+-----
   2 | Gupta | Saleem |          475987
(1 row)

```

Insert some additional values:

```

=> INSERT INTO customer4(lname, fname, membership_card)
    VALUES ('Lee', 'Chen', 598742);

```

Call the **LAST\_INSERT\_ID** function:

```

=> SELECT LAST_INSERT_ID();
last_insert_id
-----
                4
(1 row)

```

Query the table again:

```

=> SELECT * FROM customer4;
  ID | lname | fname | membership_card
-----+-----+-----+-----
   2 | Gupta | Saleem |          475987
   4 | Lee   | Chen   |          598742
(2 rows)

```

Add another row:

```

=> INSERT INTO customer4(lname, fname, membership_card)
    VALUES ('Davis', 'Bill', 469543);

```

Call the **LAST\_INSERT\_ID** function:

```

=> SELECT LAST_INSERT_ID();
LAST_INSERT_ID
-----
                6
(1 row)

```

Query the table again:

```

=> SELECT * FROM customer4;
  ID | lname | fname | membership_card
-----+-----+-----+-----
   2 | Gupta | Saleem |          475987
   4 | Lee   | Chen   |          598742

```

```
6 | Davis | Bill | 469543  
(3 rows)
```

**See Also*****ALTER SEQUENCE*** (page 586)***CREATE SEQUENCE*** (page 658)***DROP SEQUENCE*** (page 708)***V\_CATALOG.SEQUENCES*** (page 815)

Using Sequences and Sequence Privileges in the Administrator's Guide

## String Functions

String functions perform conversion, extraction, or manipulation operations on strings, or return information about strings.

This section describes functions and operators for examining and manipulating string values. Strings in this context include values of the types CHAR, VARCHAR, BINARY, and VARBINARY.

Unless otherwise noted, all of the functions listed in this section work on all four data types. As opposed to some other SQL implementations, Vertica keeps CHAR strings unpadded internally, padding them only on final output. So converting a CHAR(3) 'ab' to VARCHAR(5) results in a VARCHAR of length 2, not one with length 3 including a trailing space.

Some of the functions described here also work on data of non-string types by converting that data to a string representation first. Some functions work only on character strings, while others work only on binary strings. Many work for both. BINARY and VARBINARY functions ignore multibyte UTF-8 character boundaries.

Non-binary character string functions handle normalized multibyte UTF-8 characters, as specified by the Unicode Consortium. Unless otherwise specified, those character string functions for which it matters can optionally specify whether VARCHAR arguments should be interpreted as octet (byte) sequences, or as (locale-aware) sequences of UTF-8 characters. This is accomplished by adding "USING OCTETS" or "USING CHARACTERS" (default) as a parameter to the function.

Some character string functions are stable because in general UTF-8 case-conversion, searching and sorting can be locale dependent. Thus, LOWER is stable, while LOWERB is immutable. The USING OCTETS clause converts these functions into their "B" forms, so they become immutable. If the locale is set to collation=binary, which is the default, all string functions — except CHAR\_LENGTH/CHARACTER\_LENGTH, LENGTH, SUBSTR, and OVERLAY — are converted to their "B" forms and so are immutable.

BINARY implicitly converts to VARBINARY, so functions that take VARBINARY arguments work with BINARY.

### ASCII

Converts the first octet of a VARCHAR to an INTEGER.

#### Behavior Type

Immutable

#### Syntax

```
ASCII ( expression )
```

#### Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

## Notes

- ASCII is the opposite of the **CHR** (page 318) function.
- ASCII operates on UTF-8 characters, not only on single-byte ASCII characters. It continues to get the same results for the ASCII subset of UTF-8.

## Examples

Expression	Result
SELECT ASCII('A');	65
SELECT ASCII('ab');	97
SELECT ASCII(null);	
SELECT ASCII('');	

## BIT\_LENGTH

Returns the length of the string expression in bits (bytes \* 8) as an INTEGER.

### Behavior Type

Immutable

### Syntax

```
BIT_LENGTH ( expression )
```

### Parameters

<i>expression</i>	(CHAR or VARCHAR or BINARY or VARBINARY) is the string to convert.
-------------------	--

## Notes

BIT\_LENGTH applies to the contents of VARCHAR and VARBINARY fields.

## Examples

Expression	Result
SELECT BIT_LENGTH('abc'::varbinary);	24
SELECT BIT_LENGTH('abc'::binary);	8
SELECT BIT_LENGTH(' '::varbinary);	0
SELECT BIT_LENGTH(' '::binary);	8
SELECT BIT_LENGTH(null::varbinary);	
SELECT BIT_LENGTH(null::binary);	
SELECT BIT_LENGTH(VARCHAR 'abc');	24
SELECT BIT_LENGTH(CHAR 'abc');	24
SELECT BIT_LENGTH(CHAR(6) 'abc');	48
SELECT BIT_LENGTH(VARCHAR(6) 'abc');	24

```
SELECT BIT_LENGTH(BINARY(6) 'abc');           48
SELECT BIT_LENGTH(BINARY 'abc');             24
SELECT BIT_LENGTH(VARBINARY 'abc');          24
SELECT BIT_LENGTH(VARBINARY(6) 'abc');       24
```

### See Also

**CHARACTER\_LENGTH** (page 318), **LENGTH** (page 335), **OCTET\_LENGTH** (page 339)

## BITCOUNT

Returns the number of one-bits (sometimes referred to as set-bits) in the given VARBINARY value. This is also referred to as the population count.

### Behavior Type

Immutable

### Syntax

```
BITCOUNT ( expression )
```

### Parameters

<i>expression</i>	(BINARY or VARBINARY) is the string to return.
-------------------	--

### Examples

```
SELECT BITCOUNT(HEX_TO_BINARY('0x10'));
  bitcount
-----
         1
(1 row)
SELECT BITCOUNT(HEX_TO_BINARY('0xF0'));
  bitcount
-----
         4
(1 row)
SELECT BITCOUNT(HEX_TO_BINARY('0xAB'));
  bitcount
-----
         5
(1 row)
```

## BITSTRING\_TO\_BINARY

Translates the given VARCHAR bitstring representation into a VARBINARY value.

### Behavior Type

Immutable

**Syntax**

```
BITSTRING_TO_BINARY ( expression )
```

**Parameters**

<i>expression</i>	(VARCHAR) is the string to return.
-------------------	------------------------------------

**Notes**

VARBINARY BITSTRING\_TO\_BINARY(VARCHAR) converts data from character type (in bitstring format) to binary type. This function is the inverse of TO\_BITSTRING.

```
BITSTRING_TO_BINARY(TO_BITSTRING(x)) = x
TO_BITSTRING(BITSTRING_TO_BINARY(x)) = x
```

**Examples**

If there are an odd number of characters in the hex value, then the first character is treated as the low nibble of the first (furthest to the left) byte.

```
SELECT BITSTRING_TO_BINARY('0110000101100010');
   bitstring_to_binary
-----
   ab
(1 row)
```

If an invalid bitstring is supplied, the system returns an error:

```
SELECT BITSTRING_TO_BINARY('010102010');
ERROR:  invalid bitstring "010102010"
```

**BTRIM**

Removes the longest string consisting only of specified characters from the start and end of a string.

**Behavior Type**

Immutable

**Syntax**

```
BTRIM ( expression [ , characters-to-remove ] )
```

**Parameters**

<i>expression</i>	(CHAR or VARCHAR) is the string to modify
<i>characters-to-remove</i>	(CHAR or VARCHAR) specifies the characters to remove. The default is the space character.

**Examples**

```
SELECT BTRIM('yxtrimyyx', 'xy');
   btrim
-----
   trim
```

(1 row)

### See Also

**LTRIM** (page 338), **RTRIM** (page 348), **TRIM** (page 358)

## CHARACTER\_LENGTH

Returns an INTEGER value representing the number of characters or octets in a string. It strips the padding from CHAR expressions but not from VARCHAR expressions.

### Behavior Type

Immutable if USING OCTETS, stable otherwise.

### Syntax

```
[ CHAR_LENGTH | CHARACTER_LENGTH ] ( expression ,
... [ USING { CHARACTERS | OCTETS } ] )
```

### Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to measure
USING CHARACTERS   OCTETS	Determines whether the character length is expressed in characters (the default) or octets.

### Notes

CHARACTER\_LENGTH is identical to **LENGTH** (page 335). See **BIT\_LENGTH** (page 315) and **OCTET\_LENGTH** (page 339) for similar functions.

### Examples

```
SELECT CHAR_LENGTH('1234 '::CHAR(10), USING OCTETS);
 char_length
-----
          4
(1 row)
SELECT CHAR_LENGTH('1234 '::VARCHAR(10));
 char_length
-----
          6
(1 row)
SELECT CHAR_LENGTH(NULL::CHAR(10)) IS NULL;
 ?column?
-----
 t
(1 row)
```

## CHR

Converts the first octet of an INTEGER to a VARCHAR.

**Behavior Type**

Immutable

**Syntax**

```
CHR ( expression )
```

**Parameters**

<i>expression</i>	(INTEGER) is the string to convert and is masked to a single octet.
-------------------	---

**Notes**

- CHR is the opposite of the **ASCII** (page 314) function.
- CHR operates on UTF-8 characters, not only on single-byte ASCII characters. It continues to get the same results for the ASCII subset of UTF-8.

**Examples**

Expression	Result
------------	--------

SELECT CHR(65);	A
SELECT CHR(65+32);	a
SELECT CHR(null);	

**CONCAT**

Used to concatenate two or more VARBINARY strings. The return value is of type VARBINARY.

**Syntax**

```
CONCAT ('a', 'b')
```

**Behavior type**

Immutable

**Parameters**

a	Is the first VARBINARY string.
b	Is the second VARBINARY string.

**Examples**

```
=> SELECT CONCAT ('A', 'B');
      CONCAT
-----
```

## DECODE

Compares *expression* to each search value one by one. If *expression* is equal to a search, the function returns the corresponding result. If no match is found, the function returns default. If default is omitted, the function returns null.

### Behavior Type

Immutable

### Syntax

```
DECODE ( expression, search, result [ , search, result ]  
...[, default ] );
```

### Parameters

<i>expression</i>	Is the value to compare.
<i>search</i>	Is the value compared against <i>expression</i> .
<i>result</i>	Is the value returned, if <i>expression</i> is equal to search.
<i>default</i>	Is optional. If no matches are found, DECODE returns default. If default is omitted, then DECODE returns NULL (if no matches are found).

### Notes

DECODE is similar to the IF-THEN-ELSE and **CASE** (page 45) expression:

```
CASE expression  
WHEN search THEN result  
[WHEN search THEN result]  
[ELSE default];
```

The arguments can have any data type supported by Vertica. The result types of individual results are promoted to the least common type that can be used to represent all of them. This leads to a character string type, an exact numeric type, an approximate numeric type, or a DATETIME type, where all the various result arguments must be of the same type grouping.

### Examples

The following example converts numeric values in the weight column from the product\_dimension table to descriptive values in the output.

```
SELECT product_description, DECODE(weight,  
    2, 'Light',  
    50, 'Medium',  
    71, 'Heavy',  
    99, 'Call for help',  
    'N/A')  
FROM product_dimension  
WHERE category_description = 'Food'  
AND department_description = 'Canned Goods'  
AND sku_number BETWEEN 'SKU-#49750' AND 'SKU-#49999'
```

```
LIMIT 15;
```

product_description	case
Brand #499 canned corn	N/A
Brand #49900 fruit cocktail	Medium
Brand #49837 canned tomatoes	Heavy
Brand #49782 canned peaches	N/A
Brand #49805 chicken noodle soup	N/A
Brand #49944 canned chicken broth	N/A
Brand #49819 canned chili	N/A
Brand #49848 baked beans	N/A
Brand #49989 minestrone soup	N/A
Brand #49778 canned peaches	N/A
Brand #49770 canned peaches	N/A
Brand #4977 fruit cocktail	N/A
Brand #49933 canned olives	N/A
Brand #49750 canned olives	Call for help
Brand #49777 canned tomatoes	N/A

(15 rows)

## GREATEST

Returns the largest value in a list of expressions.

### Behavior Type

Stable

### Syntax

```
GREATEST ( expression1, expression2, ... expression-n );
```

### Parameters

*expression1*, *expression2*, and *expression-n* are the expressions to be evaluated.

### Notes

- Works for all data types, and implicitly casts similar types. See Examples.
- A NULL value in any one of the expressions returns NULL.
- Depends on the collation setting of the locale.

### Examples

This example returns 9 as the greatest in the list of expressions:

```
SELECT GREATEST(7, 5, 9);
   greatest
-----
          9
(1 row)
```

Note that putting quotes around the integer expressions returns the same result as the first example:

```
SELECT GREATEST('7', '5', '9');
   greatest
```

```
-----  
          9  
(1 row)
```

The next example returns FLOAT 1.5 as the greatest because the integer is implicitly cast to float:

```
SELECT GREATEST(1, 1.5);  
greatest  
-----  
          1.5  
(1 row)
```

The following example returns 'vertica' as the greatest:

```
SELECT GREATEST('vertica', 'analytic', 'database');  
greatest  
-----  
vertica  
(1 row)
```

Notice this next command returns NULL:

```
SELECT GREATEST('vertica', 'analytic', 'database', null);  
greatest  
-----  
  
(1 row)
```

And one more:

```
SELECT GREATEST('sit', 'site', 'sight');  
greatest  
-----  
site  
(1 row)
```

### See Also

**LEAST** (page 332)

## GREATESTB

Returns its greatest argument, using binary ordering, not UTF-8 character ordering.

### Behavior Type

Immutable

### Syntax

```
GREATESTB ( expression1, expression2, ... expression-n );
```

### Parameters

*expression1*, *expression2*, and *expression-n* are the expressions to be evaluated.

### Notes

- Works for all data types, and implicitly casts similar types. See Examples.

- A NULL value in any one of the expressions returns NULL.
- Depends on the collation setting of the locale.

### Examples

The following command selects `straße` as the greatest in the series of inputs:

```
SELECT GREATESTB('straße', 'strasse');
GREATESTB
-----
straße
(1 row)
```

This example returns 9 as the greatest in the list of expressions:

```
SELECT GREATESTB(7, 5, 9);
GREATESTB
-----
          9
(1 row)
```

Note that putting quotes around the integer expressions returns the same result as the first example:

```
GREATESTB
-----
          9
(1 row)
```

The next example returns `FLOAT 1.5` as the greatest because the integer is implicitly cast to float:

```
SELECT GREATESTB(1, 1.5);
GREATESTB
-----
          1.5
(1 row)
```

The following example returns `'vertica'` as the greatest:

```
SELECT GREATESTB('vertica', 'analytic', 'database');
GREATESTB
-----
vertica
(1 row)
```

Notice this next command returns `NULL`:

```
SELECT GREATESTB('vertica', 'analytic', 'database', null);
GREATESTB
-----

(1 row)
```

And one more:

```
SELECT GREATESTB('sit', 'site', 'sight');
GREATESTB
-----
site
(1 row)
```

**See Also**

**LEASTB** (page 333)

**HEX\_TO\_BINARY**

Translates the given VARCHAR hexadecimal representation into a VARBINARY value.

**Behavior Type**

Immutable

**Syntax**

```
HEX_TO_BINARY ( [ 0x ] expression )
```

**Parameters**

<i>expression</i>	(BINARY or VARBINARY) is the string to translate.
0x	Is optional prefix

**Notes**

VARBINARY HEX\_TO\_BINARY(VARCHAR) converts data from character type in hexadecimal format to binary type. This function is the inverse of **TO\_HEX** (page 244).

```
HEX_TO_BINARY(TO_HEX(x)) = x
```

```
TO_HEX(HEX_TO_BINARY(x)) = x
```

If there are an odd number of characters in the hexadecimal value, the first character is treated as the low nibble of the first (furthest to the left) byte.

**Examples**

If the given string begins with "0x" the prefix is ignored. For example:

```
SELECT HEX_TO_BINARY('0x6162') AS hex1, HEX_TO_BINARY('6162') AS hex2;
  hex1 | hex2
-----+-----
  ab   | ab
(1 row)
```

If an invalid hex value is given, Vertica returns an "invalid binary representation" error; for example:

```
SELECT HEX_TO_BINARY('0xffgf');
ERROR:  invalid hex string "0xffgf"
```

**See Also**

**TO\_HEX** (page 244)

**INET\_ATON**

Returns an integer that represents the value of the address in host byte order, given the dotted-quad representation of a network address as a string.

## Behavior Type

Immutable

## Syntax

```
INET_ATON ( expression )
```

## Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

## Notes

The following syntax converts an IPv4 address represented as the string A to an integer I.

INET\_ATON trims any spaces from the right of A, calls the Linux function *inet\_pton* [http://www.opengroup.org/onlinepubs/000095399/functions/inet\\_pton.html](http://www.opengroup.org/onlinepubs/000095399/functions/inet_pton.html), and converts the result from network byte order to host byte order using *ntohl* <http://opengroup.org/onlinepubs/007908775/xns/ntohl.html>.

```
INET_ATON(VARCHAR A) -> INT8 I
```

If A is NULL, too long, or *inet\_pton* returns an error, the result is NULL.

## Examples

The generated number is always in host byte order. In the following example, the number is calculated as  $209 \times 256^3 + 207 \times 256^2 + 224 \times 256 + 40$ .

```
SELECT INET_ATON('209.207.224.40');
inet_aton
-----
3520061480
(1 row)
SELECT INET_ATON('1.2.3.4');
inet_aton
-----
16909060
(1 row)
SELECT TO_HEX(INET_ATON('1.2.3.4'));
to_hex
-----
1020304
(1 row)
```

## See Also

**INET\_NTOA** (page 273)

## INET\_NTOA

Returns the dotted-quad representation of the address as a VARCHAR, given a network address as an integer in network byte order.

## Behavior Type

Immutable

## Syntax

```
INET_NTOA ( expression )
```

## Parameters

<i>expression</i>	(INTEGER) is the network address to convert.
-------------------	--

## Notes

The following syntax converts an IPv4 address represented as integer I to a string A.

INET\_NTOA converts I from host byte order to network byte order using *htonl* <http://opengroup.org/onlinepubs/007908775/xns/htonl.html>, and calls the Linux function *inet\_ntop* [http://www.opengroup.org/onlinepubs/000095399/functions/inet\\_ntop.html](http://www.opengroup.org/onlinepubs/000095399/functions/inet_ntop.html).

```
INET_NTOA(INT8 I) -> VARCHAR A
```

If I is NULL, greater than 2<sup>32</sup> or negative, the result is NULL.

## Examples

```
SELECT INET_NTOA(16909060);
  inet_ntoa
-----
  1.2.3.4
(1 row)
SELECT INET_NTOA(03021962);
  inet_ntoa
-----
  0.46.28.138
(1 row)
```

## See Also

**INET\_ATON** (page 272)

## INITCAP

Capitalizes first letter of each alphanumeric word and puts the rest in lowercase.

## Behavior Type

Stable

## Syntax

```
INITCAP ( expression )
```

## Parameters

<i>expression</i>	(VARCHAR) is the string to format.
-------------------	------------------------------------

## Notes

- Depends on collation setting of the locale.
- INITCAP is restricted to 32750 octet inputs, since it is possible for the UTF-8 representation of result to double in size.

## Examples

### Expression

```
SELECT INITCAP('high speed database');
SELECT INITCAP('LINUX TUTORIAL');
SELECT INITCAP('abc DEF 123aVc 124Btd,lAsT');

SELECT INITCAP('');
SELECT INITCAP(null);
```

### Result

```
High Speed Database
Linux Tutorial
Abc Def 123Avc
124Btd,Last
```

## INITCAPB

Capitalizes first letter of each alphanumeric word and puts the rest in lowercase.

### Behavior Type

Immutable

### Syntax

```
INITCAPB ( expression )
```

### Parameters

<i>expression</i>	(VARCHAR) is the string to format.
-------------------	------------------------------------

## Notes

Depends on collation setting of the locale.

## Examples

### Expression

```
SELECT INITCAPB('étudiant');
SELECT INITCAPB('high speed database');
SELECT INITCAPB('LINUX TUTORIAL');
SELECT INITCAPB('abc DEF 123aVc 124Btd,lAsT');

SELECT INITCAPB('');
SELECT INITCAPB(null);
```

### Result

```
éTudiant
High Speed Database
Linux Tutorial
Abc Def 123Avc
124Btd,Last
```

## INSERT

Inserts a character string into a specified location in another character string.

### Syntax

```
INSERT( 'string1', n, m, 'string2');
```

### Behavior type

Immutable

### Parameters

<i>string1</i>	(VARCHAR) Is the string in which to insert the new string.
n	A character of type INTEGER that represents the starting point for the insertion within <i>string1</i> . You specify the number of characters from the first character in <i>string1</i> as the starting point for the insertion. For example, to insert characters before "c", in the string "abcdef," enter 3.
m	A character of type INTEGER that represents the the number of characters in <i>string1</i> (if any) that should be replaced by the insertion. For example, if you want the insertion to replace the letters "cd" in the string "abcdef, " enter 2.
<i>string2</i>	(VARCHAR) Is the string to be inserted.

### Example

The following example changes the string Warehouse to Storehouse using the INSERT function:

```
=> SELECT INSERT ('Warehouse',1,3,'Stor');
      INSERT
-----
Storehouse
(1 row)
```

## INSTR

Searches *string* for *substring* and returns an integer indicating the position of the character in *string* that is the first character of this *occurrence*. The return value is based on the character position of the identified character.

### Behavior Type

Stable

### Syntax

```
INSTR ( string , substring [, position [, occurrence ] ] )
```

## Parameters

<i>string</i>	(CHAR or VARCHAR, or BINARY or VARBINARY) Is the text expression to search.
<i>substring</i>	(CHAR or VARCHAR, or BINARY or VARBINARY) Is the string to search for.
<i>position</i>	Is a nonzero integer indicating the character of string where Vertica begins the search. If position is negative, then Vertica counts backward from the end of string and then searches backward from the resulting position. The first character of string occupies the default position 1, and position cannot be 0.
<i>occurrence</i>	Is an integer indicating which occurrence of string Vertica searches. The value of occurrence must be positive (greater than 0), and the default is 1.

## Notes

Both *position* and *occurrence* must be of types that can resolve to an integer. The default values of both parameters are 1, meaning Vertica begins searching at the first character of string for the first occurrence of substring. The return value is relative to the beginning of string, regardless of the value of position, and is expressed in characters.

If the search is unsuccessful (that is, if substring does not appear *occurrence* times after the *position* character of *string*, then the return value is 0.

## Examples

The first example searches forward in string 'abc' for substring 'b'. The search returns the position in 'abc' where 'b' occurs, or position 2. Because no position parameters are given, the default search starts at 'a', position 1.

```
SELECT INSTR('abc', 'b');
INSTR
-----
      2
(1 row)
```

The following three examples use character position to search backward to find the position of a substring.

**Note:** Although it seems intuitive that the function returns a negative integer, the position of *n* occurrence is read left to right in the string, even though the search happens in reverse (from the end — or right side — of the string).

In the first example, the function counts backward one character from the end of the string, starting with character 'c'. The function then searches backward for the first occurrence of 'a', which it finds it in the first position in the search string.

```
SELECT INSTR('abc', 'a', -1);
INSTR
-----
      1
(1 row)
```

In the second example, the function counts backward one byte from the end of the string, starting with character 'c'. The function then searches backward for the first occurrence of 'a', which it finds it in the first position in the search string.

```
SELECT INSTR(VARBINARY 'abc', VARBINARY 'a', -1);
INSTR
-----
      1
(1 row)
```

In the third example, the function counts backward one character from the end of the string, starting with character 'b', and searches backward for substring 'bc', which it finds in the second position of the search string.

```
SELECT INSTR('abcb', 'bc', -1);
INSTR
-----
      2
(1 row)
```

In the fourth example, the function counts backward one character from the end of the string, starting with character 'b', and searches backward for substring 'bcef', which it does not find. The result is 0.

```
SELECT INSTR('abcb', 'bcef', -1);
INSTR
-----
      0
(1 row)
```

In the fifth example, the function counts backward one byte from the end of the string, starting with character 'b', and searches backward for substring 'bcef', which it does not find. The result is 0.

```
SELECT INSTR(VARBINARY 'abcb', VARBINARY 'bcef', -1);
INSTR
-----
      0
(1 row)
```

## INSTRB

Searches *string* for *substring* and returns an integer indicating the octet position within string that is the first *occurrence*. The return value is based on the octet position of the identified byte.

### Behavior Type

Immutable

### Syntax

```
INSTRB ( string , substring [, position [, occurrence ] ] )
```

### Parameters

<i>string</i>	Is the text expression to search.
---------------	-----------------------------------

<i>substring</i>	Is the string to search for.
<i>position</i>	Is a nonzero integer indicating the character of string where Vertica begins the search. If position is negative, then Vertica counts backward from the end of string and then searches backward from the resulting position. The first byte of string occupies the default position 1, and position cannot be 0.
<i>occurrence</i>	Is an integer indicating which occurrence of string Vertica searches. The value of occurrence must be positive (greater than 0), and the default is 1.

## Notes

Both *position* and *occurrence* must be of types that can resolve to an integer. The default values of both parameters are 1, meaning Vertica begins searching at the first byte of string for the first occurrence of substring. The return value is relative to the beginning of string, regardless of the value of position, and is expressed in octets.

If the search is unsuccessful (that is, if substring does not appear *occurrence* times after the *position* character of *string*, then the return value is 0.

## Examples

```
SELECT INSTRB('straße', 'ß');
   INSTRB
-----
          5
(1 row)
```

## See Also

***INSTR*** (page 328)

## ISUTF8

Tests whether a string is a valid UTF-8 string. Returns true if the string conforms to UTF-8 standards, and false otherwise. This function is useful to test strings for UTF-8 compliance before passing them to one of the regular expression functions, such as ***REGEXP\_LIKE*** (page 452), which expect UTF-8 characters by default.

## Syntax

```
ISUTF8( string );
```

## Parameters

<i>string</i>	The string to test for UTF-8 compliance.
---------------	--

## Examples

```
=> SELECT ISUTF8(E'\xc2\xbf'); -- UTF-8 INVERTED QUESTION MARK
   ISUTF8
-----
          t
(1 row)
```

```
=> SELECT ISUTF8(E'\xc2\xc0'); -- UNDEFINED UTF-8 CHARACTER
ISUTF8
-----
f
(1 row)
```

## LEAST

Returns the smallest value in a list of expressions.

### Behavior Type

Stable

### Syntax

```
LEAST ( expression1, expression2, ... expression-n );
```

### Parameters

*expression1*, *expression2*, and *expression-n* are the expressions to be evaluated.

### Notes

- Works for all data types, and implicitly casts similar types. See Examples below.
- A NULL value in any one of the expressions returns NULL.

### Examples

This example returns 5 as the least:

```
SELECT LEAST(7, 5, 9);
least
-----
5
(1 row)
```

Note that putting quotes around the integer expressions returns the same result as the first example:

```
SELECT LEAST('7', '5', '9');
least
-----
5
(1 row)
```

In the above example, the values are being compared as strings, so '10' would be less than '2'.

The next example returns 1.5, as INTEGER 2 is implicitly cast to FLOAT:

```
SELECT LEAST(2, 1.5);
least
-----
1.5
(1 row)
```

The following example returns 'analytic' as the least:

```
SELECT LEAST('vertica', 'analytic', 'database');
   least
-----
 analytic
(1 row)
```

Notice this next command returns NULL:

```
SELECT LEAST('vertica', 'analytic', 'database', null);
   least
-----

(1 row)
```

And one more:

```
SELECT LEAST('sit', 'site', 'sight');
   least
-----
  sight
(1 row)
```

### See Also

***GREATEST*** (page 321)

## LEASTB

Returns the function's least argument, using binary ordering, not UTF-8 character ordering.

### Behavior Type

Immutable

### Syntax

```
LEASTB ( expression1, expression2, ... expression-n );
```

### Parameters

*expression1*, *expression2*, and *expression-n* are the expressions to be evaluated.

### Notes

- Works for all data types, and implicitly casts similar types. See Examples below.
- A NULL value in any one of the expressions returns NULL.

### Examples

The following command selects strasse as the least in the series of inputs:

```
SELECT LEASTB('straße', 'strasse');
   LEASTB
-----
  strasse
(1 row)
```

This example returns 5 as the least:

```
SELECT LEASTB(7, 5, 9);
   LEASTB
-----
         5
(1 row)
```

Note that putting quotes around the integer expressions returns the same result as the first example:

```
SELECT LEASTB('7', '5', '9');
   LEASTB
-----
         5
(1 row)
```

In the above example, the values are being compared as strings, so '10' would be less than '2'.

The next example returns 1.5, as INTEGER 2 is implicitly cast to FLOAT:

```
SELECT LEASTB(2, 1.5);
   LEASTB
-----
         1.5
(1 row)
```

The following example returns 'analytic' as the least in the series of inputs:

```
SELECT LEASTB('vertica', 'analytic', 'database');
   LEASTB
-----
analytic
(1 row)
```

Notice this next command returns NULL:

```
SELECT LEASTB('vertica', 'analytic', 'database', null);
   LEASTB
-----

(1 row)
```

### See Also

***GREATESTB*** (page 322)

## LEFT

Returns the specified characters from the left side of a string.

### Behavior Type

Immutable

### Syntax

```
LEFT ( string , length )
```

## Parameters

<i>string</i>	(CHAR or VARCHAR) is the string to return.
<i>length</i>	Is an INTEGER value that specifies the count of characters to return.

## Examples

```
SELECT LEFT('vertica', 3);
```

```
left
-----
ver
(1 row)
```

```
SELECT LEFT('straße', 5);
```

```
LEFT
-----
straß
(1 row)
```

## See Also

***SUBSTR*** (page 353)

## LENGTH

Takes one argument as an input and returns an INTEGER value representing the number of characters in a string.

## Behavior Type

Immutable

## Syntax

```
LENGTH ( expression )
```

## Parameters

<i>expression</i>	(CHAR or VARCHAR or BINARY or VARBINARY) is the string to measure
-------------------	---

## Notes

- LENGTH strips the padding from CHAR expressions but not from VARCHAR expressions.
- LENGTH is identical to ***CHARACTER\_LENGTH*** (page 318) for CHAR and VARCHAR. For binary types, it is identical to octet length. See ***BIT\_LENGTH*** (page 315) and ***OCTET\_LENGTH*** (page 339) for similar functions.

## Examples

Expression	Result
SELECT LENGTH('1234 ' ::CHAR(10));	4

```
SELECT LENGTH('1234 '::VARCHAR(10));          6
SELECT LENGTH('1234 '::BINARY(10));           10
SELECT LENGTH('1234 '::VARBINARY(10));        6
SELECT LENGTH(NULL::CHAR(10)) IS NULL;        t
```

## LOWER

Returns a VARCHAR value containing the argument converted to lowercase letters.

### Behavior Type

Stable

### Syntax

```
LOWER ( expression )
```

### Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to convert
-------------------	--

### Notes

LOWER is restricted to 32750 octet inputs, since it is possible for the UTF-8 representation of result to double in size.

### Examples

```
SELECT LOWER('AbCdEfG');
  lower
-----
abcdefg
(1 row)
SELECT LOWER('The Cat In The Hat');
  lower
-----
the cat in the hat
(1 row)
SELECT LOWER('ÉTUDIANT');
  LOWER
-----
Étudiant
(1 row)
```

## LOWERRB

Returns a character string with each ASCII character converted to lowercase; multibyte UTF-8 characters are not converted.

### Behavior Type

Immutable

**Syntax**

```
LOWERB ( expression )
```

**Parameters**

<i>expression</i>	(CHAR or VARCHAR) is the string to convert
-------------------	--

**Examples**

In the following example, the multibyte UTF-8 character É is not converted to lowercase:

```
SELECT LOWERB('ÉTUDIANT');
      LOWERB
-----
Étudiant
(1 row)
SELECT LOWERB('AbCdEfG');
      LOWERB
-----
abcdefg
(1 row)
SELECT LOWERB('The Vertica Database');
      LOWERB
-----
the vertica database
(1 row)
```

**LPAD**

Returns a VARCHAR value representing a string of a specific length filled on the left with specific characters.

**Behavior Type**

Immutable

**Syntax**

```
LPAD ( expression , length [ , fill ] )
```

**Parameters**

<i>expression</i>	(CHAR OR VARCHAR) specifies the string to fill
<i>length</i>	(INTEGER) specifies the number of characters to return
<i>fill</i>	(CHAR OR VARCHAR) specifies the repeating string of characters with which to fill the output string. The default is the space character.

**Examples**

```
SELECT LPAD('database', 15, 'xzy');
      lpad
```

```
-----
xzyxzyxdatabase
(1 row)
```

If the string is already longer than the specified length it is truncated on the right:

```
SELECT LPAD('establishment', 10, 'abc');
      lpad
-----
establishm
(1 row)
```

## LTRIM

Returns a VARCHAR value representing a string with leading blanks removed from the left side (beginning).

### Behavior Type

Immutable

### Syntax

```
LTRIM ( expression [ , characters ] )
```

### Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to trim
<i>characters</i>	(CHAR or VARCHAR) specifies the characters to remove from the left side of <i>expression</i> . The default is the space character.

### Examples

```
SELECT LTRIM('zzzyyyyyyxxxxxxxxtrim', 'xyz');
      ltrim
-----
      trim
(1 row)
```

### See Also

**BTRIM** (page 317), **RTRIM** (page 348), **TRIM** (page 358)

## MD5

Calculates the MD5 hash of string, returning the result as a VARCHAR string in hexadecimal.

### Behavior Type

Immutable

### Syntax

```
MD5 ( string )
```

## Parameters

<i>string</i>	Is the argument string.
---------------	-------------------------

## Examples

```
SELECT MD5('123');
           md5
-----
202cb962ac59075b964b07152d234b70
(1 row)
```

```
SELECT MD5('Vertica'::bytea);
           md5
-----
fc45b815747d8236f9f6fdb9c2c3f676
(1 row)
```

## OCTET\_LENGTH

Returns the length of the input string expression in octets.

### Behavior Type

Immutable

### Syntax

```
OCTET_LENGTH ( expression )
```

### Parameters

<i>expression</i>	(CHAR or VARCHAR or BINARY or VARBINARY) is the string to measure.
-------------------	--

### Notes

- If the data type of *expression* is a CHAR, VARCHAR or VARBINARY, the result is the same as the actual length of *expression* in octets. For CHAR, the length does not include any trailing spaces.
- If the data type of *expression* is BINARY, the result is the same as the fixed-length of *expression*.
- If the value of *expression* is NULL, the result is NULL.

### Examples

Expression	Result
SELECT OCTET_LENGTH(CHAR(10) '1234 ');	4
SELECT OCTET_LENGTH(CHAR(10) '1234');	4
SELECT OCTET_LENGTH(CHAR(10) ' 1234');	6
SELECT OCTET_LENGTH(VARCHAR(10) '1234 ');	6

```

SELECT OCTET_LENGTH (VARCHAR(10) '1234 ');          5
SELECT OCTET_LENGTH (VARCHAR(10) '1234');          4
SELECT OCTET_LENGTH (VARCHAR(10) ' 1234');        7
SELECT OCTET_LENGTH ('abc'::VARBINARY);           3
SELECT OCTET_LENGTH (VARBINARY 'abc');           3
SELECT OCTET_LENGTH (VARBINARY 'abc ');          5
SELECT OCTET_LENGTH (BINARY(6) 'abc');           6
SELECT OCTET_LENGTH (VARBINARY '');              0
SELECT OCTET_LENGTH (' '::BINARY);               1
SELECT OCTET_LENGTH (null::VARBINARY);
SELECT OCTET_LENGTH (null::BINARY);
    
```

**See Also**

**BIT\_LENGTH** (page 315), **CHARACTER\_LENGTH** (page 318), **LENGTH** (page 335)

**OVERLAY**

Returns a VARCHAR value representing a string having had a substring replaced by another string.

**Behavior Type**

Immutable if using OCTETS, Stable otherwise

**Syntax**

```

OVERLAY ( expression1 PLACING expression2 FROM position
... [ FOR extent ]
... [ USING { CHARACTERS | OCTETS } ] )
    
```

**Parameters**

<i>expression1</i>	(CHAR or VARCHAR) is the string to process
<i>expression2</i>	(CHAR or VARCHAR) is the substring to overlay
<i>position</i>	(INTEGER) is the character or octet position (counting from one) at which to begin the overlay
<i>extent</i>	(INTEGER) specifies the number of characters or octets to replace with the overlay
USING CHARACTERS   OCTETS	Determines whether OVERLAY uses characters (the default) or octets

**Examples**

```

SELECT OVERLAY('123456789' PLACING 'xxx' FROM 2);
  overlay
-----
 1xxx56789
(1 row)
SELECT OVERLAY('123456789' PLACING 'XXX' FROM 2 USING OCTETS);
  overlay
-----
    
```

```

1XXX56789
(1 row)
SELECT OVERLAY('123456789' PLACING 'xxx' FROM 2 FOR 4);
overlay
-----
1xxx6789
(1 row)
SELECT OVERLAY('123456789' PLACING 'xxx' FROM 2 FOR 5);
overlay
-----
1xxx789
(1 row)
SELECT OVERLAY('123456789' PLACING 'xxx' FROM 2 FOR 6);
overlay
-----
1xxx89
(1 row)

```

## OVERLAYB

Returns an octet value representing a string having had a substring replaced by another string.

### Behavior Type

Immutable

### Syntax

```
OVERLAYB ( expression1, expression2, position [ , extent ] )
```

### Parameters

<i>expression1</i>	(CHAR or VARCHAR) is the string to process
<i>expression2</i>	(CHAR or VARCHAR) is the substring to overlay
<i>position</i>	(INTEGER) is the octet position (counting from one) at which to begin the overlay
<i>extent</i>	(INTEGER) specifies the number of octets to replace with the overlay

### Notes

This function treats the multibyte character string as a string of octets (bytes) and use octet numbers as incoming and outgoing position specifiers and lengths. The strings themselves are type VARCHAR, but they treated as if each byte was a separate character.

### Examples

```

SELECT OVERLAYB('123456789', 'ééé', 2);
OVERLAYB
-----
1ééé89
(1 row)
SELECT OVERLAYB('123456789', 'BBB', 2);
OVERLAYB
-----

```

```

1BBB89
(1 row)
SELECT OVERLAYB('123456789', 'xxx', 2);
OVERLAYB
-----
1xxx56789
(1 row)
SELECT OVERLAYB('123456789', 'xxx', 2, 4);
OVERLAYB
-----
1xxx6789
(1 row)
SELECT OVERLAYB('123456789', 'xxx', 2, 5);
OVERLAYB
-----
1xxx789
(1 row)
SELECT OVERLAYB('123456789', 'xxx', 2, 6);
OVERLAYB
-----
1xxx89
(1 row)

```

## POSITION

Returns an INTEGER value representing the character location of a specified substring with a string (counting from one).

### Behavior Type

Immutable if USING OCTETS, stable otherwise

### Syntax 1

```
POSITION ( substring IN string [ USING { CHARACTERS | OCTETS } ] )
```

### Parameters

<i>substring</i>	(CHAR or VARCHAR) is the substring to locate
<i>string</i>	(CHAR or VARCHAR) is the string in which to locate the substring
USING CHARACTERS   OCTETS	Determines whether the position is reported by using characters (the default) or octets.

### Syntax 2

```
POSITION ( substring IN string )
```

### Parameters

<i>substring</i>	(VARBINARY) is the substring to locate
------------------	--

<i>string</i>	(VARBINARY) is the string in which to locate the substring
---------------	--

**Notes**

- When the string and substring are CHAR or VARCHAR, the return value is based on either the character or octet position of the substring.
- When the string and substring are VARBINARY, the return value is always based on the octet position of the substring.
- The string and substring must be consistent. Do not mix VARBINARY with CHAR or VARCHAR.

**Examples**

```
SELECT POSITION('é' IN 'étudiant' USING CHARACTERS);
```

```
  position
```

```
-----
```

```
      1
```

```
(1 row)
```

```
SELECT POSITION('ß' IN 'straße' USING OCTETS);
```

```
  position
```

```
-----
```

```
      5
```

```
(1 row)
```

```
SELECT POSITION('c' IN 'abcd' USING CHARACTERS);
```

```
  position
```

```
-----
```

```
      3
```

```
(1 row)
```

```
SELECT POSITION(VARBINARY '456' IN VARBINARY '123456789');
```

```
  position
```

```
-----
```

```
      4
```

```
(1 row)
```

**POSITIONB**

Returns an INTEGER value representing the octet location of a specified substring with a string (counting from one).

**Behavior Type**

Immutable

**Syntax**

```
POSITIONB ( string, substring )
```

**Parameters**

<i>string</i>	(CHAR or VARCHAR) is the string in which to locate the substring
<i>substring</i>	(CHAR or VARCHAR) is the substring to locate

**Examples**

```
SELECT POSITIONB('straße', 'ße');
POSITIONB
-----
          5
(1 row)
SELECT POSITIONB('étudiant', 'é');
position
-----
          1
(1 row)
```

## QUOTE\_IDENT

Returns the given string, suitably quoted, to be used as an *identifier* (page 15) in a SQL statement string. Quotes are added only if necessary; that is, if the string contains non-identifier characters, is a SQL *keyword* (page 12), such as '1time', 'Next week' and 'Select'. Embedded double quotes are doubled.

### Behavior Type

Immutable

### Syntax

```
QUOTE_IDENT( string )
```

### Parameters

<i>string</i>	Is the argument string.
---------------	-------------------------

### Notes

- SQL identifiers, such as table and column names, are stored as created, and references to them are resolved using case-insensitive compares. Thus, you do not need to double-quote mixed-case identifiers.
- Vertica quotes all currently-reserved keywords, even those not currently being used.

### Examples

Quoted identifiers are case-insensitive, and Vertica does not supply the quotes:

```
SELECT QUOTE_IDENT('VErtIcA');
QUOTE_IDENT
-----
VErtIcA
(1 row)

SELECT QUOTE_IDENT('Vertica database');
QUOTE_IDENT
-----
"Vertica database"
(1 row)
```

Embedded double quotes are doubled:

```
SELECT QUOTE_IDENT('Vertica !" database');
      QUOTE_IDENT
-----
"Vertica ""!" database"
(1 row)
```

The following example uses the SQL keyword, SELECT; results are double quoted:

```
SELECT QUOTE_IDENT('select');
      QUOTE_IDENT
-----
"select"
(1 row)
```

## QUOTE\_LITERAL

Returns the given string, suitably quoted, to be used as a string literal in a SQL statement string. Embedded single quotes and backslashes are doubled.

### Behavior Type

Immutable

### Syntax

```
QUOTE_LITERAL ( string )
```

### Parameters

<i>string</i>	Is the argument string.
---------------	-------------------------

### Notes

Vertica recognizes two consecutive single quotes within a string literal as one single quote character. For example, 'You''re here!'. This is the SQL standard representation and is preferred over the form, 'You\'re here!', as backslashes are not parsed as before.

### Examples

```
SELECT QUOTE_LITERAL('You''re here!');
      QUOTE_LITERAL
-----
'You''re here!'
(1 row)
```

```
SELECT QUOTE_LITERAL('You\'re here!');
WARNING: nonstandard use of \' in a string literal at character 22
HINT: Use '' to write quotes in strings, or use the escape string syntax (E'').
```

### See Also

***String Literals (Character)*** (page 21)

## REPEAT

Returns a VARCHAR or VARBINARY value that repeats the given value COUNT times, given a value and a count this function.

If the return value is truncated the given value might not be repeated count times, and the last occurrence of the given value might be truncated.

## Behavior Type

Immutable

## Syntax

```
REPEAT ( string , repetitions )
```

## Parameters

<i>string</i>	(CHAR or VARCHAR or BINARY or VARBINARY) is the string to repeat
<i>repetitions</i>	(INTEGER) is the number of times to repeat the string

## Notes

If the repetitions field depends on the contents of a column (is not a constant), then the repeat operator maximum length is 65000 bytes. You can add a cast of the repeat to cast the result down to a size big enough for your purposes (reflects the actual maximum size) so you can do other things with the result.

REPEAT () and || check for result strings longer than 65000. REPEAT () silently truncates to 65000 octets, and || reports an error (including the octet length).

## Examples

The following example repeats 'vmart' three times:

```
SELECT REPEAT ('vmart', 3);
       repeat
-----
vmartvmartvmart
(1 row)
```

If you run the following example, you get an error message:

```
SELECT '123456' || REPEAT('a', colx);
ERROR: Operator || may give a 65006-byte Varchar result; the limit is 65000 bytes.
```

If you know that `colx` can never be greater than 3, the solution is to add a cast (`::VARCHAR(3)`):

```
SELECT '123456' || REPEAT('a', colx)::VARCHAR(3);
```

If `colx` is greater than 3, the repeat is truncated to exactly three (3) a's.

## REPLACE

Replaces all occurrences of characters in a string with another set of characters.

## Behavior Type

Immutable

**Syntax**

```
REPLACE ( string , target , replacement )
```

**Parameters**

<i>string</i>	(CHAR OR VARCHAR) is the string to which to perform the replacement
<i>target</i>	(CHAR OR VARCHAR) is the string to replace
<i>replacement</i>	(CHAR OR VARCHAR) is the string with which to replace the target

**Examples**

```
SELECT REPLACE('Documentation%20Library', '%20', ' ');
      replace
```

```
-----
Documentation Library
(1 row)
```

```
SELECT REPLACE('This & That', '&', 'and');
      replace
```

```
-----
This and That
(1 row)
```

```
SELECT REPLACE('straße', 'ß', 'ss');
      REPLACE
```

```
-----
strasse
(1 row)
```

**RIGHT**

Returns the specified characters from the right side of a string.

**Behavior Type**

Immutable

**Syntax**

```
RIGHT ( string , length )
```

**Parameters**

<i>string</i>	(CHAR or VARCHAR) is the string to return.
<i>length</i>	Is an INTEGER value that specifies the count of characters to return.

**Examples**

The following command returns the last three characters of the string 'vertica':

```
SELECT RIGHT('vertica', 3);
      right
```

```
-----
ica
```

(1 row)

The following command returns the last two characters of the string 'straße':

```
SELECT RIGHT('straße', 2);
      RIGHT
-----
      ße
(1 row)
```

**See Also**

**SUBSTR** (page 353)

**RPAD**

Returns a VARCHAR value representing a string of a specific length filled on the right with specific characters.

**Behavior Type**

Immutable

**Syntax**

```
RPAD ( expression , length [ , fill ] )
```

**Parameters**

<i>expression</i>	(CHAR OR VARCHAR) specifies the string to fill
<i>length</i>	(INTEGER) specifies the number of characters to return
<i>fill</i>	(CHAR OR VARCHAR) specifies the repeating string of characters with which to fill the output string. The default is the space character.

**Examples**

```
SELECT RPAD('database', 15, 'xzy');
      rpad
-----
databasexzyxzyx
(1 row)
```

If the string is already longer than the specified length it is truncated on the right:

```
SELECT RPAD('database', 6, 'xzy');
      rpad
-----
databa
(1 row)
```

**RTRIM**

Returns a VARCHAR value representing a string with trailing blanks removed from the right side (end).

## Behavior Type

Immutable

## Syntax

```
RTRIM ( expression [ , characters ] )
```

## Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to trim
<i>characters</i>	(CHAR or VARCHAR) specifies the characters to remove from the right side of <i>expression</i> . The default is the space character.

## Examples

```
SELECT RTRIM('trimzzzyyyyyyxxxxxxxx', 'xyz');
   ltrim
-----
   trim
(1 row)
```

## See Also

**BTRIM** (page 317), **LTRIM** (page 338), **TRIM** (page 358)

## SPACE

Inserts blank spaces into a specified location within a character string.

## Syntax

```
SELECT INSERT( 'string1', || SPACE (n) || 'string2');
```

## Parameters

<i>string1</i>	(VARCHAR) Is the string after which to insert the space.
<i>n</i>	A character of type INTEGER that represents the number of spaces to insert.
<i>string2</i>	( VARCHAR) Is the remainder of the string that appears after the inserted spaces

## Example

The following example inserts 10 spaces between the strings 'x' and 'y':

```
SELECT 'x' || SPACE(10) || 'y';
   ?column?
-----
   x          y
(1 row)
```

## SPLIT\_PART

Splits string on the delimiter and returns the location of the beginning of the given field (counting from one).

### Behavior Type

Stable

### Syntax

```
SPLIT_PART ( string , delimiter , field )
```

### Parameters

<i>string</i>	Is the argument string.
<i>delimiter</i>	Is the given delimiter.
<i>field</i>	(INTEGER) is the number of the part to return.

### Note

Use this with the character form of the subfield.

### Examples

The specified integer of 2 returns the second string, or `def`.

```
SELECT SPLIT_PART('abc~@~def~@~ghi', '~@~', 2);
split_part
-----
def
(1 row)
```

Here, we specify 3, which returns the third string, or `789`.

```
SELECT SPLIT_PART('123~|~456~|~789', '~|~', 3);
split_part
-----
789
(1 row)
```

Note that the tildes are for readability only. Omitting them returns the same results:

```
SELECT SPLIT_PART('123|456|789', '|', 3);
split_part
-----
789
(1 row)
```

See what happens if you specify an integer that exceeds the number of strings: No results.

```
SELECT SPLIT_PART('123|456|789', '|', 4);
split_part
-----
```

```
(1 row)
```

The above result is not null, it is the empty string.

```
SELECT SPLIT_PART('123|456|789', '|', 4) IS NULL;
?column?
```

```
-----
```

```
f
(1 row)
```

If `SPLIT_PART` had returned `NULL`, `LENGTH` would have returned null.

```
SELECT LENGTH (SPLIT_PART('123|456|789', '|', 4));
length
```

```
-----
```

```
0
(1 row)
```

## SPLIT\_PARTB

Splits string on the delimiter and returns the location of the beginning of the given field (counting from one).

### Behavior Type

Immutable

### Syntax

```
SPLIT_PARTB ( string , delimiter , field )
```

### Parameters

<i>string</i>	Is the argument string.
<i>delimiter</i>	Is the given delimiter.
<i>field</i>	(INTEGER) is the number of the part to return.

### Note

Use this function with the character form of the subfield.

### Examples

The specified integer of 3 returns the third string, or `soupçon`.

```
SELECT SPLIT_PARTB('straße~@~café~@~soupçon', '~@~', 3);
SPLIT_PARTB
```

```
-----
```

```
soupçon
(1 row)
```

Note that the tildes are for readability only. Omitting them returns the same results:

```
SELECT SPLIT_PARTB('straße @ café @ soupçon', '@', 3);
SPLIT_PARTB
```

```
-----
```

```
soupçon
```

(1 row)

See what happens if you specify an integer that exceeds the number of strings: No results.

```
SELECT SPLIT_PARTB('straße @ café @ soupçon', '@', 4);
SPLIT_PARTB
```

-----

(1 row)

The above result is not null, it is the empty string.

```
SELECT SPLIT_PARTB('straße @ café @ soupçon', '@', 4) IS NULL;
?column?
```

-----

f

(1 row)

## STRPOS

Returns an INTEGER value representing the character location of a specified substring within a string (counting from one).

### Behavior Type

Stable

### Syntax

```
STRPOS ( string , substring )
```

### Parameters

<i>string</i>	(CHAR or VARCHAR) is the string in which to locate the substring
<i>substring</i>	(CHAR or VARCHAR) is the substring to locate

### Notes

STRPOS is identical to **POSITION** (page 342) except for the order of the arguments.

### Examples

```
SELECT STRPOS('abcd','c');
```

```
  strpos
```

-----

3

(1 row)

## STRPOSB

Returns an INTEGER value representing the character location of a specified substring within a string (counting from one).

### Behavior Type

Immutable

**Syntax**

```
STRPOSB ( string , substring )
```

**Parameters**

<i>string</i>	(CHAR or VARCHAR) is the string in which to locate the substring
<i>substring</i>	(CHAR or VARCHAR) is the substring to locate

**Notes**

STRPOSB is identical to **POSITIONB** (page 343) except for the order of the arguments.

**Examples**

```
SELECT STRPOSB('straße', 'ße');
STRPOSB
-----
          5
(1 row)
SELECT STRPOSB('étudiant', 'é');
position
-----
          1
(1 row)
```

**SUBSTR**

Returns a VARCHAR value representing a substring of a specified string.

**Behavior Type**

Immutable

**Syntax**

```
SUBSTR ( string , position [ , extent ] )
```

**Parameters**

<i>string</i>	(CHAR or VARCHAR or BINARY or VARBINARY) is the string from which to extract a substring.
<i>position</i>	(INTEGER or DOUBLE PRECISION) is the starting position of the substring (counting from one by characters). Note that this function truncates DOUBLE PRECISION input values.
<i>extent</i>	(INTEGER or DOUBLE PRECISION) is the length of the substring to extract (in characters). The default is the end of the string. Note that this function truncates DOUBLE PRECISION input values.

**Notes**

SUBSTR performs the same function as **SUBSTRING** (page 355). The only difference is the syntax allowed.

## Examples

```
SELECT SUBSTR('123456789', 3, 2);
  substr
-----
    34
(1 row)
SELECT SUBSTR('123456789', 3);
  substr
-----
3456789
(1 row)
SELECT SUBSTR(TO_BITSTRING(HEX_TO_BINARY('0x10')), 2, 2);
  substr
-----
    00
(1 row)
SELECT SUBSTR(TO_HEX(10010), 2, 2);
  substr
-----
    71
(1 row)
```

## SUBSTRB

Returns an octet value representing the substring of a specified string.

### Behavior Type

Immutable

### Syntax

```
SUBSTRB ( string , position [ , extent ] )
```

### Parameters

<i>string</i>	(CHAR or VARCHAR or BINARY or VARBINARY) is the string from which to extract a substring.
<i>position</i>	(INTEGER or DOUBLE PRECISION) is the starting position of the substring (counting from one in octets). Note that this function truncates DOUBLE PRECISION input values.
<i>extent</i>	(INTEGER or DOUBLE PRECISION) is the length of the substring to extract (in octets). The default is the end of the string. Note that this function truncates DOUBLE PRECISION input values.

### Notes

This function treats the multibyte character string as a string of octets (bytes) and use octet numbers as incoming and outgoing position specifiers and lengths. The strings themselves are type VARCHAR, but they treated as if each octet was a separate character.

## Examples

```
SELECT SUBSTRB('soupçon', 5);
SUBSTRB
-----
çon
(1 row)
SELECT SUBSTRB('soupçon', 5, 2);
SUBSTRB
-----
ç
(1 row)
```

## SUBSTRING

Returns a value representing a substring of the specified string at the given position, given a value, a position, and an optional length.

### Behavior Type

Immutable if USING OCTETS, stable otherwise.

### Syntax

```
SUBSTRING ( string , position [ , length ]
... [USING {CHARACTERS | OCTETS } ] )
SUBSTRING ( string FROM position [ FOR length ]
... [USING { CHARACTERS | OCTETS } ] )
```

### Parameters

<i>string</i>	(CHAR or VARCHAR or BINARY or VARBINARY) is the string from which to extract a substring
<i>position</i>	(INTEGER or DOUBLE PRECISION) is the starting position of the substring (counting from one by either characters or octets). (The default is characters.) If position is greater than the length of the given value, an empty value is returned. Note that this function truncates DOUBLE PRECISION input values.
<i>length</i>	(INTEGER or DOUBLE PRECISION) is the length of the substring to extract in either characters or octets. (The default is characters.) The default is the end of the string. If a length is given the result is at most that many bytes. The maximum length is the length of the given value less the given position. If no length is given or if the given length is greater than the maximum length then the length is set to the maximum length. Note that this function truncates DOUBLE PRECISION input values.
USING CHARACTERS  OCTETS	Determines whether the value is expressed in characters (the default) or octets.

## Notes

Neither length nor position can be negative, and the position cannot be zero because it is one based. If these forms are violated, the system returns an error:

```
SELECT SUBSTRING('ab'::binary(2), -1, 2);
ERROR:  negative or zero substring start position not allowed
```

## Examples

```
SELECT SUBSTRING('soupçon', 5, 2 USING CHARACTERS);
substring
-----
çö
(1 row)
SELECT SUBSTRING('soupçon', 5, 2 USING OCTETS);
substrb
-----
ç
(1 row)
```

## TO\_BITSTRING

Returns a VARCHAR that represents the given VARBINARY value in bitstring format

### Behavior Type

Immutable

### Syntax

```
TO_BITSTRING ( expression )
```

### Parameters

<i>expression</i>	(VARCHAR) is the string to return.
-------------------	------------------------------------

## Notes

VARCHAR TO\_BITSTRING(VARBINARY) converts data from binary type to character type (where the character representation is the bitstring format). This function is the inverse of BITSTRING\_TO\_BINARY:

```
TO_BITSTRING(BITSTRING_TO_BINARY(x)) = x
BITSTRING_TO_BINARY(TO_BITSTRING(x)) = x
```

## Examples

```
SELECT TO_BITSTRING('ab'::BINARY(2));
to_bitstring
-----
0110000101100010
(1 row)
SELECT TO_BITSTRING(HEX_TO_BINARY('0x10'));
to_bitstring
```

```

-----
00010000
(1 row)
SELECT TO_BITSTRING(HEX_TO_BINARY('0xF0'));
to_bitstring
-----
11110000
(1 row)

```

**See Also**

**BITCOUNT** (page 316) and **BITSTRING\_TO\_BINARY** (page 316)

**TO\_HEX**

Returns a VARCHAR or VARBINARY representing the hexadecimal equivalent of a number.

**Behavior Type**

Immutable

**Syntax**

```
TO_HEX ( number )
```

**Parameters**

<i>number</i>	(INTEGER) is the number to convert to hexadecimal
---------------	---

**Notes**

VARCHAR TO\_HEX(INTEGER) and VARCHAR TO\_HEX(VARBINARY) are similar. The function converts data from binary type to character type (where the character representation is in hexadecimal format). This function is the inverse of HEX\_TO\_BINARY.

```
TO_HEX(HEX_TO_BINARY(x)) = x .
HEX_TO_BINARY(TO_HEX(x)) = x .
```

**Examples**

```

SELECT TO_HEX(123456789);
to_hex
-----
75bcd15
(1 row)

```

For VARBINARY inputs, the returned value is not preceded by "0x". For example:

```

SELECT TO_HEX('ab'::binary(2));
to_hex
-----
6162
(1 row)

```

## TRANSLATE

Replaces individual characters in *string\_to\_replace* with other characters.

### Behavior Type

Immutable

### Syntax

```
TRANSLATE ( string_to_replace , from_string , to_string );
```

### Parameters

<i>string_to_replace</i>	Is the string to be translated.
<i>from_string</i>	Contains characters that should be replaced in <i>string_to_replace</i> .
<i>to_string</i>	Any character in <i>string_to_replace</i> that matches a character in <i>from_string</i> is replaced by the corresponding character in <i>to_string</i> .

### Example

```
SELECT TRANSLATE('straße', 'ß', 'ss');
TRANSLATE
-----
strase
(1 row)
```

## TRIM

Combines the BTRIM, LTRIM, and RTRIM functions into a single function.

### Behavior Type

Immutable

### Syntax

```
TRIM ( [ [ LEADING | TRAILING | BOTH ] characters FROM ] expression )
```

### Parameters

LEADING	Removes the specified characters from the left side of the string
TRAILING	Removes the specified characters from the right side of the string
BOTH	Removes the specified characters from both sides of the string (default)
<i>characters</i>	(CHAR or VARCHAR) specifies the characters to remove from <i>expression</i> . The default is the space character.
<i>expression</i>	(CHAR or VARCHAR) is the string to trim

## Examples

```

SELECT '-' || TRIM(LEADING 'x' FROM 'xxdatabasexx') || '-';
?column?
-----
-databasexx-
(1 row)
SELECT '-' || TRIM(TRAILING 'x' FROM 'xxdatabasexx') || '-';
?column?
-----
-xxdatabase-
(1 row)
SELECT '-' || TRIM(BOTH 'x' FROM 'xxdatabasexx') || '-';
?column?
-----
-database-
(1 row)
SELECT '-' || TRIM('x' FROM 'xxdatabasexx') || '-';
?column?
-----
-database-
(1 row)
SELECT '-' || TRIM(LEADING FROM ' database ') || '-';
?column?
-----
-database -
(1 row)
SELECT '-' || TRIM(' database ') || '-';
?column?
-----
-database-
(1 row)

```

## See Also

**BTRIM** (page 317)

**LTRIM** (page 338)

**RTRIM** (page 348)

## UPPER

Returns a VARCHAR value containing the argument converted to uppercase letters.

### Behavior Type

Stable

### Syntax

```
UPPER ( expression )
```

### Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to convert
-------------------	--

## Notes

UPPER is restricted to 32750 octet inputs, since it is possible for the UTF-8 representation of result to double in size.

## Examples

```
SELECT UPPER('AbCdEfG');
  upper
-----
ABCDEFG
(1 row)
SELECT UPPER('étudiant');
  UPPER
-----
étUDIANT
(1 row)
```

## UPPERB

Returns a character string with each ASCII character converted to uppercase; multibyte UTF-8 characters are not converted.

## Behavior Type

Immutable

## Syntax

```
UPPERB ( expression )
```

## Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to convert
-------------------	--

## Examples

In the following example, the multibyte UTF-8 character é is not converted to uppercase:

```
SELECT UPPERB('étudiant');
  UPPERB
-----
étUDIANT
(1 row)

SELECT UPPERB('AbCdEfG');
  UPPERB
-----
ABCDEFG
(1 row)
SELECT UPPERB('The Vertica Database');
  UPPERB
-----
```

THE VERTICA DATABASE  
(1 row)

## V6\_ATON

Converts an IPv6 address represented as a character string to a binary string.

### Behavior Type

Immutable

### Syntax

V6\_ATON ( *expression* )

### Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

### Notes

The following syntax converts an IPv6 address represented as the character string A to a binary string B.

V6\_ATON trims any spaces from the right of A and calls the Linux function *inet\_pton* [http://www.opengroup.org/onlinepubs/000095399/functions/inet\\_ntop.html](http://www.opengroup.org/onlinepubs/000095399/functions/inet_ntop.html).

V6\_ATON (VARCHAR A) -> VARBINARY(16) B

If A has no colons it is prepended with '::ffff:'. If A is NULL, too long, or if *inet\_pton* returns an error, the result is NULL.

### Examples

```
SELECT V6_ATON('2001:DB8::8:800:200C:417A');
           v6_aton
-----
\001\015\270\000\000\000\000\000\010\010\000 \014Az
(1 row)
SELECT TO_HEX(V6_ATON('2001:DB8::8:800:200C:417A'));
           to_hex
-----
20010db80000000000000080800200c417a
(1 row)
SELECT V6_ATON('1.2.3.4');
           v6_aton
-----
\000\000\000\000\000\000\000\000\000\000\000\000\377\377\001\002\003\004
(1 row)
SELECT V6_ATON('::1.2.3.4');
           v6_aton
-----
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\001\002\003\004
(1 row)
```

**See Also****V6\_NTOA** (page 275)**V6\_NTOA**

Converts an IPv6 address represented as varbinary to a character string.

**Behavior Type**

Immutable

**Syntax**V6\_NTOA ( *expression* )**Parameters**

<i>expression</i>	(VARBINARY) is the binary string to convert.
-------------------	--

**Notes**

The following syntax converts an IPv6 address represented as VARBINARY B to a string A.

V6\_NTOA right-pads B to 16 bytes with zeros, if necessary, and calls the Linux function *inet\_ntop* [http://www.opengroup.org/onlinepubs/000095399/functions/inet\\_ntop.html](http://www.opengroup.org/onlinepubs/000095399/functions/inet_ntop.html).

V6\_NTOA (VARBINARY B) -&gt; VARCHAR A

If B is NULL or longer than 16 bytes, the result is NULL.

Vertica automatically converts the form '::ffff:1.2.3.4' to '1.2.3.4'.

**Examples**

```
SELECT V6_NTOA(' \001\015\270\000\000\000\000\000\010\010\000 \014Az');
      v6_ntoa
-----
2001:db8::8:800:200c:417a
(1 row)
SELECT V6_NTOA(V6_ATON('1.2.3.4'));
      v6_ntoa
-----
1.2.3.4
(1 row)
SELECT V6_NTOA(V6_ATON('::1.2.3.4'));
```

```

v6_ntoa
-----
::1.2.3.4
(1 row)

```

**See Also****N6\_ATON** (page 273)**V6\_SUBNETA**

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a binary or alphanumeric IPv6 address.

**Behavior Type**

Immutable

**Syntax**

```
V6_SUBNETA ( expression1, expression2 )
```

**Parameters**

<i>expression1</i>	(VARBINARY or VARCHAR) is the string to calculate.
<i>expression2</i>	(INTEGER) is the size of the subnet.

**Notes**

The following syntax calculates a subnet address in CIDR format from a binary or varchar IPv6 address.

V6\_SUBNETA masks a binary IPv6 address B so that the N leftmost bits form a subnet address, while the remaining rightmost bits are cleared. It then converts to an alphanumeric IPv6 address, appending a slash and N.

```
V6_SUBNETA(BINARY B, INT8 N) -> VARCHAR C
```

The following syntax calculates a subnet address in CIDR format from an alphanumeric IPv6 address.

```
V6_SUBNETA(VARCHAR A, INT8 N) -> V6_SUBNETA(V6_ATON(A), N) -> VARCHAR C
```

**Examples**

```

SELECT V6_SUBNETA(V6_ATON('2001:db8::8:800:200c:417a'), 28);
   v6_subneta
-----
2001:db0::/28
(1 row)

```

**See Also****V6\_SUBNETN** (page 276)

## V6\_SUBNETN

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a varbinary or alphanumeric IPv6 address.

### Behavior Type

Immutable

### Syntax

```
V6_SUBNETN ( expression1, expression2 )
```

### Parameters

<i>expression1</i>	(VARBINARY or VARCHAR or INTEGER) is the string to calculate.
<i>expression2</i>	(INTEGER) is the size of the subnet.

### Notes

The following syntax masks a BINARY IPv6 address **B** so that the N left-most bits of **S** form a subnet address, while the remaining right-most bits are cleared.

V6\_SUBNETN right-pads B to 16 bytes with zeros, if necessary and masks B, preserving its N-bit subnet prefix.

```
V6_SUBNETN (VARBINARY B, INT8 N) -> VARBINARY(16) S
```

If B is NULL or longer than 16 bytes, or if N is not between 0 and 128 inclusive, the result is NULL.

$S = [B]/N$  in **Classless Inter-Domain Routing**

[http://en.wikipedia.org/wiki/Classless\\_Inter-Domain\\_Routing](http://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing) notation (CIDR notation).

The following syntax masks an alphanumeric IPv6 address **A** so that the N leftmost bits form a subnet address, while the remaining rightmost bits are cleared.

```
V6_SUBNETN (VARCHAR A, INT8 N) -> V6_SUBNETN (V6_ATON(A), N) -> VARBINARY(16) S
```

### Example

```
SELECT V6_SUBNETN (V6_ATON('2001:db8::8:800:200c:417a'), 28);
           v6_subnetn
```

```
-----
\001\015\260\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
```

### See Also

**V6\_SUBNETA** (page 275)

## V6\_TYPE

Characterizes a binary or alphanumeric IPv6 address B as an integer type.

## Behavior Type

Immutable

## Syntax

V6\_TYPE ( *expression* )

## Parameters

<i>expression</i>	(VARBINARY or VARCHAR) is the type to convert.
-------------------	--

## Notes

V6\_TYPE(VARBINARY B) returns INT8 T.

V6\_TYPE(VARCHAR A) -> V6\_TYPE(V6\_ATON(A)) -> INT8 T

The IPv6 types are defined in the Network Working Group's **IP Version 6 Addressing Architecture memo** <http://www.ietf.org/rfc/rfc4291.txt>.

```

GLOBAL = 0      Global unicast addresses
LINKLOCAL = 1   Link-Local unicast (and Private-Use) addresses
LOOPBACK = 2    Loopback
UNSPECIFIED = 3 Unspecified
MULTICAST = 4   Multicast

```

IPv4-mapped and IPv4-compatible IPv6 addresses are also interpreted, as specified in **IPv4 Global Unicast Address Assignments** <http://www.iana.org/assignments/ipv4-address-space>.

- For IPv4, Private-Use is grouped with Link-Local.
- If B is VARBINARY, it is right-padded to 16 bytes with zeros, if necessary.
- If B is NULL or longer than 16 bytes, the result is NULL.

## Details

IPv4 (either kind):

```

0.0.0.0/8      UNSPECIFIED
10.0.0.0/8     LINKLOCAL
127.0.0.0/8    LOOPBACK
169.254.0.0/16 LINKLOCAL
172.16.0.0/12  LINKLOCAL
192.168.0.0/16 LINKLOCAL
224.0.0.0/4    MULTICAST
others         GLOBAL

```

IPv6:

```

::0/128        UNSPECIFIED
::1/128        LOOPBACK
fe80::/10      LINKLOCAL
ff00::/8       MULTICAST
others         GLOBAL

```

## Examples

```
SELECT V6_TYPE(V6_ATON('192.168.2.10'));
v6_type
-----
          1
(1 row)
SELECT V6_TYPE(V6_ATON('2001:db8::8:800:200c:417a'));
v6_type
-----
          0
(1 row)
```

## See Also

***INET\_ATON*** (page 272)

***IP Version 6 Addressing Architecture*** <http://www.ietf.org/rfc/rfc4291.txt>

***IPv4 Global Unicast Address Assignments***  
<http://www.iana.org/assignments/ipv4-address-space>

## System Information Functions

These functions provide system information regarding user sessions. The superuser has unrestricted access to all system information, but users can view only information about their own, current sessions.

### CURRENT\_DATABASE

Returns a VARCHAR value containing the name of the database to which you are connected.

#### Behavior Type

Immutable

#### Syntax

```
CURRENT_DATABASE()
```

#### Notes

The CURRENT\_DATABASE function does not require parentheses.

#### Examples

```
SELECT CURRENT_DATABASE();
  current_database
-----
  vmartschema
(1 row)
```

The following command returns the same results without the parentheses:

```
SELECT CURRENT_DATABASE;
  current_database
-----
  vmartschema
(1 row)
```

### CURRENT\_SCHEMA

Shows the resolved name of \$User.

#### Behavior Type

Stable

#### Syntax

```
CURRENT_SCHEMA()
```

#### Notes

If the search path for USER1 is: \$USER, COMMON, PUBLIC:

SELECT CURRENT\_SCHEMA() returns the following output if schema USER1 exists:

USER1

If schema USER1 does not exist, it returns the following output:

COMMON

### Example

```
SELECT CURRENT_SCHEMA();
  current_schema
-----
  public
(1 row)
```

## CURRENT\_USER

Returns a VARCHAR containing the name of the user who initiated the current database connection.

### Behavior Type

Stable

### Syntax

```
CURRENT_USER()
```

### Notes

- The CURRENT\_USER function does not require parentheses.
- This function is useful for permission checking and is equivalent to **SESSION\_USER** (page 369) and **USER** (page 370).

### Examples

```
SELECT CURRENT_USER();
  current_user
-----
  dbadmin
(1 row)
```

The following command returns the same results without the parentheses:

```
SELECT CURRENT_USER;
  current_user
-----
  dbadmin
(1 row)
```

## HAS\_TABLE\_PRIVILEGE

Returns a true/false value indicating whether a user can access a table in a particular way.

### Behavior Type

Stable

## Syntax

```
HAS_TABLE_PRIVILEGE ( [ user, ] table , privilege )
```

## Parameters

<i>user</i>	Specifies the name or OID of a database user. The default is the <b>CURRENT_USER</b> (page 368).
<i>table</i>	Specifies the name or OID of a table in the logical schema.
<i>privilege</i>	<ul style="list-style-type: none"> <li>▪ <b>SELECT</b> Allows the user to <b>SELECT</b> from any column of the specified table.</li> <li>▪ <b>INSERT</b> Allows the user to <b>INSERT</b> records into the specified table and to use the <b>COPY</b> (page 607) command to load the table.</li> <li>▪ <b>UPDATE</b> Allows the user to <b>UPDATE</b> records in the specified table.</li> <li>▪ <b>DELETE</b> Allows the user to delete a row from the specified table.</li> <li>▪ <b>REFERENCES</b> Allows the user to create a foreign key constraint (privileges required on both the referencing and referenced tables).</li> </ul>

## Examples

```
SELECT HAS_TABLE_PRIVILEGE('store.store_dimension', 'SELECT');
has_table_privilege
-----
t
(1 row)
SELECT HAS_TABLE_PRIVILEGE('release', 'store.store_dimension',
'INSERT');
has_table_privilege
-----
t
(1 row)
SELECT HAS_TABLE_PRIVILEGE('store.store_dimension', 'UPDATE');
has_table_privilege
-----
t
(1 row)
SELECT HAS_TABLE_PRIVILEGE('store.store_dimension', 'REFERENCES');
has_table_privilege
-----
t
(1 row)
SELECT HAS_TABLE_PRIVILEGE(45035996273711159, 45035996273711160,
'select');
has_table_privilege
-----
t
(1 row)
```

## SESSION\_USER

Returns a VARCHAR containing the name of the user who initiated the current database session.

## Behavior Type

Stable

## Syntax

```
SESSION_USER()
```

## Notes

- The `SESSION_USER` function does not require parentheses.
- Is equivalent to ***CURRENT\_USER*** (page 368) and ***USER*** (page 370).

## Examples

```
SELECT SESSION_USER();
  session_user
-----
  dbadmin
(1 row)
```

The following command returns the same results without the parentheses:

```
SELECT SESSION_USER;
  session_user
-----
  dbadmin
(1 row)
```

## USER

Returns a VARCHAR containing the name of the user who initiated the current database connection.

## Behavior Type

Stable

## Syntax

```
USER()
```

## Notes

- The `USER` function does not require parentheses.
- Is equivalent to ***CURRENT\_USER*** (page 368).

## Examples

```
SELECT USER();
  current_user
-----
  dbadmin
(1 row)
```

The following command returns the same results without the parentheses:

```
SELECT USER;
```

```
current_user
-----
dbadmin
(1 row)
```

## VERSION

Returns a VARCHAR containing a Vertica node's version information.

### Behavior Type

Stable

### Syntax

```
VERSION()
```

### Examples

```
SELECT VERSION();
                VERSION
-----
Vertica Analytic Database v4.0.12-20100513010203
(1 row)
```

The parentheses are required. If you omit them, the system returns an error:

```
SELECT VERSION;
ERROR: column "version" does not exist
```

## URI Encode/Decode Functions

The functions in this section follow the RFC 3986 standard for percent-encoding a Universal Resource Identifier (URI).

### URI\_PERCENT\_DECODE

Decodes a percent-encoded Universal Resource Identifier (URI) according to the RFC 3986 standard.

#### Syntax

```
URI_PERCENT_DECODE (expression)
```

#### Behavior type

Immutable

#### Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

### Examples

The following example invokes `uri_percent_decode` on the `Websites` column of the `URI` table and returns a decoded URI:

```
=> SELECT URI_PERCENT_DECODE(Websites) from URI;
       URI_PERCENT_DECODE
-----
http://www.faqs.org/rfcs/rfc3986.html x xj%a%
(1 row)
```

The following example returns the original URI in the Websites column and its decoded version:

```
=> SELECT Websites, URI_PERCENT_DECODE (Websites) from URI;
       Websites |
-----+-----
http://www.faqs.org/rfcs/rfc3986.html+x%20x%6a%a% |
http://www.faqs.org/rfcs/rfc3986.html x xj%a%
(1 row)
```

## URI\_PERCENT\_ENCODE

Encodes a Universal Resource Identifier (URI) according to the RFC 3986 standard for percent encoding. In addition, for compatibility with older encoders this function converts '+' to space; space is converted to %20 by uri\_percent\_encode.

### Syntax

```
URI_PERCENT_ENCODE (expression)
```

### Behavior type

Immutable

### Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

### Example

The following example shows how the `uri_percent_encode` function is invoked on a the Websites column of the URI table and returns an encoded URI:

```
=> SELECT URI_PERCENT_ENCODE(Websites) from URI;
       URI_PERCENT_ENCODE
-----
http%3A%2F%2Fexample.com%2F%3F%3D11%2F15
(1 row)
```

The following example returns the original URI in the Websites column and it's encoded form:

```
=> SELECT Websites, URI_PERCENT_ENCODE(Websites) from URI;
       Websites |
-----+-----
http://example.com/?=11/15 | http%3A%2F%2Fexample.com%2F%3F%3D11%2F15
(1 row)
```



## Vertica Functions

The functions in this section are used to query or change the internal state of Vertica and are not part of the SQL standard. Since these meta-functions access internal data structures, they cannot be called in DML, DDL, or `SELECT` queries.

The behavior type of Vertica built-in functions is immutable.

### Alphabetical List of Vertica Functions

This section contains all Vertica-specific functions, listed alphabetically. These meta functions are also grouped into their appropriate category; for example:

- **Catalog management functions** (page 473)
- **Cluster scaling functions** (page 479)
- **Constraint management functions** (page 480)
- **Data Collector functions** (page 490)
- **Database management functions** (page 492)
- **Epoch management functions** (page 499)
- **License management functions** (page 505)
- **Partition management functions** (page 511)
- **Pattern matching functions** (page 519)
- **Projection management functions** (page 523)
- **Purge functions** (page 530)
- **Regular expression functions** (page 533)
- **Session management functions** (page 546)
- **Statistic management functions** (page 556)
- **Storage management functions** (page 563)
- **Time series functions** (page 569)
- **Tuple Mover functions** (page 572)
- **Workload management functions** (page 573)

### ADD\_LOCATION

Adds a location to store data.

#### Syntax

```
ADD_LOCATION ( 'path' , [ 'node' , 'usage_string' ] )
```

#### Parameters

<i>path</i>	Specifies where the storage location is mounted. Path must be an empty directory with write permissions for user, group, or all.
-------------	---

<i>node</i>	Is the Vertica node where the location is available. If this parameter is omitted, <i>node</i> defaults to the initiator.
<i>usage_string</i>	Is one of the following: <ul style="list-style-type: none"> <li>▪ DATA: Only data is stored in the location.</li> <li>▪ TEMP: Only temporary files that are created during loads or queries are stored in the location.</li> <li>▪ DATA,TEMP: Both types of files are stored in the location.</li> </ul> If this parameter is omitted, the default is DATA,TEMP.

### Notes

- By default, the location is used to store both data and temporary files.
- Locations can be added from any node to any node.
- Either *node* and *usage\_string* must both be specified or neither of them specified.
- Information about storage locations is visible **V\_MONITOR.DISK\_STORAGE** (page 844).
- A storage location annotation called CATALOG indicates the location is used to store the catalog and is visible in V\_MONITOR.DISK\_STORAGE. However, no new locations can be added, as CATALOG locations and existing CATALOG annotations cannot be removed.

### Example

This example adds a location that stores data and temporary files:

```
SELECT ADD_LOCATION('/secondVerticaStorageLocation/');
```

This example adds a location to store data only:

```
SELECT ADD_LOCATION('/secondVerticaStorageLocation/' , 'node2' , 'DATA');
```

### See Also

**ALTER\_LOCATION\_USE** (page 376)

**RETIRE\_LOCATION** (page 461)

## ADVANCE\_EPOCH

Manually closes the current epoch and begins a new epoch.

### Syntax

```
ADVANCE_EPOCH ( [ integer ] )
```

### Parameters

<i>integer</i>	Specifies the number of epochs to advance.
----------------	--

### Note

This function is primarily maintained for backward compatibility with earlier versions of Vertica.

### Example

The following command increments the epoch number by 1:

```
=> SELECT ADVANCE_EPOCH(1);
```

### See Also

**ALTER PROJECTION** (page 580)

## ALTER\_LOCATION\_USE

Alters the type of files stored in the specified storage location.

### Syntax

```
ALTER_LOCATION_USE ( 'path' , [ 'node' ] , 'usage_string' )
```

### Parameters

<i>path</i>	Specifies where the storage location is mounted.
<i>node</i>	[Optional] Is the Vertica node where the location is available. If this parameter is omitted, <i>node</i> defaults to the initiator.
<i>usage_string</i>	Is one of the following: <ul style="list-style-type: none"><li>▪ DATA: Only data is stored in the location.</li><li>▪ TEMP: Only temporary files that are created during loads or queries are stored in the location.</li><li>▪ DATA,TEMP: Both types of files are stored in the location.</li></ul>

### Notes

- Altering the type of files stored in a particular location is useful if you create additional storage locations and you want to isolate execution engine temporary files from data files.
- After modifying the location's use, at least one location must remain for storing data and temp files. These files can be stored in the same storage location or separate storage locations.

- When a storage location is altered, it stores only the type of information indicated from that point forward. For example:
  - If you modify a storage location that previously stored both temp and data files so that it only stores temp files, the data is eventually merged out through the ATM. You can also merge it out manually.
  - If you modify a storage location that previously stored both temp and data files so that it only stores data files, all currently running statements that use these temp files, such as queries and loads, continue to run. Subsequent statements will no longer use this location.

### Example

The following example alters the storage location on node3 to store data only:

```
=> SELECT ALTER_LOCATION_USE ('/thirdVerticaStorageLocation/' , 'node3' ,
'DATA');
```

### See Also

**ADD\_LOCATION** (page 374)

**RETIRE\_LOCATION** (page 461)

Modifying Storage Locations in the Administrator's Guide

## ANALYZE\_CONSTRAINTS

Analyzes and reports on constraint violations within the current schema search path.

You can check for constraint violations by passing an empty argument (which returns violations on all tables within the current schema), by passing a single table argument, or by passing two arguments containing a table name and a column or list of columns.

### Syntax

```
ANALYZE_CONSTRAINTS [ ( '' )
... | ( schema.table )
... | [ ( schema.table , column ) ]
```

### Parameters

<code>('')</code>	Analyzes and reports on all tables within the current schema search path.
<code><i>table</i></code>	Analyzes and reports on all constraints referring to the specified table.
<code><i>column</i></code>	Analyzes and reports on all constraints referring to specified table that contains the specified columns.

### Notes

- `ANALYZE_CONSTRAINTS()`, takes locks in the same way that `SELECT * FROM t1` holds a lock on table `t1`. See **LOCKS** (page 858) for additional information.

- Use **COPY** (page 607) with NO COMMIT keywords to incorporate detection of constraint violations into the load process. Vertica checks for constraint violations when queries are run, not when data is loaded. To avoid constraint violations, load data without committing it and then perform a post-load check of your data using the ANALYZE\_CONSTRAINTS function. If the function finds constraint violations, you can roll back the load because you have not committed it.
- ANALYZE\_CONSTRAINTS() fails if the database cannot perform constraint checks, such as when the system is out of resources. Vertica returns an error that identifies the specific condition that caused the failure.
- When ANALYZE\_CONSTRAINTS finds violations, such as when you insert a duplicate value into a primary key, you can correct errors using the following functions. Effects last until the end of the session only:
  - SELECT **DISABLE\_DUPLICATE\_KEY\_ERROR** (page 403)
  - SELECT **REENABLE\_DUPLICATE\_KEY\_ERROR** (page 446)
- If you specify the wrong table, the system returns an error message:
 

```
SELECT ANALYZE_CONSTRAINTS('abc');
ERROR: 'abc' is not a table in the current search path
```
- If you issue the function using incorrect syntax, the system returns an error message with a hint:
 

```
ANALYZE ALL CONSTRAINT;
Or
ANALYZE CONSTRAINT abc;
ERROR: ANALYZE CONSTRAINT is not supported.
HINT: You may consider using analyze_constraints().
```
- ANALYZE\_CONSTRAINTS returns an error if run from a non-default locale; for example:
 

```
=> \locale LEN
INFO: Canonical locale: 'en'
INFO:   English
INFO: Standard collation: 'LEN'
=> SELECT ANALYZE_CONSTRAINTS('t1');
ERROR: ANALYZE_CONSTRAINTS is currently not supported in non-default
       locales
HINT: Set the locale in this session to en_US@collation=binary using
       the
       command "\locale en_US@collation=binary"
```

## Return Values

ANALYZE\_CONSTRAINTS returns results in a structured set (see table below) that lists the schema name, table name, column name, constraint name, constraint type, and the column values that caused the violation.

If the result set is empty, then no constraint violations exist; for example:

```
SELECT ANALYZE_CONSTRAINTS ('public.product_dimension', 'product_key');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
(0 rows)
```

The following result set, on the other hand, shows a primary key violation, along with the value that caused the violation ('10'):

```
SELECT ANALYZE_CONSTRAINTS ('');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 store      | t1         | c1           | pk_t1          | PRIMARY        | ('10')
(1 row)
```

The result set columns are described in further detail in the following table:

Column Name	Data Type	Description
Schema Name	VARCHAR	The name of the schema.
Table Name	VARCHAR	The name of the table, if specified.
Column Names	VARCHAR	Names of columns containing constraints. Multiple columns are in a comma-separated list: store_key, store_key, date_key,
Constraint Name	VARCHAR	The given name of the primary key, foreign key, unique, or not null constraint, if specified.
Constraint Type	VARCHAR	Identified by one of the following strings: 'PRIMARY KEY', 'FOREIGN KEY', 'UNIQUE', or 'NOT NULL'.
Column Values	VARCHAR	Value of the constraint column, in the same order in which Column Names contains the value of that column in the violating row. When interpreted as SQL, the value of this column forms a list of values of the same type as the columns in Column Names; for example: ( '1' ), ( '1', 'z' )

## Examples

Given the following inputs, Vertica returns one row, indicating one violation, because the same primary key value (10) was inserted into table t1 twice:

```
CREATE TABLE t1(c1 INT);
ALTER TABLE t1 ADD CONSTRAINT pk_t1 PRIMARY KEY (c1);
CREATE PROJECTION t1_p (c1) AS SELECT * FROM t1 UNSEGMENTED ALL NODES;
INSERT INTO t1 values (10);
INSERT INTO t1 values (10); --Duplicate primary key value
SELECT ANALYZE_CONSTRAINTS('t1');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | t1         | c1           | pk_t1          | PRIMARY        | ('10')
(1 row)
```

If the second INSERT statement above had contained any different value, the result would have been 0 rows (no violations).

In this example, create a table that contains 3 integer columns, one a unique key and one a primary key:

```
CREATE TABLE fact_1(
  f INTEGER,
  f_UK INTEGER UNIQUE,
  f_PK INTEGER PRIMARY KEY
);
```

Try issuing a command that refers to a nonexistent column:

```
SELECT ANALYZE_CONSTRAINTS('f_BB', 'f2');
ERROR: 'f_BB' is not a table name in the current search path
```

Insert some values into table `fact_1` and commit the changes:

```
INSERT INTO fact_1 values (1, 1, 1);
COMMIT;
```

Now issue the `ANALYZE_CONSTRAINTS` command on table `fact_1`. No constraint violations are expected and none are found:

```
SELECT ANALYZE_CONSTRAINTS('fact_1');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
(0 rows)
```

Now insert duplicate unique and primary key values and run `ANALYZE_CONSTRAINTS` on table `fact_1` again. The system shows two violations: one against the primary key and one against the unique key:

```
INSERT INTO fact_1 VALUES (1, 1, 1);
COMMIT;
SELECT ANALYZE_CONSTRAINTS('fact_1');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | fact_1     | f_pk         | -               | PRIMARY         | ('1')
 public      | fact_1     | f_uk         | -               | UNIQUE          | ('1')
(2 rows)
```

The following command looks for constraint validation on only the unique key in table `fact_1`:

```
SELECT ANALYZE_CONSTRAINTS('fact_1', 'f_UK');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | fact_1     | f_uk         | C_UNIQUE        | UNIQUE          | ('1')
(1 row)
```

The following example shows that you can specify the same column more than once; the function, however, returns the violation once only:

```
SELECT ANALYZE_CONSTRAINTS('fact_1', 'f_PK, F_PK');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | fact_1     | f_pk         | C_PRIMARY       | PRIMARY         | ('1')
(1 row)
```

The following example creates a new dimension table, `dim_1`, and inserts a foreign key and different (character) data types:

```
CREATE TABLE dim_1 (b VARCHAR(3), b_PK VARCHAR(4), b_FK INTEGER REFERENCES fact_1(f_PK));
```

Alter the table to create a multicolumn unique key and multicolumn foreign key and create superprojections:

```
ALTER TABLE dim_1 ADD CONSTRAINT dim_1_multiuk PRIMARY KEY (b, b_PK);
```

The following command inserts a missing foreign key (0) in table `dim_1` and commits the changes:

```
INSERT INTO dim_1 VALUES ('r1', 'Xpk1', 0);
COMMIT;
```

Checking for constraints on table dim\_1 detects a foreign key violation:

```
SELECT ANALYZE_CONSTRAINTS('dim_1');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | dim_1      | b_fk         | C_FOREIGN      | FOREIGN        | ('0')
(1 row)
```

Now add a duplicate value into the unique key and commit the changes:

```
INSERT INTO dim_1 values ('r2', 'Xpk1', 1);
INSERT INTO dim_1 values ('r1', 'Xpk1', 1);
COMMIT;
```

Checking for constraint violations on table dim\_1 detects the duplicate unique key error:

```
SELECT ANALYZE_CONSTRAINTS('dim_1');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | dim_1      | b, b_pk      | dim_1_multiuk  | PRIMARY        | ('r1', 'Xpk1')
 public      | dim_1      | b_fk         | C_FOREIGN      | FOREIGN        | ('0')
(2 rows)
```

Now create a table with multicolumn foreign key and create the superprojections:

```
CREATE TABLE dim_2(z_fk1 VARCHAR(3), z_fk2 VARCHAR(4));
ALTER TABLE dim_2 ADD CONSTRAINT dim_2_multifk FOREIGN KEY (z_fk1, z_fk2) REFERENCES dim_1(b, b_PK);
```

Now insert a foreign key that matches a foreign key in table dim\_1 and commit the changes:

```
INSERT INTO dim_2 VALUES ('r1', 'Xpk1');
COMMIT;
```

Checking for constraints on table dim\_2 detects no violations:

```
SELECT ANALYZE_CONSTRAINTS('dim_2');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
(0 rows)
```

Add a value that does not match and commit the change:

```
INSERT INTO dim_2 values ('r1', 'NONE');
COMMIT;
```

Checking for constraints on table dim\_2 detects a foreign key violation:

```
SELECT ANALYZE_CONSTRAINTS('dim_2');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | dim_2      | z_fk1, z_fk2 | dim_2_multifk  | FOREIGN        | ('r1', 'NONE')
(1 row)
```

Now analyze all constraints on all tables:

```
SELECT ANALYZE_CONSTRAINTS('');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | dim_1      | b, b_pk      | dim_1_multiuk  | PRIMARY        | ('r1', 'Xpk1')
 public      | dim_1      | b_fk         | C_FOREIGN      | FOREIGN        | ('0')
 public      | dim_2      | z_fk1, z_fk2 | dim_2_multifk  | FOREIGN        | ('r1', 'NONE')
 public      | fact_1     | f_pk         | C_PRIMARY      | PRIMARY        | ('1')
 public      | fact_1     | f_uk         | C_UNIQUE       | UNIQUE         | ('1')
(5 rows)
```

To quickly clean up your database, issue the following command:

```
DROP TABLE fact_1 cascade;
DROP TABLE dim_1 cascade;
```

```
DROP TABLE dim_2 cascade;
```

To learn how to remove violating rows, see the ***DISABLE\_DUPLICATE\_KEY\_ERROR*** (page 403) function.

### See Also

Adding Constraints in the Administrator's Guide

***COPY*** (page 607)

***ALTER TABLE*** (page 589)

***CREATE TABLE*** (page 664)

## ANALYZE\_HISTOGRAM

Works much like ***ANALYZE\_STATISTICS()*** (page 385) in that it collects and aggregates data samples and storage information from all nodes on which a projection is stored, then writes statistics to the catalog so that the data can be used by the query optimizer to recommend the best possible plan. The only difference is that ***ANALYZE\_HISTOGRAM*** lets you specify how much of the disk should be read, allowing you to optimize either for speed or accuracy during histogram collection.

### Syntax

```
ANALYZE_HISTOGRAM ( ( ' ' | '[ schema.]table' | 'column-name' ) [, ( percent ) ] )
```

### Return Value

- 0 - For success.
- 1 - For failure. Refer to `vertica.log` for details.

### Parameters

' '	Empty string. Collects statistics for all tables.
<i>[schema.]table</i>	Specifies the name of the table and optional schema and collects statistics for all projections of the specified table. <b>Note:</b> When using more than one schema, specify the schema that contains the projection.
<i>column-name</i>	Specifies the name of a single table column, typically a predicate column, and collects statistics for that column only.
<i>percent</i>	(Default 10) Specifies what fraction of the data to read from disk with a range of 1 to 100. The smaller the percentage, the faster the analysis; however, because less data is collected, sampling accuracy will be affected. <b>Note:</b> If you specify the percent parameter as 100, the entire projection is read from disk, and 128K rows are chosen at random. Otherwise, (row count) * (percent/100) rows are read from the projection in 100 contiguous bands with a minimum of

128K rows read. The one exception is if the column is first in the chosen projection's sort order, then all data is read from disk.
---

## Notes

- `ANALYZE_HISTOGRAM()` is a DDL operation that auto commits the current transaction, if any.
- You can cancel `ANALYZE_HISTOGRAM` mid analysis by issuing CTRL-C in a vsql shell or by invoking the **`INTERRUPT_STATEMENT()`** (page 430) function.
- **`ANALYZE_STATISTICS()`** (page 385) is an alias for `ANALYZE_HISTOGRAM()`; the only difference is `ANALYZE_HISTOGRAM` includes the percent parameter.
- To retrieve hints about underperforming queries and their root causes, use the **`ANALYZE_WORKLOAD()`** (page 386) function, which runs the Workload Analyzer and analyzes the workload history, identifying underperforming queries and their root causes. The function's result set returns a set of recommendations; for example `"run analyze_statistics on schema.table.column"`.

## Examples

In the following command, the system reads 10% of data from disk (default) and returns 0 for success:

```
=> SELECT ANALYZE_STATISTICS('shipping_dimension.shipping_key');
ANALYZE_STATISTICS
-----
                                0
(1 row)
```

The next command performs the same functionality as the previous `ANALYZE_STATISTICS()` command and returns 0 for success:

```
=> SELECT ANALYZE_HISTOGRAM('shipping_dimension.shipping_key');
ANALYZE_HISTOGRAM
-----
                                0
(1 row)
```

With the percent parameter specified as 100, the following command performs a full column scan and returns 0 for success:

```
=> SELECT ANALYZE_HISTOGRAM('shipping_dimension.shipping_key', 100);
ANALYZE_HISTOGRAM
-----
                                0
(1 row)
```

In this command, only 0.1% (1/1000) of the disk is read:

```
=> SELECT ANALYZE_HISTOGRAM('shipping_dimension.shipping_key', 0.1);
ANALYZE_HISTOGRAM
-----
                                0
(1 row)
```

For use cases, see [Collecting Statistics in the Administrator's Guide](#)

**See Also**

***ANALYZE\_STATISTICS*** (page 385)

***ANALYZE\_WORKLOAD*** (page 386)

***DROP\_STATISTICS*** (page 410)

***EXPORT\_STATISTICS*** (page 423)

***IMPORT\_STATISTICS*** (page 433)

***INTERRUPT\_STATEMENT*** (page 430)

## ANALYZE\_STATISTICS

Collects and aggregates data samples and storage information from all nodes on which a projection is stored, then writes statistics into the catalog so that the data can be used by the query optimizer to recommend the best possible plan. Without statistics, the query optimizer would assume uniform distribution of data values and equal storage usage for all projections.

ANALYZE\_STATISTICS() is an alias for **ANALYZE\_HISTOGRAM()** (page 382).

### Syntax

```
ANALYZE_STATISTICS [ ( ' ) | ( '[schema.]table' ) | ( 'column-name' ) ]
```

### Return Value

- 0 - For success.
- 1 - For failure. Refer to `vertica.log` for details.

### Parameters

' '	Empty string. Collects statistics for all tables.
<i>[schema.]table</i>	Specifies the name of the table and optional schema and collects statistics for all projections of the specified table. <b>Note:</b> When using more than one schema, specify the schema that contains the projection.
<i>column-name</i>	Specifies the name of a single table column, typically a predicate column. and collects statistics for only that column.

### Notes

- ANALYZE\_STATISTICS is a DDL operation that auto commits the current transaction, if any.
- Pre-join projection statistics are updated on any pre-joined tables.
- You can cancel ANALYZE\_STATISTICS mid analysis by issuing CTRL-C in a vsql shell or by invoking the **INTERRUPT\_STATEMENT()** (page 430) function.
- Always run ANALYZE\_STATISTICS on a table rather than a projection.
- Invoking ANALYZE\_STATISTICS() against very large tables with many projections could return results more slowly. To expedite things, you can run ANALYZE\_STATISTICS() on a single column. For example, it is a good idea to run statistics on a single column after you alter a table to add or drop a column.
- By default, more than one column is analyzed in a single query execution plan (subject to resource limits) in order to reduce the latency (time delay) of plan execution and help speed up analysis of relatively small tables with a large number of columns.
- To retrieve hints about underperforming queries and their root causes, use the **ANALYZE\_WORKLOAD()** (page 386) function, which runs the Workload Analyzer and analyzes the workload history, identifying underperforming queries and their root causes. The function's result set returns a set of recommendations; for example `run analyze_statistics on schema.table.column`. These tuning recommendations are provided so the database administrator can act upon them.

## Examples

The following command computes statistics on all projections in the Vmart database and returns 0 (success):

```
=> SELECT ANALYZE_STATISTICS ('');
analyze_statistics
-----
0
(1 row)
```

The following command computes statistics on a single table (`shipping_dimension`) and returns 0 (success):

```
=> SELECT ANALYZE_STATISTICS ('shipping_dimension');
analyze_statistics
-----
0
(1 row)
```

The following command computes statistics on a single important column (`shipping_key`) for all projections in the `shipping_dimension` table and returns 0 (success):

```
=> SELECT ANALYZE_STATISTICS('shipping_dimension.shipping_key');
analyze_statistics
-----
0
(1 row)
```

For use cases, see [Collecting Statistics in the Administrator's Guide](#)

## See Also

***ANALYZE\_WORKLOAD*** (page 386)

***DROP\_STATISTICS*** (page 410)

***EXPORT\_STATISTICS*** (page 423)

***IMPORT\_STATISTICS*** (page 433)

***INTERRUPT\_STATEMENT*** (page 430)

## ANALYZE\_WORKLOAD

Runs the Workload Analyzer, a utility that analyzes the system workload history, resources, and configurations to identify underperforming queries, as well as the root causes for poor query performance. The function returns a set of tuning recommendations for all events in 'scope' since the time specified. If you don't specify a time, the function returns recommendations on events since the last recorded invocation of `ANALYZE_WORKLOAD()`. The tuning recommendations are provided so the database administrator can act upon them.

### Syntax 1

```
ANALYZE_WORKLOAD ( 'scope' , [ true ] );
```

## Syntax 2

```
ANALYZE_WORKLOAD ( 'scope' , 'since_time' );
```

### Parameters

<i>scope</i>	Specifies which catalog objects to analyze. Can be one of: <ul style="list-style-type: none"> <li>▪ " (empty string) returns recommendations for all objects in database</li> <li>▪ 'table_name' returns all recommendations related to a table</li> <li>▪ 'schema_name' returns recommendations on all objects in a schema</li> </ul>
<i>since_time</i>	Limits the recommendations from all events of 'scope' since the specified time up to the current system status. When omitted, the function returns the recommendations on events since the last recorded invocation of ANALYZE_WORKLOAD. <b>Note:</b> Strings used for this parameter must be explicitly cast to TIMESTAMP or TIMESTAMPTZ. See the Examples section below.
<i>true</i>	Specifies that this analysis invocation will be recorded in the system. The default value is false (do not record in system). If recorded, subsequent calls to ANALYZE_WORKLOAD analyze only the events that have occurred since this recorded time; all prior events are ignored.

### Return Value

ANALYZE\_WORKLOAD() returns aggregated tuning recommendations, as described in the following table. See the Notes section below this table for example output for the parameters:

Column	Data type	Description
<i>observation_count</i>	INTEGER	An integer for the total number of events observed for this tuning recommendation.
<i>first_observation_time</i>	TIMESTAMP TZ	The timestamp for when the event first occurred. If null, the tuning recommendation is from the current status of the system instead of from any event.
<i>last_observation_time</i>	TIMESTAMP TZ	The timestamp for when the event last occurred. If null, the tuning recommendation is from the current status of the system instead of from any event.
<i>tuning_parameter</i>	VARCHAR	The objects on which tuning action will be applied.
<i>tuning_description</i>	VARCHAR	Returns the tuning recommendation from the Workload Analyzer.
<i>tuning_command</i>	VARCHAR	Returns a command string if tuning action is a SQL command.

<i>tuning_cost</i>	VARCHAR	The cost of performing the recommended tuning operation. Is one of: <ul style="list-style-type: none"> <li>▪ LOW</li> <li>▪ MEDIUM</li> <li>▪ HIGH.</li> </ul>
--------------------	---------	--

## Notes

- If the 'scope' you enter is invalid, the system returns an error:  

```
ERROR: Invalid scope in ANALYZE_WORKLOAD: schema or table my_table does not exist
```
- An example return value for *tuning\_parameter* is `schema.table.column`; for example:  
`store.store_orders_fact.date_ordered`  
The above identifies the `date_ordered` column as the column that needs tuning. This is useful because it means the DBA can run `ANALYZE_STATISTICS` on a single column, rather than on an entire table.
- Example values returned for the *tuning\_description* column are:
  - Run database designer on table `public.t`
  - Create replicated projection for table `public.vendor_dimension`
  - Consider query-specific design on query
  - Reset configuration parameter with `set_config_parameter()`
  - Re-segment projection `public.p_t` on high-cardinality column(s)
- Example return values for the *tuning\_command* column are listed below. The first statement tells the user to update statistics on a particular `schema.table.column`:  

```
SELECT ANALYZE_STATISTICS('public.tablename.column-name');
```

The next statement indicates a mismatched configuration parameter 'LockTimeout':  

```
SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name = 'LockTimeout';
```

## Examples

The following statement returns recommendations for all objects in all schemas. Analysis is based on the events since the last recorded analysis:

```
=> SELECT ANALYZE_WORKLOAD('', true);
```

The `true` setting tells the Workload Analyzer to record this analysis invocation. The next invocation of `ANALYZE_WORKLOAD()` analyzes events from this point forward.

The next statement instructs the Workload Analyzer to save all events for table `t1`:

```
=> \x
Expanded display is on.
=> SELECT ANALYZE_WORKLOAD('t1');
```

```
-[ RECORD 1 ]-----+-----
observation_count      | 1
first_observation_time |
```

```

last_observation_time |
tuning_parameter      | public.t1
tuning_description    | run database designer on table public.t1
tuning_command        |
tuning_cost           | HIGH

```

**Note:** Null observation times shown above indicate that the tuning recommendation is from the current status of the system instead of from any event.

The recommendation output can also be queried using the **V\_MONITOR.TUNING\_RECOMMENDATIONS** (page 899) system table:

```
=> SELECT * FROM tuning_recommendations;
```

```

-[ RECORD 1 ]-----+-----
observation_count    | 1
first_observation_time |
last_observation_time |
tuning_parameter     | public.T1
tuning_description   | run database designer on table public.T1
tuning_command       |
tuning_cost          | HIGH

```

This statement analyzes workloads on all tables one week before today (now):

```
=> SELECT ANALYZE_WORKLOAD('', NOW() - INTERVAL '1 week');
```

```

-[ RECORD 1
]-----+-----
observation_count    | 4
first_observation_time | 2011-04-07 13:57:17.799003-04
last_observation_time | 2011-04-12 12:05:26.856456-04
tuning_parameter     | store.store_orders_fact.date_ordered
tuning_description   | analyze statistics on table column
store.store_orders_fact.date_ordered
tuning_command       | select
analyze_statistics('store.store_orders_fact.date_ordered');
tuning_cost          | MEDIUM
-[ RECORD 14
]-----+-----
observation_count    | 2
first_observation_time | 2011-04-12 17:52:03.022644-04
last_observation_time | 2011-04-12 17:52:03.02301-04
tuning_parameter     | SELECT x FROM t WHERE x > (SELECT SUM(DISTINCT x) FROM
| t GROUP BY y) OR x < 9;
tuning_description   | consider query-specific design on query
tuning_command       |
tuning_cost          | HIGH

```

Note that the Workload Analyzer found two issues in the above results:

- In RECORD 1 the date\_ordered column in the store.store\_orders\_fact table. The Workload Analyzer recommends that the DBA run ANALYZE\_STATISTICS() on that column. The function also returns the query the DBA can run, which is output in the tuning\_command column.

- In RECORD 2 there is an under performing query. The Workload Analyzer recommends that the DBA consider using the Database Designer to run a query-specific design on that particular query. The cost is HIGH, which lets the DBD know he or she might want to let the query run after hours.

The following four statements are some of the different ways to express the same query. All return the same results for workloads on table t1 since October 4, 2010.

```
=> SELECT ANALYZE_WORKLOAD('t1', TIMESTAMP '2010-10-04 11:18:15');
=> SELECT ANALYZE_WORKLOAD('t1', '2010-10-04 11:18:15'::TIMESTAMPTZ);
=> SELECT ANALYZE_WORKLOAD('t1', 'October 4, 2010'::TIMESTAMP);
=> SELECT ANALYZE_WORKLOAD('t1', '10-04-10'::TIMESTAMPTZ);
```

This statement analyzes workloads on all tables two days before today (now):

```
=> SELECT ANALYZE_WORKLOAD('', NOW() - '2 DAYS'::INTERVAL);
```

The results above indicate an issue with the customer\_state column in the customer\_dimension table and recommends that the DBA run ANALYZE\_STATISTICS() on that column.

```
-[ RECORD 1
]-----+-----
observation_count      | 2
first_observation_time | 2011-04-12 11:13:57.568675-04
last_observation_time  | 2011-04-12 12:05:08.200439-04
tuning_parameter       | public.customer_dimension.customer_state
tuning_description     | analyze statistics on table column public.customer_dimen
                       | sion.customer_state
tuning_command         | select
analyze_statistics('public.customer_dimension.cu
                       | stomer_state');
tuning_cost             | MEDIUM
```

## See Also

**V\_MONITOR.TUNING\_RECOMMENDATIONS** (page 899)

Analyzing Workloads in the Administrator's Guide

## AUDIT

Estimates the raw data size of a database, a schema, a projection, or a table as it is counted in an audit of the database size. AUDIT estimates the size using the same data sampling method as the audit that Vertica performs to determine if a database is compliant with any database size allowances in its license. The results of this function are not considered when Vertica determines whether the size of the database complies with the Vertica license's data allowance. See How Vertica Calculates Database Size in the Administrator's Guide for details.

**Note:** This function can only audit the size of tables, schemas, and databases which the user has permission to access. If a non-superuser attempts to audit the entire database, the audit will only estimate the size of the data that the user is allowed to read.

## Syntax

```
AUDIT([name] [, granularity] [, error_tolerance [, confidence_level]])
```

## Parameters

<i>name</i>	A string containing the name of the schema, projection, or table to be audited. If this string is empty (''), the entire database is audited.
<i>granularity</i>	<p>A string containing the level at which the audit results are reported. The recognized levels are:</p> <ul style="list-style-type: none"> <li>▪ 'schema'</li> <li>▪ 'table'</li> <li>▪ 'projection'</li> </ul> <p>By default, the granularity is the same level as <i>name</i>. For example, if <i>name</i> is a schema, then the size of the entire schema is reported. If you instead specify 'table' as the granularity, AUDIT reports the size of each table in the schema. The granularity must be finer than that of object: specifying 'schema' for an audit of a table has no effect.</p> <p>The results of an audit with a granularity are reported in the V_CATALOG.USER_AUDITS system table.</p>
<i>error_tolerance</i>	<p>The percent margin of error allowed in the estimation, as an decimal number between 0 and 100. The default value is 5, for a 5% margin of error.</p> <p><b>Note:</b> The lower this value is, the more resources the audit uses since it will perform more data sampling. Setting this value to 0 results in a full audit of the database, which is very resource intensive, as all of the data in the database is analyzed. Doing a full audit of the database significantly impacts performance and is not recommended on a production database.</p>
<i>confidence_level</i>	<p>The statistical confidence level percentage of the estimate, as a decimal between 0 and 100. The default is 99, for a confidence level of 99%.</p> <p><b>Note:</b> The higher this value is, the more resources the function uses since it will perform more data sampling. Setting this value to 1 results in a full audit of the database, which is very resource intensive, as all of the database is analyzed. Doing a full audit of the database significantly impacts performance and is not recommended on a production database.</p>

## Notes

Due to the iterative sampling used in the auditing process, making the error tolerance a small fraction of a percent (0.00001, for example) can cause the AUDIT function to run for a longer period than a full database audit.

## Examples

To audit the entire database:

```
=> SELECT AUDIT('');
      AUDIT
-----
      76376696
(1 row)
```

To audit the database with a 25% error tolerance:

```
=> SELECT AUDIT('',25);
      AUDIT
-----
      75797126
(1 row)
```

To audit the database with a 25% level of tolerance and a 90% confidence level:

```
=> SELECT AUDIT('',25,90);
      AUDIT
-----
      76402672
(1 row)
```

To audit just the online\_sales schema in the VMart example database:

```
VMart=> SELECT AUDIT('online_sales');
      AUDIT
-----
      35716504
(1 row)
```

To audit the online\_sales schema and report the results by table:

```
=> SELECT AUDIT('online_sales','table');
      AUDIT
-----
      See table sizes in v_catalog.user_audits for schema online_sales
(1 row)

=> \x
Expanded display is on.

=> SELECT * FROM user_audits WHERE object_schema = 'online_sales';
-[ RECORD 1 ]-----+-----
size_bytes          | 64960
user_id             | 45035996273704962
user_name           | dbadmin
```

```

object_id          | 45035996273717636
object_type        | TABLE
object_schema      | online_sales
object_name        | online_page_dimension
audit_start_timestamp | 2011-04-05 09:24:48.224081-04
audit_end_timestamp | 2011-04-05 09:24:48.337551-04
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling      | f
confidence_interval_lower_bound_bytes | 64960
confidence_interval_upper_bound_bytes | 64960
sample_count       | 0
cell_count         | 0
-[ RECORD 2 ]-----+-----
size_bytes         | 20197
user_id            | 45035996273704962
user_name          | dbadmin
object_id          | 45035996273717640
object_type        | TABLE
object_schema      | online_sales
object_name        | call_center_dimension
audit_start_timestamp | 2011-04-05 09:24:48.340206-04
audit_end_timestamp | 2011-04-05 09:24:48.365915-04
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling      | f
confidence_interval_lower_bound_bytes | 20197
confidence_interval_upper_bound_bytes | 20197
sample_count       | 0
cell_count         | 0
-[ RECORD 3 ]-----+-----
size_bytes         | 35614800
user_id            | 45035996273704962
user_name          | dbadmin
object_id          | 45035996273717644
object_type        | TABLE
object_schema      | online_sales
object_name        | online_sales_fact
audit_start_timestamp | 2011-04-05 09:24:48.368575-04
audit_end_timestamp | 2011-04-05 09:24:48.379307-04
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling      | t
confidence_interval_lower_bound_bytes | 34692956
confidence_interval_upper_bound_bytes | 36536644
sample_count       | 10000
cell_count         | 9000000

```

## AUDIT\_LICENSE\_SIZE

Triggers an immediate audit of the database size to determine if it is in compliance the raw data storage allowance included in your Vertica license. The audit is performed in the background, so this function call returns immediately. To see the results of the audit when it is done, use the **GET\_COMPLIANCE\_STATUS** (page 425) function.

### Syntax

```
AUDIT_LICENSE_SIZE()
```

### Example

```
=> SELECT audit_license_size();
   audit_license_size
-----
Service hurried
(1 row)
```

## AUDIT\_LICENSE\_TERM

Triggers an immediate audit to determine if the Vertica license has expired. The audit happens in the background, so this function returns immediately. To see the result of the audit, use the **GET\_COMPLIANCE\_STATUS** (page 425) function.

### Syntax

```
AUDIT_LICENSE_TERM()
```

### Example

```
=> SELECT AUDIT_LICENSE_TERM();
   AUDIT_LICENSE_TERM
-----
Service hurried
(1 row)
```

## CANCEL\_REFRESH

Cancels refresh-related internal operations initiated by **START\_REFRESH()**.

### Syntax

```
CANCEL_REFRESH()
```

### Notes

- Refresh tasks run in a background thread in an internal session, so you cannot use **INTERRUPT\_STATEMENT** (page 430) to cancel those statements. Instead, use **CANCEL\_REFRESH** to cancel statements that are run by refresh-related internal sessions.
- Run **CANCEL\_REFRESH()** on the same node on which **START\_REFRESH()** was initiated.
- **CANCEL\_REFRESH()** cancels the refresh operation running on a node, waits for the cancelation to complete, and returns **SUCCESS**.

- Only one set of refresh operations runs on a node at any time.

### See Also

***INTERRUPT\_STATEMENT*** (page 430)

***SESSIONS*** (page 888)

***START\_REFRESH*** (page 470)

***PROJECTION\_REFRESHES*** (page 863)

## **CLEAR\_QUERY\_REPOSITORY**

Triggers Vertica to clear query data from the query repository immediately.

### Syntax

```
CLEAR_QUERY_REPOSITORY()
```

### Notes

Before using this function:

- 1 Note the value of the QueryRepoRetentionTime parameter.
- 2 Set the QueryRepoRetentionTime parameter to zero (0). (See Configuring Query Repository in the Troubleshooting Guide.)

```
=> SELECT SET_CONFIG_PARAMETER('QueryRepoRetentionTime', '0');
```

Once you have cleared the query repository, set the QueryRepoRetentionTime parameter back to the original value (before you changed it to zero). The default value is 100.

### Example

```
SELECT CLEAR_QUERY_REPOSITORY();
   CLEAR_QUERY_REPOSITORY
-----
Query Repository Cleaned
(1 row)
```

### See Also

***V\_MONITOR.QUERY\_METRICS*** (page 868)

***V\_MONITOR.QUERY\_PROFILES*** (page 870)

Collecting Query Information in the Troubleshooting Guide

Configuration Parameters in the Administrator's Guide

## **CLEAR\_PROJECTION\_REFRESHES**

Triggers Vertica to clear information about refresh operations for projections immediately.

### Syntax

```
CLEAR_PROJECTION_REFRESHES()
```

## Notes

Information about a refresh operation—whether successful or unsuccessful—is maintained in the **PROJECTION\_REFRESHES** (page 863) system table until either the **CLEAR\_PROJECTION\_REFRESHES** (page 395)() function is executed or the storage quota for the table is exceeded. The **PROJECTION\_REFRESHES.IS\_EXECUTING** column returns a boolean value that indicates whether the refresh is currently running (t) or occurred in the past (f).

## Example

To immediately purge projection refresh history, use the **CLEAR\_PROJECTION\_REFRESHES**() function:

```
=> SELECT CLEAR_PROJECTION_REFRESHES ();
      CLEAR_PROJECTION_REFRESHES
-----
      CLEAR
      (1 row)
```

Only the rows where the **PROJECTION\_REFRESHES.IS\_EXECUTING** column equals false are cleared.

## See Also

**PROJECTION\_REFRESHES** (page 863)

**REFRESH** (page 446)

**START\_REFRESH** (page 470)

Clearing **PROJECTION\_REFRESHES** History in the Administrator's Guide

## CLEAR\_RESOURCE\_REJECTIONS

Clears the content of the **RESOURCE\_REJECTIONS** (page 882) and **DISK\_RESOURCE\_REJECTIONS** (page 843) system tables. Normally, these tables are only cleared during a node restart. This function lets you clear the tables whenever you need. For example, you may want to clear the tables after having resolved a disk space issue that caused disk resource rejections.

## CLOSE\_SESSION

Interrupts the specified external session, rolls back the current transaction, if any, and closes the socket.

## Syntax

```
CLOSE_SESSION ( 'sessionid' )
```

## Parameters

<i>sessionid</i>	A string that specifies the session to close. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
------------------	--

## Notes

- Closing of the session is processed asynchronously. It could take some time for the session to be closed. Check the **SESSIONS** (page 888) table for the status.
- Database shutdown is prevented if new sessions connect after the `CLOSE_SESSION()` command is invoked (and before the database is actually shut down. See **Controlling Sessions** below.

## Messages

The following are the messages you could encounter:

- For a badly formatted sessionID  
`close_session | Session close command sent. Check SESSIONS for progress.`  
`Error: invalid Session ID format`
- For an incorrect sessionID parameter  
`Error: Invalid session ID or statement key`

## Examples

User session opened. RECORD 2 shows the user session running COPY DIRECT statement.

```
vmartdb=> SELECT * FROM sessions;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (SELECT * FROM sessions;)
statement_start    | 2011-01-03 15:36:13.896288
statement_id       | 10
last_statement_duration_us | 14978
current_statement  | select * from sessions;
ssl_state          | None
authentication_method | Trust
-[ RECORD 2 ]-----+-----
node_name          | v_vmartdb_node0002
user_name          | dbadmin
client_hostname    | 127.0.0.1:57174
client_pid         | 30117
login_timestamp    | 2011-01-03 15:33:00.842021-05
session_id         | stress05-27944:0xcla
client_label       |
transaction_start  | 2011-01-03 15:34:46.538102
transaction_id     | -1
transaction_description | user dbadmin (COPY ClickStream_Fact FROM
'/data/clickstream/1g/ClickStream_Fact.tbl'
DELIMITER '|' NULL '\\n' DIRECT;)
statement_start    | 2011-01-03 15:34:46.538862
```

```
statement_id |
last_statement_duration_us | 26250
current_statement | COPY ClickStream_Fact FROM '/data/clickstream
                    /1g/ClickStream_Fact.tbl' DELIMITER '|' NULL
                    '\\n' DIRECT;
ssl_state | None
authentication_method | Trust
```

**Close user session stress05-27944:0xc1a**

```
vmartdb=> \x
Expanded display is off.
vmartdb=> SELECT CLOSE_SESSION('stress05-27944:0xc1a');
                CLOSE_SESSION
```

```
-----
Session close command sent. Check v_monitor.sessions for progress.
(1 row)
```

Query the sessions table again for current status, and you can see that the second session has been closed:

```
=> SELECT * FROM SESSIONS;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (select * from SESSIONS;)
statement_start    | 2011-01-03 16:12:07.841298
statement_id       | 20
last_statement_duration_us | 2099
current_statement  | SELECT * FROM SESSIONS;
ssl_state          | None
authentication_method | Trust
```

**Controlling Sessions**

The database administrator must be able to disallow new incoming connections in order to shut down the database. On a busy system, database shutdown is prevented if new sessions connect after the `CLOSE_SESSION` or `CLOSE_ALL_SESSIONS()` command is invoked — and before the database actually shuts down.

One option is for the administrator to issue the `SHUTDOWN('true')` command, which forces the database to shut down and disallow new connections. See **SHUTDOWN** (page 469) in the SQL Reference Manual.

Another option is to modify the `MaxClientSessions` parameter from its original value to 0, in order to prevent new non-dbadmin users from connecting to the database.

- 1 Determine the original value for the `MaxClientSessions` parameter by querying the `V_MONITOR.CONFIGURATIONS_PARAMETERS` (page 832) system table:

```
=> SELECT CURRENT_VALUE FROM CONFIGURATION_PARAMETERS WHERE
       parameter_name='MaxClientSessions';
       CURRENT_VALUE
```

```
-----
       50
(1 row)
```

- 2 Set the `MaxClientSessions` parameter to 0 to prevent new non-dbadmin connections:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 0);
```

**Note:** The previous command allows up to five administrators to log in.

- 3 Issue the `CLOSE_ALL_SESSIONS()` command to remove existing sessions:

```
=> SELECT CLOSE_ALL_SESSIONS();
```

- 4 Query the `SESSIONS` table:

```
=> SELECT * FROM SESSIONS;
```

When the session no longer appears in the `SESSIONS` table, disconnect and run the Stop Database command.

- 5 Restart the database.

- 6 Restore the `MaxClientSessions` parameter to its original value:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 50);
```

### See Also

**CLOSE\_ALL\_SESSIONS** (page 399), **CONFIGURATION\_PARAMETERS** (page 832), **SESSIONS** (page 888), **SHUTDOWN** (page 469)

Managing Sessions and Configuration Parameters in the Administrator's Guide

Shutdown Problems in the Troubleshooting Guide

## CLOSE\_ALL\_SESSIONS

Closes all external sessions except the one issuing the `CLOSE_ALL_SESSIONS` functions.

### Syntax

```
CLOSE_ALL_SESSIONS()
```

### Notes

Closing of the sessions is processed asynchronously. It might take some time for the session to be closed. Check the **SESSIONS** (page 888) table for the status.

Database shutdown is prevented if new sessions connect after the `CLOSE_SESSION` or `CLOSE_ALL_SESSIONS()` command is invoked (and before the database is actually shut down). See **Controlling Sessions** below.

### Message

```
close_all_sessions | Close all sessions command sent.
```

Check SESSIONS for progress.

## Examples

Two user sessions opened, each on a different node:

```
vmartdb=> SELECT * FROM sessions;
-[ RECORD 1
]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (select * from sessions;)
statement_start    | 2011-01-03 15:36:13.896288
statement_id       | 10
last_statement_duration_us | 14978
current_statement  | select * from sessions;
ssl_state          | None
authentication_method | Trust
-[ RECORD 2
]-----+-----
node_name          | v_vmartdb_node0002
user_name          | dbadmin
client_hostname    | 127.0.0.1:57174
client_pid         | 30117
login_timestamp    | 2011-01-03 15:33:00.842021-05
session_id         | stress05-27944:0xc1a
client_label       |
transaction_start  | 2011-01-03 15:34:46.538102
transaction_id     | -1
transaction_description | user dbadmin (COPY Mart_Fact FROM
'/data/mart_Fact.tbl'
                                DELIMITER '|' NULL '\\\n';)
statement_start    | 2011-01-03 15:34:46.538862
statement_id       |
last_statement_duration_us | 26250
current_statement  | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER
'|'
                                NULL '\\\n';
ssl_state          | None
authentication_method | Trust
-[ RECORD 3
]-----+-----
node_name          | v_vmartdb_node0003
user_name          | dbadmin
client_hostname    | 127.0.0.1:56367
client_pid         | 1191
login_timestamp    | 2011-01-03 15:31:44.939302-05
session_id         | stress06-25663:0xbec
```

```

client_label          |
transaction_start    | 2011-01-03 15:34:51.05939
transaction_id       | 54043195528458775
transaction_description | user dbadmin (COPY Mart_Fact FROM
'/data/Mart_Fact.tbl'
                    DELIMITER '|' NULL '\\n' DIRECT;);
statement_start      | 2011-01-03 15:35:46.436748
statement_id         |
last_statement_duration_us | 1591403
current_statement    | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER
'|'
                    NULL '\\n' DIRECT;
ssl_state            | None
authentication_method | Trust

```

### Close all sessions:

```

vmartdb=> \x
Expanded display is off.
vmartdb=> SELECT CLOSE_ALL_SESSIONS () ;
                CLOSE_ALL_SESSIONS

```

-----

Close all sessions command sent. Check v\_monitor.sessions for progress.  
(1 row)

### Sessions contents after issuing the CLOSE\_ALL\_SESSIONS() command:

```

=> SELECT * FROM SESSIONS;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (SELECT * FROM sessions;)
statement_start    | 2011-01-03 16:19:56.720071
statement_id       | 25
last_statement_duration_us | 15605
current_statement  | SELECT * FROM SESSIONS;
ssl_state          | None
authentication_method | Trust

```

## Controlling Sessions

The database administrator must be able to disallow new incoming connections in order to shut down the database. On a busy system, database shutdown is prevented if new sessions connect after the `CLOSE_SESSION` or `CLOSE_ALL_SESSIONS()` command is invoked — and before the database actually shuts down.

One option is for the administrator to issue the `SHUTDOWN('true')` command, which forces the database to shut down and disallow new connections. See **SHUTDOWN** (page 469) in the SQL Reference Manual.

Another option is to modify the `MaxClientSessions` parameter from its original value to 0, in order to prevent new non-dbadmin users from connecting to the database.

- 1 Determine the original value for the `MaxClientSessions` parameter by querying the `V_MONITOR.CONFIGURATIONS_PARAMETERS` (page 832) system table:

```
=> SELECT CURRENT_VALUE FROM CONFIGURATION_PARAMETERS WHERE
      parameter_name='MaxClientSessions';
      CURRENT_VALUE
-----
          50
(1 row)
```

- 2 Set the `MaxClientSessions` parameter to 0 to prevent new non-dbadmin connections:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 0);
```

**Note:** The previous command allows up to five administrators to log in.

- 3 Issue the `CLOSE_ALL_SESSIONS()` command to remove existing sessions:

```
=> SELECT CLOSE_ALL_SESSIONS();
```

- 4 Query the `SESSIONS` table:

```
=> SELECT * FROM SESSIONS;
```

When the session no longer appears in the `SESSIONS` table, disconnect and run the Stop Database command.

- 5 Restart the database.

- 6 Restore the `MaxClientSessions` parameter to its original value:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 50);
```

### See Also

**CLOSE\_SESSION** (page 396)

**CONFIGURATION\_PARAMETERS** (page 832)

**SHUTDOWN** (page 469)

**V\_MONITOR.SESSIONS** (page 888)

Managing Sessions and Configuration Parameters in the Administrator's Guide

Shutdown Problems in the Troubleshooting Guide

### CURRENT\_SCHEMA

Shows the resolved name of `$User`.

### Behavior Type

Stable

### Syntax

```
CURRENT_SCHEMA()
```

**Notes**

If the search path for USER1 is: \$USER, COMMON, PUBLIC:

SELECT CURRENT\_SCHEMA() returns the following output if schema USER1 exists:

```
USER1
```

If schema USER1 does not exist, it returns the following output:

```
COMMON
```

**Example**

```
SELECT CURRENT_SCHEMA();
  current_schema
-----
  public
(1 row)
```

**DISABLE\_DUPLICATE\_KEY\_ERROR**

Disables error messaging when Vertica finds duplicate PRIMARY KEY/UNIQUE KEY values at run time. Queries execute as though no constraints are defined on the schema. Effects are session scoped.

**CAUTION:** When called, DISABLE\_DUPLICATE\_KEY\_ERROR() suppresses data integrity checking and can lead to incorrect query results. Use this function only after you insert duplicate primary keys into a dimension table in the presence of a prejoin projection. Then correct the violations and turn integrity checking back on with **REENABLE\_DUPLICATE\_KEY\_ERROR** (page 446)().

**Syntax**

```
DISABLE_DUPLICATE_KEY_ERROR();
```

**Notes**

The following series of commands create a table named `dim` and the corresponding projection:

```
CREATE TABLE dim (pk INTEGER PRIMARY KEY, x INTEGER);
CREATE PROJECTION dim_p (pk, x) AS SELECT * FROM dim ORDER BY x UNSEGMENTED ALL
NODES;
```

The next two statements create a table named `fact` and the pre-join projection that joins `fact` to `dim`.

```
CREATE TABLE fact (fk INTEGER REFERENCES dim(pk));
CREATE PROJECTION prejoin_p (fk, pk, x) AS SELECT * FROM fact, dim WHERE pk=fk ORDER
BY x;
```

The following statements load values into table `dim`. Notice the last statement inserts a duplicate primary key value of 1:

```
INSERT INTO dim values (1,1);
INSERT INTO dim values (2,2);
INSERT INTO dim values (1,2); --Constraint violation
COMMIT;
```

Table `dim` now contains duplicate primary key values, but you cannot delete the violating row because of the presence of the pre-join projection. Any attempt to delete the record results in the following error message:

```
ROLLBACK: Duplicate primary key detected in FK-PK join Hash-Join (x dim_p), value
1
```

In order to remove the constraint violation (`pk=1`), use the following sequence of commands, which puts the database back into the state just before the duplicate primary key was added.

To remove the violation:

- 1 First save the original `dim` rows that match the duplicated primary key.

```
CREATE TEMP TABLE dim_temp(pk integer, x integer);
INSERT INTO dim_temp SELECT * FROM dim WHERE pk=1 AND x=1; -- original
dim row
```

- 2 Temporarily disable error messaging on duplicate constraint values:

```
SELECT DISABLE_DUPLICATE_KEY_ERROR();
```

**Caution:** Remember that issuing this command suppresses the enforcement of data integrity checking.

- 3 Remove the the original row that contains duplicate values:

```
DELETE FROM dim WHERE pk=1;
```

- 4 Allow the database to resume data integrity checking:

```
SELECT REENABLE_DUPLICATE_KEY_ERROR();
```

- 5 Reinsert the original values back into the dimension table:

```
INSERT INTO dim SELECT * from dim_temp;
COMMIT;
```

- 6 Validate your dimension and fact tables.

If you receive the following error message, it means that the duplicate records you want to delete are not identical. That is, the records contain values that differ in at least one column that is not a primary key; for example, (1,1) and (1,2).

```
ROLLBACK: Delete: could not find a data row to delete (data integrity violation?)
```

The difference between this message and the rollback message in the previous example is that a fact row contains a foreign key that matches the duplicated primary key, which has been inserted. Thus, a row with values from the fact and dimension table is now in the pre-join projection. In order for the DELETE statement (Step 3 in the following example) to complete successfully, extra predicates are required to identify the original dimension table values (the values that are in the pre-join).

This example is nearly identical to the previous example, except that an additional INSERT statement joins the fact table to the dimension table by a primary key value of 1:

```
INSERT INTO dim values (1,1);
INSERT INTO dim values (2,2);
INSERT INTO fact values (1); -- New insert statement joins fact with dim on
primary key value=1
INSERT INTO dim values (1,2); -- Duplicate primary key value=1
COMMIT;
```

To remove the violation:

- 1 First save the original `dim` and `fact` rows that match the duplicated primary key:

```
CREATE TEMP TABLE dim_temp(pk integer, x integer);
CREATE TEMP TABLE fact_temp(fk integer);
INSERT INTO dim_temp SELECT * FROM dim WHERE pk=1 AND x=1; -- original
dim row
INSERT INTO fact_temp SELECT * FROM fact WHERE fk=1;
```

- 2 Temporarily suppresses the enforcement of data integrity checking:

```
SELECT DISABLE_DUPLICATE_KEY_ERROR();
```

- 3 Remove the duplicate primary keys. These steps implicitly remove all fact rows with the matching foreign key, as well.

- a) Remove the the original row that contains duplicate values:

```
DELETE FROM dim WHERE pk=1 AND x=1;
```

Note: The extra predicate (`x=1`) specifies removal of the original (1, 1) row, rather than the newly inserted (1, 2) values that caused the violation.

- b) Remove all remaining rows:

```
DELETE FROM dim WHERE pk=1;
```

- 4 Turn on integrity checking:

```
SELECT REENABLE_DUPLICATE_KEY_ERROR();
```

- 5 Reinsert the original values back into the fact and dimension table:

```
INSERT INTO dim SELECT * from dim_temp;
INSERT INTO fact SELECT * from fact_temp;
COMMIT;
```

- 6 Validate your dimension and fact tables.

## See Also

***ANALYZE\_CONSTRAINTS*** (page 377)

***REENABLE\_DUPLICATE\_KEY\_ERROR*** (page 446)

## DISABLE\_LOCAL\_SEGMENTS

Disable local data segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes. See Local Data Segmentation in the Administrator's Guide for details.

### Syntax

```
DISABLE_LOCAL_SEGMENTS()
```

### Example

```
=> SELECT DISABLE_LOCAL_SEGMENTS();
DISABLE_LOCAL_SEGMENTS
-----
DISABLED
(1 row)
```

## DISPLAY\_LICENSE

Returns the terms of your Vertica license. This includes the start and end dates for which the license is valid (or "Perpetual" if the license has no expiration), and the amount of data your database can store.

### Syntax

```
DISPLAY_LICENSE()
```

### Examples

```
=> SELECT DISPLAY_LICENSE();
          DISPLAY_LICENSE
```

```
-----
Vertica Systems, Inc.
1/1/2011
12/31/2011
30
50TB
```

```
(1 row)
```

## DO\_TM\_TASK

Runs a Tuple Mover operation (moveout) on one or more projections defined on the specified table. You do not need to stop the Tuple Mover to run this function.

### Syntax

```
DO_TM_TASK ( 'task' [ , '[ schema.]table' | 'projection' ] )
```

### Parameters

<i>task</i>	Is one of the following tuple mover operations: <ul style="list-style-type: none"><li>▪ 'moveout' — Moves out all projections on the specified table (if a particular projection is not specified).</li><li>▪ 'analyze_row_count' — Automatically collects the number of rows in a projection every 60 seconds and aggregates row counts calculated during loads.</li></ul>
<i>[schema.]table</i>	Runs a tuple mover operation for all projections within the specified table. When using more than one schema, specify the schema that contains the table with the projections you want to affect.
<i>projection</i>	If <i>projection</i> is not passed as an argument, all projections in the system are used. If <i>projection</i> is specified, DO_TM_TASK looks for a projection of that name and, if found, uses it; if a named projection is not found, the function looks for a table with that name and, if found, moves out all projections on that table.

## Notes

`DO_TM_TASK()` is useful because you can move out all projections from a table or database without having to name each projection individually.

## Examples

The following example performs a moveout of all projections for table `t1`:

```
=> SELECT DO_TM_TASK('moveout', 't1');
```

The following example performs a moveout for projections `t1_p`:

```
=> SELECT DO_TM_TASK('moveout', 't1_p')
```

## See Also

***COLUMN\_STORAGE*** (page 830)

***DROP\_PARTITION*** (page 408)

***DUMP\_PARTITION\_KEYS*** (page 414)

***DUMP\_PROJECTION\_PARTITION\_KEYS*** (page 415)

***DUMP\_TABLE\_PARTITION\_KEYS*** (page 416)

***PARTITION\_PROJECTION*** (page 440)

Partitioning Tables in the Administrator's Guide

Collecting Statistics in the Administrator's Guide

## DROP\_LOCATION

Removes the specified storage location.

### Syntax

```
DROP_LOCATION ( 'path' , 'site' )
```

### Parameters

<i>path</i>	Specifies where the storage location to drop is mounted.
<i>site</i>	Is the Vertica site where the location is available.

## Notes

- Dropping a storage location is a permanent operation and cannot be undone. Therefore, Vertica recommends that you retire a storage location before dropping it. This allows you to verify that you actually want to drop a storage location before doing so. Additionally, you can easily restore a retired storage location.
- Dropping storage locations is limited to locations that contain only temp files.

- If a location used to store data and you modified it to store only temp files, the location might still contain data files. If the storage location contains data files, Vertica does not allow you to drop it. You can manually merge out all the data in this location, wait for the ATM to mergeout the data files automatically, or you can drop partitions. Deleting data files does not work.

### Example

The following example drops a storage location on node3 that was used to store temp files:

```
=> SELECT DROP_LOCATION('/secondVerticaStorageLocation/' , 'node3');
```

### See Also

- **RETIRE\_LOCATION** (page 461) in this SQL Reference Manual
- Dropping Storage Locations and Retiring Storage Locations in the Administrator's Guide

## DROP\_PARTITION

Forces the partition of projections (if needed) and then drops the specified partition.

### Syntax

```
DROP_PARTITION [ ( table_name ) , ( partition_value ) ]
```

### Parameters

<i>table_name</i>	Specifies the name of the table. Note: The specified <i>table_name</i> argument cannot be used as a dimension table in a pre-joined projection and cannot contain projections that are not up to date (have not been refreshed)
<i>partition_value</i>	Must be specified as a string (within quotes) for all data types; for example: <code>DROP_PARTITION('trade', '2006');</code>

### Notes and Restrictions

In general, if a ROS container has data that belongs to  $n+1$  partitions and you want to drop a specific partition, the DROP\_PARTITION operation:

- 1 Forces the partition of data into two containers where
  - one container holds the data that belongs to the partition that is to be dropped
  - another container holds the remaining  $n$  partitions
- 2 Drops the specified partition.

You can also use the **MERGE\_PARTITIONS** (page 439) function to merges ROS containers that have data belonging to partitions in a specified partition key range; for example, `[partitionKeyFrom, partitionKeyTo]`.

DROP\_PARTITION forces a moveout if there is data in the WOS (WOS is not partition aware).

DROP\_PARTITION acquires an exclusive lock on the table to prevent DELETE | UPDATE | INSERT | COPY statements from affecting the table, as well as any SELECT statements issued at SERIALizable isolation level.

Users must be the table owner to drop a partition. They must have `MODIFY ( INSERT | UPDATE | DELETE )` permissions in order to:

- Partition a projection/table
- Merge partitions
- Run mergeout, moveout or purge operations on a projection

`DROP_PARTITION` operations cannot be performed on tables with projections that are not up to date (have not been refreshed).

### Examples

Using the example schema in *Defining Partitions*, the following command explicitly drops the 2006 partition key from table `trade`:

```
SELECT DROP_PARTITION('trade', 2006);
      DROP_PARTITION
-----
Partition dropped
(1 row)
```

Here, the partition key is specified:

```
SELECT DROP_PARTITION('trade', EXTRACT('year' FROM '2006-01-01'::date));
      DROP_PARTITION
-----
Partition dropped
(1 row)
```

The following example creates a table called `dates` and partitions the table by year:

```
CREATE TABLE dates (
    year INTEGER NOT NULL,
    month VARCHAR(8) NOT NULL)
PARTITION BY year * 12 + month;
```

The following statement drops the partition using a constant for Oct 2007 ( $2007*12 + 10 = 24094$ ):

```
SELECT DROP_PARTITION('dates', '24094');
      DROP_PARTITION
-----
Partition dropped
(1 row)
```

Alternatively, the expression can be placed in line: `SELECT DROP_PARTITION('dates', 2007*12 + 10);`

### See Also

***ADVANCE EPOCH*** (page 376)

***ALTER PROJECTION*** (page 580)

***COLUMN\_STORAGE*** (page 830)

***CREATE TABLE*** (page 664)

***DO\_TM\_TASK*** (page 406)

**DUMP\_PARTITION\_KEYS** (page 414)

**DUMP\_PROJECTION\_PARTITION\_KEYS** (page 415)

**DUMP\_TABLE\_PARTITION\_KEYS** (page 416)

**MERGE\_PARTITIONS** (page 439)

**PARTITION\_PROJECTION** (page 440)

**PARTITION\_TABLE** (page 441)

**PROJECTIONS** (page 811)

Dropping Partitions in the Administrator's Guide

## DROP\_STATISTICS

Removes statistics for the specified table or type.

### Syntax

```
DROP_STATISTICS('[ schema.]table' [, {'BASE' | 'HISTOGRAMS' | 'EXTENDED' | 'ALL'}
]);
```

### Return Value

- 0 - For success.
- 1 - For failure. Refer to *vertica.log* for details about the failure.

### Parameters

<code>' '</code>	Empty string. Drops statistics for all projections.
<code>[schema.]table</code>	Drops statistics for all projections within the specified table. When using more than one schema, specify the schema that contains the table with the projections you want to delete.
<code>projection</code>	Drops statistics for the specified projection.
<code>TYPE</code>	Specifies the type of statistics to drop: <ul style="list-style-type: none"><li>▪ 'BASE' (default) drops histograms and row counts (min/max column values, histogram, etc.. Extended statistics remain.</li><li>▪ 'HISTOGRAMS' drops only the histograms. Row counts and extended statistics remain.</li><li>▪ 'EXTENDED' drops soft unique keys. Histograms and row counts remain.</li><li>▪ 'ALL' drops all types of statistics.</li></ul>

### Notes

Once dropped, statistics can be time consuming to regenerate.

### Example

The following command analyzes all statistics on the VMart schema database:

```
vmart=> SELECT ANALYZE_STATISTICS('');
ANALYZE_STATISTICS
-----
0
(1 row)
```

This command drops base statistics for table `store_sales_fact` in the `store` schema:

```
vmart=> SELECT DROP_STATISTICS('store.store_sales_fact', 'BASE');
drop_statistics
-----
0
(1 row)
```

Note that this command works the same as the previous command:

```
vmart=> SELECT DROP_STATISTICS('store.store_sales_fact');
DROP_STATISTICS
-----
0
(1 row)
```

The following example drops statistics for all projections in the database and returns 0 (success):

```
vmart=> SELECT DROP_STATISTICS ('', 'ALL');
DROP_STATISTICS
-----
0
(1 row)
```

This command also drops statistics for all table projections:

```
vmart=> SELECT DROP_STATISTICS ('');
DROP_STATISTICS
-----
0
(1 row)
```

The following command drops extended statistics for the `shipping_dimension` table and returns 0 (success):

```
vmart=> SELECT DROP_STATISTICS ('shipping_dimension', 'EXTENDED');
DROP_STATISTICS
-----
0
(1 row)
```

The following command drops statistics for one of the `shipping_dimension` table's projections and returns 0 (success):

```
vmart=> SELECT DROP_STATISTICS('shipping_dimension',
'shipping_dimension_node02');
DROP_STATISTICS
-----
0
(1 row)
```

For use cases, see [Collecting Statistics in the Administrator's Guide](#)

**See Also**

***ANALYZE\_STATISTICS*** (page 385)

***EXPORT\_STATISTICS*** (page 423)

***IMPORT\_STATISTICS*** (page 433)

## DUMP\_CATALOG

Returns an internal representation of the Vertica catalog. This function is used for diagnostic purposes.

### Syntax

```
DUMP_CATALOG()
```

### Notes

To obtain an internal representation of the Vertica catalog for diagnosis, run the query:

```
SELECT DUMP_CATALOG();
```

The output is written to the specified file:

```
\o /tmp/catalog.txt  
SELECT DUMP_CATALOG();  
\o
```

Send the output to **Technical Support** (on page 1).

## DUMP\_LOCKTABLE

Returns information about deadlocked clients and the resources they are waiting for.

### Syntax

```
DUMP_LOCKTABLE ( )
```

### Notes

Use DUMP\_LOCKTABLE if Vertica becomes unresponsive:

- 1 Open an additional vsql connection.
  - 2 Execute the query:
 

```
SELECT DUMP_LOCKTABLE ( ) ;
```

 The output is written to vsql. See [Monitoring the Log Files](#).
  - 3 Copy the output and send it to **Technical Support** (on page 1).
- You can also see who is connected using the following command:

```
SELECT * FROM SESSIONS;
```

Close all sessions using the following command:

```
SELECT CLOSE_ALL_SESSIONS ( ) ;
```

Close a single session using the following command:

How to close a single session:

```
SELECT CLOSE_SESSION('session_id');
```

You get the session\_id value from the **V\_MONITOR.SESSIONS** (page 888) system table.

### See Also

**CLOSE\_ALL\_SESSIONS** (page 399)

**CLOSE\_SESSION** (page 396)

**V\_MONITOR.LOCKS** (page 858)

**V\_MONITOR.SESSIONS** (page 888)

## DUMP\_PARTITION\_KEYS

Dumps the partition keys of all projections in the system.

### Syntax

```
DUMP_PARTITION_KEYS ( )
```

### Example

```
SELECT DUMP_PARTITION_KEYS ( ) ;
```

### See Also

**DO\_TM\_TASK** (page 406)

**DROP\_PARTITION** (page 408)

**DUMP\_PROJECTION\_PARTITION\_KEYS** (page 415)

**DUMP\_TABLE\_PARTITION\_KEYS** (page 416)

**PARTITIONS** (page 862) system table

**PARTITION\_PROJECTION** (page 440)

**PARTITION\_TABLE** (page 441)

**V\_MONITOR.PARTITIONS** (page 862)

Partitioning Tables in the Administrator's Guide

## **DUMP\_PROJECTION\_PARTITION\_KEYS**

Dumps the partition keys of the specified projection.

### **Syntax**

```
DUMP_PROJECTION_PARTITION_KEYS( 'projection_name' )
```

### **Parameters**

<i>projection_name</i>	Specifies the name of the projection.
------------------------	---------------------------------------

### **Example**

The following example creates a simple table called `states` and partitions the data by state:

```
CREATE TABLE states (
    year INTEGER NOT NULL,
    state VARCHAR NOT NULL)
PARTITION BY state;
CREATE PROJECTION states_p (state, year) AS SELECT * FROM states
ORDER BY state, year UNSEGMENTED ALL NODES;
```

Now drop the partition key of the specified projection:

```
SELECT DUMP_PROJECTION_PARTITION_KEYS( 'states_p_node0001' );
Partition keys on node helios_node0001
Projection 'states_p_node0001'
No of partition keys: 1
Partition keys on node helios_node0002
...
(1 row)
```

### **See Also**

**DO\_TM\_TASK** (page 406)

**DROP\_PARTITION** (page 408)

**DUMP\_PARTITION\_KEYS** (page 414)

**DUMP\_TABLE\_PARTITION\_KEYS** (page 416)

***PARTITION\_PROJECTION*** (page 440)

***PARTITION\_TABLE*** (page 441)

***PROJECTIONS*** (page 811) system table

Partitioning Tables in the Administrator's Guide

## **DUMP\_TABLE\_PARTITION\_KEYS**

Dumps the partition keys of all projections anchored on the specified table.

### **Syntax**

```
DUMP_TABLE_PARTITION_KEYS ( 'table_name' )
```

### **Parameters**

<i>table_name</i>	Specifies the name of the table.
-------------------	----------------------------------

### **Example**

The following example creates a simple table called `states` and partitions the data by state:

```
CREATE TABLE states (  
    year INTEGER NOT NULL,  
    state VARCHAR NOT NULL)  
PARTITION BY state;  
CREATE PROJECTION states_p (state, year) AS SELECT * FROM states  
ORDER BY state, year UNSEGMENTED ALL NODES;
```

Now drop the partition keys of all projections anchored on table `states`:

```
SELECT DUMP_TABLE_PARTITION_KEYS( 'states' );  
Partition keys on helios_node0001  
Projection 'states_p_node0004'  
No of partition keys: 1  
Projection 'states_p_node0003'  
No of partition keys: 1  
Projection 'states_p_node0002'  
No of partition keys: 1  
Projection 'states_p_node0001'  
No of partition keys: 1  
Partition keys on helios_node0002  
...  
(1 row)
```

### **See Also**

***DO\_TM\_TASK*** (page 406)

***DROP\_PARTITION*** (page 408)

***DUMP\_PARTITION\_KEYS*** (page 415)

***DUMP\_PROJECTION\_PARTITION\_KEYS*** (page 416)

***PARTITION\_PROJECTION*** (page 440)

**PARTITION\_TABLE** (page 441)

Partitioning Tables in the Administrator's Guide

**ENABLE\_LOCAL\_SEGMENTS**

Enables local storage segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes. See Local Data Segmentation in the Administrator's Guide for more information.

**Syntax**

```
ENABLE_LOCAL_SEGMENTS ()
```

**Example**

```
=> SELECT ENABLE_LOCAL_SEGMENTS ();
      ENABLE_LOCAL_SEGMENTS
-----
      ENABLED
(1 row)
```

**EVALUATE\_DELETE\_PERFORMANCE**

Evaluates projections for potential **DELETE** (page 699) performance issues. If there are issues found, a warning message is displayed. For steps you can take to resolve delete and update performance issues, see Optimizing Deletes and Updates for Performance in the Administrator's Guide. This function uses data sampling to determine whether there are any issues with a projection. Therefore, it does not generate false-positives warnings, but it can miss some cases where there are performance issues.

**Note:** Optimizing for delete performance is the same as optimizing for update performance. So, you can use this function to help optimize a projection for updates as well as deletes.

**Syntax**

```
EVALUATE_DELETE_PERFORMANCE ( 'target' )
```

**Parameters**

<i>target</i>	<p>The name of a projection or table. If you supply the name of a projection, only that projection is evaluated for DELETE performance issues. If you supply the name of a table, then all of the projections anchored to the table will be evaluated for issues.</p> <p>If you do not provide a projection or table name, EVALUATE_DELETE_PERFORMANCE examines all of the projections that you can access for DELETE performance issues. Depending on the size of your database, this may take a long time.</p>
---------------	--

**Note:** When evaluating multiple projections, EVALUATE\_DELETE\_PERFORMANCE reports up to ten projections that have issues, and refers you to a table that contains the full list of issues it has found.

## Example

The following example demonstrates how you can use `EVALUATE_DELETE_PERFORMANCE` to evaluate your projections for slow `DELETE` performance.

```
=> create table example (A int, B int,C int);
CREATE TABLE
=> create projection one_sort (A,B,C) as (select A,B,C from example) order by A;
CREATE PROJECTION
=> create projection two_sort (A,B,C) as (select A,B,C from example) order by A,B;
CREATE PROJECTION
=> select evaluate_delete_performance('one_sort');
          evaluate_delete_performance
-----
No projection delete performance concerns found.
(1 row)
=> select evaluate_delete_performance('two_sort');
          evaluate_delete_performance
-----
No projection delete performance concerns found.
(1 row)
```

The previous example showed that there was no structural issue with the projection that would cause poor `DELETE` performance. However, the data contained within the projection can create potential delete issues if the sorted columns do not uniquely identify a row or small number of rows. In the following example, Perl is used to populate the table with data using a nested series of loops. The inner loop populates column C, the middle loop populates column B, and the outer loop populates column A. The result is column A contains only three distinct values (0, 1, and 2), while column B slowly varies between 20 and 0 and column C changes in each row.

`EVALUATE_DELETE_PERFORMANCE` is run against the projections again to see if the data within the projections causes any potential `DELETE` performance issues.

```
=> \! perl -e 'for ($i=0; $i<3; $i++) { for ($j=0; $j<21; $j++) { for ($k=0; $k<19; $k++) { printf "%d,%d,%d\n", $i,$j,$k;}}}' | /opt/vertica/bin/vsql -c "copy example from stdin delimiter ',' direct;"
Password:
=> select * from example;
```

```
 A | B | C
-----
 0 | 20 | 18
 0 | 20 | 17
 0 | 20 | 16
 0 | 20 | 15
 0 | 20 | 14
 0 | 20 | 13
 0 | 20 | 12
 0 | 20 | 11
 0 | 20 | 10
 0 | 20 | 9
 0 | 20 | 8
 0 | 20 | 7
 0 | 20 | 6
 0 | 20 | 5
 0 | 20 | 4
 0 | 20 | 3
 0 | 20 | 2
 0 | 20 | 1
 0 | 20 | 0
 0 | 19 | 18
1157 rows omitted
 2 | 1 | 0
 2 | 0 | 18
 2 | 0 | 17
 2 | 0 | 16
```

```

2 | 0 | 15
2 | 0 | 14
2 | 0 | 13
2 | 0 | 12
2 | 0 | 11
2 | 0 | 10
2 | 0 | 9
2 | 0 | 8
2 | 0 | 7
2 | 0 | 6
2 | 0 | 5
2 | 0 | 4
2 | 0 | 3
2 | 0 | 2
2 | 0 | 1
2 | 0 | 0

```

```
=> SELECT COUNT (*) FROM example;
COUNT
```

```
-----
1197
(1 row)
```

```
=> SELECT COUNT (DISTINCT A) FROM example;
COUNT
```

```
-----
3
(1 row)
```

```
=> select evaluate_delete_performance('one_sort');
evaluate_delete_performance
```

```
-----
Projection exhibits delete performance concerns.
(1 row)
```

```
release=> select evaluate_delete_performance('two_sort');
evaluate_delete_performance
```

```
-----
No projection delete performance concerns found.
(1 row)
```

The `one_sort` projection has potential delete issues since it only sorts on column A which has few distinct values. This means that each value in the sort column corresponds to many rows in the projection, which negatively impacts DELETE performance. Since the `two_sort` projection is sorted on columns A and B, each combination of values in the two sort columns identifies just a few rows, allowing deletes to be performed faster.

Not supplying a projection name results in all of the projections you can access being evaluated for DELETE performance issues.

```
=> select evaluate_delete_performance();
```

```
evaluate_delete_performance
```

```
-----
The following projection exhibits delete performance concerns:
```

```
"public"."one_sort"
```

```
See v_catalog.projection_delete_concerns for more details.
```

```
(1 row)
```

## EVENT\_NAME

Returns a VARCHAR value representing the name of the event that matched the row.

## Syntax

```
EVENT_NAME ()
```

## Notes

Pattern matching functions must be used in ***MATCH clause*** (page 763) syntax; for example, if you call `EVENT_NAME()` on its own, Vertica returns the following error message:

```
=> SELECT event_name ();
ERROR:  query with pattern matching function event_name must include a MATCH clause
```

## Example

**Note:** This example uses the schema defined in Event Series Pattern Matching in the Programmer's Guide. For a more detailed example, see that topic.

The following statement analyzes users' browsing history on website2.com and identifies patterns where the user landed on website2.com from another Web site (Entry) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs the values for `EVENT_NAME()`, which is the name of the event that matched the row.

```
SELECT uid,
       sid,
       ts,
       refurl,
       pageurl,
       action,
       event_name ()
FROM clickstream_log
MATCH
(PARTITION BY uid, sid ORDER BY ts
 DEFINE
  Entry   AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
  Onsite  AS PageURL ILIKE      '%website2.com%' AND Action='V',
  Purchase AS PageURL ILIKE      '%website2.com%' AND Action = 'P'
 PATTERN
  P AS (Entry Onsite* Purchase)
 RESULTS ALL ROWS);
```

uid	sid	ts	refurl	pageurl	action	event_name
1	100	12:00:00	website1.com	website2.com/home	V	Entry
1	100	12:01:00	website2.com/home	website2.com/floby	V	Onsite
1	100	12:02:00	website2.com/floby	website2.com/shamwow	V	Onsite
1	100	12:03:00	website2.com/shamwow	website2.com/buy	P	Purchase
2	100	12:10:00	website1.com	website2.com/home	V	Entry
2	100	12:11:00	website2.com/home	website2.com/forks	V	Onsite
2	100	12:13:00	website2.com/forks	website2.com/buy	P	Purchase

(7 rows)

## See Also

***MATCH Clause*** (page 763)

***MATCH\_ID*** (page 437)

***PATTERN\_ID*** (page 443)

Event Series Pattern Matching in the Programmer's Guide

## EXPORT\_CATALOG

Generates a SQL script that can be used to recreate a physical schema design in its current state on a different cluster.

### Syntax

```
EXPORT_CATALOG ( [ 'destination' ] , [ 'type' ] )
```

### Parameters

<i>destination</i>	Specifies the path and name of the SQL output file. An empty string ( ' ' ), which is the default, dumps the script to standard output. A user who is not a DBA can only specify an empty string.
<i>type</i>	Determines what is exported: <ul style="list-style-type: none"> <li>▪ DESIGN — Exports schemas, tables, constraints, views, and projections to which the user has access. This is the default value.</li> <li>▪ DESIGN_ALL — Exports all the design objects plus system objects created in Database Designer (for example, design contexts and their tables). The objects that are exported are only the ones to which the user has access.</li> <li>▪ TABLES — Exports all tables, constraints, and projections for those tables for which the user has permissions. See also <b>EXPORT_TABLES</b> (page 424).</li> </ul>

### Notes

- Exporting a design is useful for quickly moving a design to another cluster.
- The script generated by this function:
  - Creates only the non-virtual objects for which the user has access.
  - Automatically runs **MARK\_DESIGN\_KSAFE()** (page 436) with the correct K-safety value to ensure the design copy has the same same K-safety value as the original design.
  - Exports catalog objects in their Oid order.
- If a projection is created with no sort order, Vertica implicitly assigns a sort order based on the SELECT columns in the projection definition. The sort order is explicitly defined in the exported script.

### Restrictions

The export script Vertica generates is portable as long as all the projections were generated using UNSEGMENTED ALL NODES or SEGMENTED ALL NODES. Projections might not exist on ALL NODES for the following reasons:

- A projection was dropped from a node.
- A projection was created only on a subset of nodes.
- An additional node was added since the projection set was created and the design wasn't extended through Database Designer deployment.

**Example**

The following example exports the design to standard output:

```
=> SELECT EXPORT_CATALOG(' ', 'DESIGN');
```

**EXPORT\_OBJECTS**

Generates a SQL script that can be used to recreate catalog objects on a different cluster.

**Syntax**

```
EXPORT_OBJECTS( [ 'destination' ] , [ 'scope' ] , [ 'bool_value' ] )
```

**Parameters**

<i>destination</i>	Specifies the path and name of the SQL output file. An empty string (''), which is the default, dumps the script to standard output. A user who is not a DBA can only specify an empty string.
<i>scope</i>	<p>Determines the set of catalog objects to be exported where scope is one of the following:</p> <ul style="list-style-type: none"> <li>▪ an empty string ('')—exports all non-virtual objects to which the user has access, including constraints. (Note that constraints are not objects which can be passed as individual arguments.) This is the default if no scope is specified.</li> <li>▪ a comma-delimited list of items in which each item can be one of the following: <ul style="list-style-type: none"> <li>▪ —'&lt;schema&gt;.&lt;obj&gt;'—matches the named object. The named object can be a table, projection, or view.</li> <li>▪ —'&lt;obj&gt;'—matches the named object within the current search path. The named object can be a schema, table, projection, or view. If the named object is a schema, Vertica exports all non-virtual objects to which the user has access within that schema. If a schema and table both have the same name, the schema takes precedence.</li> </ul> </li> </ul> <p>EXPORT_OBJECTS returns an error if:</p> <ul style="list-style-type: none"> <li>▪ an explicitly-specified object does not exist.</li> <li>▪ the user has no access to the specified object.</li> </ul>
<i>bool-value</i>	<p>Use one of the following:</p> <ul style="list-style-type: none"> <li>▪ true—incorporates a MAKE_DESIGN_KSAFE statement with the correct K-safety value for the database at the end of the output script.</li> <li>▪ false—omits the MAKE_DESIGN_KSAFE statement from the script.</li> </ul> <p>Adding the MAKE_DESIGN_KSAFE statement is useful if you are planning to import the script into a new database and you want the new database to inherit the K-safety value from the original database.</p> <p>By default, this parameter is true.</p>

## Notes

- The script generated by this function:
  - Creates only the non-virtual objects for which the user has access.
  - Exports catalog objects in their Oid order.
- None of the parameters for EXPORT\_OBJECTS accepts a NULL value as input.

## Example

The following example exports the all the non-virtual objects to which the user has access to standard output. It does not incorporate the MAKE\_DESIGN\_KSAFE statement at the end of the file.

```
=> SELECT EXPORT_OBJECTS(' ',' ',false);
```

## EXPORT\_STATISTICS

Generates an XML file that contains statistics for the database.

**Note:** Before you export statistics for the database, run **ANALYZE\_STATISTICS()** (page 385) to collect and aggregate data samples and storage information. If you do not use ANALYZE\_STATISTICS, the Database Designer produces a suboptimal projection similar to projections created for temporary designs.

## Syntax

```
EXPORT_STATISTICS ( 'destination' )
```

## Parameters

<i>destination</i>	Specifies the path and name of the XML output file. An empty string returns the script to standard output.
--------------------	--

## Example

The following command exports statistics on the VMart example database to a file:

```
vmart=> SELECT EXPORT_STATISTICS('/vmart/statistics/vmart_stats');
        export_statistics
-----
Statistics exported successfully
(1 row)
```

## See Also

**ANALYZE\_STATISTICS** (page 385)

**DROP\_STATISTICS** (page 410)

**IMPORT\_STATISTICS** (page 433)

Collecting Statistics in the Administrator's Guide

## EXPORT\_TABLES

Generates a SQL script that can be used to recreate a logical schema (schemas, tables, constraints, and views) on a different cluster.

### Syntax

```
EXPORT_TABLES ( [ 'destination' ] , [ 'scope' ] )
```

### Parameters

<i>destination</i>	Specifies the path and name of the SQL output file. An empty string ( ' ' ), which is the default, dumps the script to standard output. A user who is not a DBA can only specify an empty string.
<i>scope</i>	<p>Determines the tables to be exported where scope is one of the following:</p> <ul style="list-style-type: none"> <li>▪ an empty string ( ' ' )—exports all non-virtual objects to which the user has access, including constraints. (Note that constraints are not objects which can be passed as individual arguments.) This is the default if no scope is specified.</li> <li>▪ a comma-delimited list of items in which each item can be one of the following: <ul style="list-style-type: none"> <li>▪ —'&lt;schema&gt;.&lt;obj&gt;'—matches the named object. The named object can be a table or view.</li> <li>▪ —'&lt;obj&gt;'—matches the named object within the current search path. The named object can be a schema, table, or view. If the named object is a schema, Vertica exports all non-virtual objects to which the user has access within that schema. If a schema and table both have the same name, the schema takes precedence.</li> </ul> </li> </ul> <p>EXPORT_TABLES returns an error if:</p> <ul style="list-style-type: none"> <li>▪ an explicitly-specified object does not exist.</li> <li>▪ The user has no access to the specified object.</li> </ul>

### Notes

- The script generated by this function:
  - Creates only the non-virtual objects for which the user has access.
  - Exports catalog objects in their Oid order.
- If projections are specified in the scope parameter, they are ignored.
- None of the parameters for EXPORT\_TABLES accepts a NULL value as input.

### Example

The following example exports the store.store\_orders\_fact table to standard output:

```
=> SELECT EXPORT_TABLES(' ', 'store.store_orders_fact');
```

## GET\_AHM\_EPOCH

Returns the number of the epoch in which the Ancient History Mark is located. Data deleted up to and including the AHM epoch can be purged from physical storage.

### Syntax

```
GET_AHM_EPOCH()
```

**Note:** The AHM epoch is 0 (zero) by default (purge is disabled).

### Examples

```
SELECT GET_AHM_EPOCH();
       get_ahm_epoch
-----
Current AHM epoch: 0
(1 row)
```

## GET\_AHM\_TIME

Returns a **TIMESTAMP** value representing the Ancient History Mark. Data deleted up to and including the AHM epoch can be purged from physical storage.

### Syntax

```
GET_AHM_TIME()
```

### Examples

```
SELECT GET_AHM_TIME();
       GET_AHM_TIME
-----
Current AHM Time: 2010-05-13 12:48:10.532332-04
(1 row)
```

### See Also

**SET DATESTYLE** (page 767) for information about valid **TIMESTAMP** (page 90) values.

## GET\_COMPLIANCE\_STATUS

Displays whether your database is in compliance with your Vertica license agreement. This information includes the results of Vertica's most recent audit of the database size (if your license has a data allowance as part of its terms), and the license term (if your license has an end date).

The information displayed by **GET\_COMPLIANCE\_STATUS** includes:

- The estimated size of the database (see *How Vertica Calculates Database Size* in the *Administrator's Guide* for an explanation of the size estimate).
- The raw data size allowed by your Vertica license.
- The percentage of your allowance that your database is currently using.
- The date and time of the last audit.
- Whether your database complies with the data allowance terms of your license agreement.
- The end date of your license.
- How many days remain until your license expires.

**Note:** If your license does not have a data allowance or end date, some of the values may not appear in GET\_COMPLIANCE\_STATUS's output

If the audit shows your license is not in compliance with your data allowance, you should either delete data to bring the size of the database under the licensed amount, or upgrade your license. If your license term has expired, you should contact Vertica immediately to renew your license. See Managing Your License Key in the Administrator's Guide for further details.

## Syntax

```
GET_COMPLIANCE_STATUS()
```

## Example

```
-----
                                GET_COMPLIANCE_STATUS
-----
---
Raw Data Size: 0.071GB +/- 0.003GB
License Size : 500.000GB
Utilization  : 0%
Audit Time   : 2011-03-09 09:54:09.538704+00
Compliance Status : The database is in compliance with respect to raw data size.

License End Date: 04/06/2011
Days Remaining: 28.59
(1 row)
```

## GET\_AUDIT\_TIME

Reports the time when the automatic audit of database size occurs. Vertica performs this audit if your Vertica license includes a data size allowance. For details of this audit, see Managing Your License Key in the Administrator's Guide. To change the time the audit runs, use the **SET\_AUDIT\_TIME** (page 465) function.

## Syntax

```
GET_AUDIT_TIME()
```

## Example

```
=> SELECT get_audit_time();
           get_audit_time
-----
The audit is scheduled to run at 11:59 PM each day.
(1 row)
```

## GET\_CURRENT\_EPOCH

Returns the number of the current epoch. The epoch into which data (COPY, INSERT, UPDATE, and DELETE operations) is currently being written. The current epoch advances automatically every three minutes.

**Syntax**

```
GET_CURRENT_EPOCH()
```

**Examples**

```
SELECT GET_CURRENT_EPOCH();
       GET_CURRENT_EPOCH
-----
                683
(1 row)
```

**GET\_DATA\_COLLECTOR\_POLICY**

Retrieves a brief statement about the retention policy for the specified component.

**Syntax**

```
GET_DATA_COLLECTOR_POLICY( 'component' )
```

**Parameters**

<i>component</i>	Returns the retention policy for the specified component.
------------------	---

**Notes**

For a current list of component names, query the **V\_MONITOR.DATA\_COLLECTOR** (page 837) system table. For example:

```
=> SELECT DISTINCT component, description FROM data_collector ORDER BY 1 ASC;
```

**Example**

The following example returns the history of all resource acquisitions (the ResourceAcquisitions component):

```
=> SELECT get_data_collector_policy('ResourceAcquisitions');
       get_data_collector_policy
-----
1000KB kept in memory, 10000KB kept on disk.
(1 row)
```

**See Also**

**V\_MONITOR.DATA\_COLLECTOR** (page 837)

Retaining Monitoring Information in the Troubleshooting Guide

**GET\_LAST\_GOOD\_EPOCH**

Returns the number of the last good epoch. A term used in manual recovery, LGE (Last Good Epoch) refers to the most recent epoch that can be recovered.

**Syntax**

```
GET_LAST_GOOD_EPOCH()
```

**Examples**

```
SELECT GET_LAST_GOOD_EPOCH();
```

```
GET_LAST_GOOD_EPOCH
```

```
-----  
682
```

```
(1 row)
```

## GET\_NUM\_ACCEPTED\_ROWS

Returns the number of rows loaded into the database for the last completed load for the current session.

### Syntax

```
GET_NUM_ACCEPTED_ROWS ();
```

### Notes

- Only loads from STDIN or a single file on the initiator are supported. This function cannot be called for multi-node loads.
- Information is not available for a load that is currently running. Check the system table **LOAD\_STREAMS** (page 856) for its status.
- Data regarding loads does not persist, and is dropped when a new load is initiated.
- GET\_NUM\_ACCEPTED\_ROWS is a meta-function, Do not use it as a value in an INSERT query.

## GET\_NUM\_REJECTED\_ROWS

Returns the number of rows that were rejected during the last completed load for the current session.

### Syntax

```
GET_NUM_REJECTED_ROWS ();
```

### Notes

- Only loads from STDIN or a single file on the initiator are supported. This function cannot be called for multi-node loads.
- Information is not available for a load that is currently running. Check the system table **LOAD\_STREAMS** (page 856) for its status.
- Data regarding loads does not persist, and is dropped when a new load is initiated.
- GET\_NUM\_REJECTED\_ROWS is a meta-function, Do not use it as a value in an INSERT query.

## GET\_PROJECTION\_STATUS

Returns information relevant to the status of a projection.

### Syntax

```
GET_PROJECTION_STATUS ( [ schema-name. ] projection );
```

### Parameters

<code>[<i>schema-name.</i>] <i>projection</i></code>	Is the name of the projection for which to display status. When using more than one schema, specify the schema that contains
--	--

the projection.
-----------------

## Description

GET\_PROJECTION\_STATUS returns information relevant to the status of a projection:

- The current K-safety status of the database
- The number of nodes in the database
- Whether the projection is segmented
- The number and names of buddy projections
- Whether the projection is safe
- Whether the projection is up-to-date
- Whether statistics have been computed for the projection

## Notes

- You can use GET\_PROJECTION\_STATUS to monitor the progress of a projection data refresh. See **ALTER PROJECTION** (page 580).
- When using GET\_PROJECTION\_STATUS or GET\_PROJECTIONS you must provide the name and node (for example, ABC\_NODE01) instead of just ABC.
- To view a list of the nodes in a database, use the View Database Command in the Administration Tools.

## Examples

```
=> SELECT GET_PROJECTION_STATUS('public.customer_dimension_site01');
```

```
GET_PROJECTION_STATUS
```

```
-----
Current system K is 1.
# of Nodes: 4.
public.customer_dimension_site01 [Segmented: No] [Seg Cols: ] [K: 3]
[public.customer_dimension_site04, public.customer_dimension_site03,
public.customer_dimension_site02] [Safe: Yes] [UpToDate: Yes][Stats: Yes]
```

## See Also

**ALTER PROJECTION** (page 580)

**GET\_PROJECTIONS** (page 429)

## GET\_PROJECTIONS, GET\_TABLE\_PROJECTIONS

**Note:** This function was formerly named GET\_TABLE\_PROJECTIONS(). Vertica still supports the former function name.

Returns information relevant to the status of a table:

- The current K-safety status of the database
- The number of sites (nodes) in the database
- The number of projections for which the specified table is the anchor table
- For each projection:
  - The projection's buddy projections
  - Whether the projection is segmented

- Whether the projection is safe
- Whether the projection is up-to-date

## Syntax

```
GET_PROJECTIONS ( [ schema-name. ] table )
```

## Parameters

<i>[schema-name.] table</i>	Is the name of the table for which to list projections. When using more than one schema, specify the schema that contains the table.
-----------------------------	--

## Notes

- You can use GET\_PROJECTIONS to monitor the progress of a projection data refresh. See **ALTER PROJECTION** (page 580).
- When using GET\_PROJECTIONS or GET\_PROJECTION\_STATUS for replicated projections created using the ALL NODES syntax, you must provide the name and node (for example, ABC\_NODE01 instead of just ABC).
- To view a list of the nodes in a database, use the View Database Command in the Administration Tools.

## Examples

The following example gets information about the store\_dimension table in the VMart schema:

```
=> SELECT GET_PROJECTIONS('store.store_dimension');
```

```
-----  
Current system K is 1.
```

```
# of Nodes: 4.
```

```
Table store.store_dimension has 4 projections.
```

```
Projection Name: [Segmented] [Seg Cols] [# of Buddies] [Buddy Projections] [Safe] [UptoDate]
```

```
-----  
store.store_dimension_node0004 [Segmented: No] [Seg Cols: ] [K: 3] [store.store_dimension_node0003,  
store.store_dimension_node0002, store.store_dimension_node0001] [Safe: Yes] [UptoDate: Yes][Stats:  
Yes]
```

```
store.store_dimension_node0003 [Segmented: No] [Seg Cols: ] [K: 3] [store.store_dimension_node0004,  
store.store_dimension_node0002, store.store_dimension_node0001] [Safe: Yes] [UptoDate: Yes][Stats:  
Yes]
```

```
store.store_dimension_node0002 [Segmented: No] [Seg Cols: ] [K: 3] [store.store_dimension_node0004,  
store.store_dimension_node0003, store.store_dimension_node0001] [Safe: Yes] [UptoDate: Yes][Stats:  
Yes]
```

```
store.store_dimension_node0001 [Segmented: No] [Seg Cols: ] [K: 3] [store.store_dimension_node0004,  
store.store_dimension_node0003, store.store_dimension_node0002] [Safe: Yes] [UptoDate: Yes][Stats:  
Yes]
```

```
(1 row)
```

## See Also

**ALTER PROJECTION** (page 580)

**GET\_PROJECTION\_STATUS** (page 428)

## INTERRUPT\_STATEMENT

Interrupts the specified statement (within an external session), rolls back the current transaction, and writes a success or failure message to the log file.



```

transaction_description | user dbadmin (select * from sessions;)
statement_start         | 2011-01-03 15:36:13.896288
statement_id           | 10
last_statement_duration_us | 14978
current_statement      | select * from sessions;
ssl_state              | None
authentication_method  | Trust
-[ RECORD 2
]-----+-----
node_name              | v_vmartdb_node0003
user_name              | dbadmin
client_hostname        | 127.0.0.1:56367
client_pid             | 1191
login_timestamp        | 2011-01-03 15:31:44.939302-05
session_id             | stress06-25663:0xbe
client_label           |
transaction_start      | 2011-01-03 15:34:51.05939
transaction_id         | 54043195528458775
transaction_description | user dbadmin (COPY Mart_Fact FROM
'/data/Mart_Fact.tbl'
                        DELIMITER '|' NULL '\\n' DIRECT;)
statement_start        | 2011-01-03 15:35:46.436748
statement_id           | 5
last_statement_duration_us | 1591403
current_statement      | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER
'|'
                        NULL '\\n' DIRECT;
ssl_state              | None
authentication_method  | Trust

```

Interrupt the COPY DIRECT statement running in stress06-25663:0xbe:

```

=> \x
Expanded display is off.
=> SELECT INTERRUPT_STATEMENT('stress06-25663:0x1537', 5);
           interrupt_statement
-----
Statement interrupt sent. Check v_monitor.sessions for progress.
(1 row)

```

Verify that the interrupted statement is no longer active by looking at the current\_statement column in the SESSIONS system table. This column becomes blank when the statement has been interrupted:

```

=> SELECT * FROM SESSIONS;
-[ RECORD 1
]-----+-----
node_name              | v_vmartdb_node0001
user_name              | dbadmin
client_hostname        | 127.0.0.1:52110
client_pid             | 4554
login_timestamp        | 2011-01-03 14:05:40.252625-05
session_id             | stress04-4325:0x14
client_label           |
transaction_start      | 2011-01-03 14:05:44.325781

```

```

transaction_id          | 45035996273728326
transaction_description | user dbadmin (select * from sessions;)
statement_start        | 2011-01-03 15:36:13.896288
statement_id           | 10
last_statement_duration_us | 14978
current_statement      | select * from sessions;
ssl_state              | None
authentication_method  | Trust
-[ RECORD 2
]-----+-----
node_name              | v_vmartdb_node0003
user_name              | dbadmin
client_hostname        | 127.0.0.1:56367
client_pid             | 1191
login_timestamp        | 2011-01-03 15:31:44.939302-05
session_id             | stress06-25663:0xbe
client_label           |
transaction_start      | 2011-01-03 15:34:51.05939
transaction_id         | 54043195528458775
transaction_description | user dbadmin (COPY Mart_Fact FROM
'/data/Mart_Fact.tbl'
                        DELIMITER '|' NULL '\\\n' DIRECT;)
statement_start        | 2011-01-03 15:35:46.436748
statement_id           | 5
last_statement_duration_us | 1591403
current_statement      |
ssl_state              | None
authentication_method  | Trust

```

**See Also****SESSIONS** (page 888)

Managing Sessions and Configuration Parameters in the Administrator's Guide

**IMPORT\_STATISTICS**

Imports statistics from the XML file generated by the EXPORT\_STATISTICS command.

**Syntax**IMPORT\_STATISTICS ( '*destination*' )**Parameters**

<i>destination</i>	Specifies the path and name of the XML input file (which is the output of EXPORT_STATISTICS function).
--------------------	--

**Notes**

- Imported statistics override existing statistics for all projections on the specified table.
- For use cases, see Collecting Statistics in the Administrator's Guide

**See Also****ANALYZE\_STATISTICS** (page 385)

**DROP\_STATISTICS** (page 410)

**EXPORT\_STATISTICS** (page 423)

## ISUTF8

Tests whether a string is a valid UTF-8 string. Returns true if the string conforms to UTF-8 standards, and false otherwise. This function is useful to test strings for UTF-8 compliance before passing them to one of the regular expression functions, such as **REGEXP\_LIKE** (page 452), which expect UTF-8 characters by default.

### Syntax

```
ISUTF8( string );
```

### Parameters

<i>string</i>	The string to test for UTF-8 compliance.
---------------	--

### Examples

```
=> SELECT ISUTF8(E'\xC2\xBF'); -- UTF-8 INVERTED QUESTION MARK
ISUTF8
-----
t
(1 row)
```

```
=> SELECT ISUTF8(E'\xC2\xC0'); -- UNDEFINED UTF-8 CHARACTER
ISUTF8
-----
f
(1 row)
```

## MAKE\_AHM\_NOW

Sets the Ancient History Mark (AHM) to the greatest allowable value, and lets you drop any projections that existed before the issue occurred.

**Caution:** This function is intended for use by Administrators only.

### Syntax

```
MAKE_AHM_NOW ( [ true ] )
```

### Parameters

<i>true</i>	[Optional] Allows AHM to advance when nodes are down. <b>Note:</b> If the AHM is advanced after the last good epoch of the failed nodes, those nodes must recover all data from scratch. Use with care.
-------------	--

## Notes

- The `MAKE_AHM_NOW` function performs the following operations:
  - Advances the epoch.
  - Performs a moveout operation on all projections.
  - Sets the AHM to LGE — at least to the current epoch at the time `MAKE_AHM_NOW()` was issued.
- All history is lost and you cannot perform historical queries prior to the current epoch.

## Example

```
=> SELECT MAKE_AHM_NOW();
       MAKE_AHM_NOW
```

```
-----
AHM set (New AHM Epoch: 683)
(1 row)
```

The following command allows the AHM to advance, even though node 2 is down:

```
=> SELECT MAKE_AHM_NOW(true);
WARNING: Received no response from v_vmartdb_node0002 in get cluster LGE
WARNING: Received no response from v_vmartdb_node0002 in get cluster LGE
WARNING: Received no response from v_vmartdb_node0002 in set AHM
       MAKE_AHM_NOW
```

```
-----
AHM set (New AHM Epoch: 684)
(1 row)
```

## See Also

***DROP PROJECTION*** (page 704)

***MARK\_DESIGN\_KSAFE*** (page 436)

***SET\_AHM\_EPOCH*** (page 463)

***SET\_AHM\_TIME*** (page 464)

## MARK\_DESIGN\_KSAFE

Enables or disables high availability in your environment, in case of a failure. Before enabling recovery, MARK\_DESIGN\_KSAFE queries the catalog to determine whether a cluster's physical schema design meets the following requirements:

- Dimension tables are replicated on all nodes.
- Fact table superprojections are segmented with each segment on a different node.
- Each fact table projection has at least one buddy projection for K-safety=1 (or two buddy projections for K-safety=2).

Buddy projections are also segmented across database nodes, but the distribution is modified so that segments that contain the same data are distributed to different nodes. See High Availability Through Projections in the Concepts Guide.

**Note:** Projections are considered to be buddies if they contain the same columns and have the same segmentation. They can have different sort orders.

MARK\_DESIGN\_KSAFE does not change the physical schema in any way.

### Syntax

```
SELECT MARK_DESIGN_KSAFE ( k )
```

### Parameters

<i>k</i>	<p>2 enables high availability if the schema design meets requirements for K-safety=2</p> <p>1 enables high availability if the schema design meets requirements for K-safety=1</p> <p>0 disables high availability</p>
----------	---

If you specify a *k* value of one (1) or two (2), Vertica returns one of the following messages.

#### Success:

```
Marked design n-safe
```

#### Failure:

```
The schema does not meet requirements for K=n.
Fact table projection projection-name
has insufficient "buddy" projections.
```

*n* in the message is 1 or 2 and represents the *k* value.

### Notes

- The database's internal recovery state persists across database restarts but it is not checked at startup time.
- If a database has automatic recovery enabled, you must temporarily disable automatic recovery before creating a new table.
- When one node fails on a system marked K-safe=1, the remaining nodes are available for DML operations.

## Examples

```
=> SELECT MARK_DESIGN_KSAFE(1);
      mark_design_ksafe
```

```
-----
Marked design 1-safe
(1 row)
```

If the physical schema design is not K-Safe, messages indicate which projections do not have a buddy:

```
=> SELECT MARK_DESIGN_KSAFE(1);
The given K value is not correct; the schema is 0-safe
Projection pp1 has 0 buddies, which is smaller than the given K of 1
Projection pp2 has 0 buddies, which is smaller than the given K of 1
.
.
.
(1 row)
```

## See Also

**SYSTEM** (page 898)

High Availability and Recovery in the Concepts Guide

**SQL System Tables (Monitoring APIs)** (page 794) topic in the Administrator's Guide

Using Identically Segmented Projections in the Programmer's Guide

Failure Recovery in the Troubleshooting Guide

## MATCH\_ID

Returns an INTEGER value that is the sequential number of the match.

### Syntax

```
MATCH_ID()
```

### Notes

Pattern matching functions must be used in **MATCH clause** (page 763) syntax; for example, if you call MATCH\_ID() on its own, Vertica returns the following error message:

```
=> SELECT match_id();
ERROR:  query with pattern matching function match_id must include a MATCH clause
```

### Example

**Note:** This example uses the schema defined in Event Series Pattern Matching in the Programmer's Guide. For a more detailed example, see that topic.

The following statement analyzes users' browsing history on website2.com and identifies patterns where the user landed on website2.com from another Web site (Entry) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs values for the MATCH\_ID(), which represents a sequential number of the match.

```

SELECT uid,
       sid,
       ts,
       refurl,
       pageurl,
       action,
       match_id()
FROM clickstream_log
MATCH
(PARTITION BY uid, sid ORDER BY ts
 DEFINE
  Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
  Onsite AS PageURL ILIKE '%website2.com%' AND Action='V',
  Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'
 PATTERN
  P AS (Entry Onsite* Purchase)
 RESULTS ALL ROWS);
uid | sid | ts | refurl | pageurl | action | match_id
-----+-----+-----+-----+-----+-----+-----
 2 | 100 | 12:10:00 | website1.com | website2.com/home | V | 1
 2 | 100 | 12:11:00 | website2.com/home | website2.com/forks | V | 2
 2 | 100 | 12:13:00 | website2.com/forks | website2.com/buy | P | 3
 1 | 100 | 12:00:00 | website1.com | website2.com/home | V | 1
 1 | 100 | 12:01:00 | website2.com/home | website2.com/floby | V | 2
 1 | 100 | 12:02:00 | website2.com/floby | website2.com/shamwow | V | 3
 1 | 100 | 12:03:00 | website2.com/shamwow | website2.com/buy | P | 4
(7 rows)

```

**See Also**

**MATCH Clause** (page 763)

**EVENT\_NAME** (page 419)

**PATTERN\_ID** (page 443)

Event Series Pattern Matching in the Programmer's Guide

**MEASURE\_LOCATION\_PERFORMANCE**

Measures disk performance for the location specified.

**Syntax**

```
MEASURE_LOCATION_PERFORMANCE ( 'path' , 'node' )
```

**Parameters**

<i>path</i>	Specifies where the storage location to measure is mounted.
<i>node</i>	Is the Vertica node where the location to be measured is available..

**Notes**

- If you intend to create a tiered disk architecture in which projections, columns, and partitions are stored on different disks based on predicted or measured access patterns, you need to measure storage location performance for each location in which data is stored. You do not need to measure storage location performance for temp data storage locations because temporary files are stored based on available space.

- This method of measuring storage location performance applies only to configured clusters. If you want to measure a disk before configuring a cluster see [Measuring Location Performance](#).
- Storage location performance equates to the amount of time it takes to read and write 1 MB of data from the disk. This time equates to:

IO time = Time to read/write 1MB + Time to seek = 1/Throughput + 1/Latency

Throughput is the average throughput of sequential reads/writes (units in MB per second)

Latency is for random reads only in seeks (units in seeks per second)

**Note:** The IO time of a faster storage location is less than a slower storage location.

### Example

The following example measures the performance of a storage location on node2:

```
=> SELECT MEASURE_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/' ,
'node2');
```

```
WARNING: measure_location_performance can take a long time. Please check logs for
progress
```

```
measure_location_performance
```

```
-----
Throughput : 122 MB/sec. Latency : 140 seeks/sec
```

### See Also

**ADD\_LOCATION** (page 374)

**ALTER\_LOCATION\_USE** (page 376)

**RETIRE\_LOCATION** (page 461)

Measuring Location Performance in the Administrator's Guide

## MERGE\_PARTITIONS

Merges ROS containers that have data belonging to partitions in a specified partition key range: [ *partitionKeyFrom*, *partitionKeyTo* ] .

### Syntax

```
MERGE_PARTITIONS [ ( table_name ) ,
... ( partition_key_from ) , ( partition_key_to ) ]
```

### Parameters

<i>table_name</i>	Specifies the name of the table
<i>partition_key_from</i>	Specifies the start point of the partition
<i>partition_key_to</i>	Specifies the end point of the partition

### Notes

- Partitioning functions take immutable functions only, in order that the same information be available across all nodes.

- The edge values are included in the range, and `partition_key_from` must be less than or equal to `partition_key_to`.
- Inclusion of partitions in the range is based on the application of less than(<)/greater than(>) operators of the corresponding data type.

**Note:** No restrictions are placed on a partition key's data type.

- If `partition_key_from` is the same as `partition_key_to`, all ROS containers of the partition key are merged into one ROS.

Users must be the table owner to drop a partition. They must have `MODIFY ( INSERT | UPDATE | DELETE )` permissions in order to:

- Partition a projection/table
- Merge partitions
- Run mergeout, moveout or purge operations on a projection

## Examples

```
=> SELECT MERGE_PARTITIONS('T1', '200', '400');
=> SELECT MERGE_PARTITIONS('T1', '800', '800');
=> SELECT MERGE_PARTITIONS('T1', 'CA', 'MA');
=> SELECT MERGE_PARTITIONS('T1', 'false', 'true');
=> SELECT MERGE_PARTITIONS('T1', '06/06/2008', '06/07/2008');
=> SELECT MERGE_PARTITIONS('T1', '02:01:10', '04:20:40');
=> SELECT MERGE_PARTITIONS('T1', '06/06/2008 02:01:10', '06/07/2008 02:01:10');
=> SELECT MERGE_PARTITIONS('T1', '8 hours', '1 day 4 hours 20 seconds');
```

## PARTITION\_PROJECTION

Forces a split of ROS containers of the specified projection.

### Syntax

```
PARTITION_PROJECTION ( projection_name )
```

### Parameters

<i>projection_name</i>	Specifies the name of the projection.
------------------------	---------------------------------------

### Notes

Partitioning expressions take immutable functions only, in order that the same information be available across all nodes.

`PARTITION_PROJECTION()` is similar to ***PARTITION\_TABLE*** (page 441)(), except that `PARTITION_PROJECTION` works only on the specified projection, instead of the table.

Vertica internal operations (mergeout, refresh, and recovery) maintain partition separation except in certain cases:

- Recovery of a projection when the buddy projection from which the partition is recovering is identically sorted. If the projection is undergoing a full rebuild, it is recovered one ROS container at a time. The projection ends up with a storage layout identical to its buddy and is, therefore, properly segmented.

**Note:** In the case of a partial rebuild, all recovered data goes into a single ROS container and must be partitioned manually.

- Manual tuple mover operations often output a single storage container, combining any existing partitions; for example, after executing any of the `PURGE()` operations.

Users must be the table owner to drop a partition. They must have `MODIFY ( INSERT | UPDATE | DELETE )` permissions in order to:

- Partition a projection/table
- Merge partitions
- Run mergeout, moveout or purge operations on a projection

`PARTITION_PROJECTION()` purges data while partitioning ROS containers if deletes were applied before the AHM epoch.

### Example

The following command forces a split of ROS containers on the `states_p_node01` projection:

```
=> SELECT PARTITION_PROJECTION ('states_p_node01');
      partition_projection
-----
Projection partitioned
(1 row)
```

### See Also

***DO\_TM\_TASK*** (page 406)

***DROP\_PARTITION*** (page 408)

***DUMP\_PARTITION\_KEYS*** (page 414)

***DUMP\_PROJECTION\_PARTITION\_KEYS*** (page 415)

***DUMP\_TABLE\_PARTITION\_KEYS*** (page 416)

***PARTITION\_TABLE*** (page 441)

Partitioning Tables in the Administrator's Guide

### **PARTITION\_TABLE**

Forces the system to break up any ROS containers that contain multiple distinct values of the partitioning expression. Only ROS containers with more than one distinct value participate in the split.

### Syntax

```
PARTITION_TABLE ( 'table_name' )
```

## Parameters

<code>table_name</code>	Specifies the name of the table.
-------------------------	----------------------------------

## Notes

PARTITION\_TABLE is similar to **PARTITION\_PROJECTION** (page 440), except that PARTITION\_TABLE works on the specified table.

Vertica internal operations (mergeout, refresh, and recovery) maintain partition separation except in certain cases:

- Recovery of a projection when the buddy projection from which the partition is recovering is identically sorted. If the projection is undergoing a full rebuild, it is recovered one ROS container at a time. The projection ends up with a storage layout identical to its buddy and is, therefore, properly segmented.

**Note:** In the case of a partial rebuild, all recovered data goes into a single ROS container and must be partitioned manually.

- Manual tuple mover operations often output a single storage container, combining any existing partitions; for example, after executing any of the `PURGE()` operations.

Users must be the table owner to drop a partition. They must have `MODIFY ( INSERT | UPDATE | DELETE )` permissions in order to:

- Partition a projection/table
- Merge partitions
- Run mergeout, moveout or purge operations on a projection

Partitioning functions take immutable functions only, in order that the same information be available across all nodes.

## Example

The following example creates a simple table called `states` and partitions data by state.

```
=> CREATE TABLE states (
    year INTEGER NOT NULL,
    state VARCHAR NOT NULL)
PARTITION BY state;
=> CREATE PROJECTION states_p (state, year) AS
SELECT * FROM states
ORDER BY state, year UNSEGMENTED ALL NODES;
```

Now issue the command to partition table `states`:

```
=> SELECT PARTITION_TABLE('states');
PARTITION_TABLE
-----
partition operation for projection 'states_p_node0004'
partition operation for projection 'states_p_node0003'
partition operation for projection 'states_p_node0002'
partition operation for projection 'states_p_node0001'
```

(1 row)

## See Also

**DO\_TM\_TASK** (page 406)

**DROP\_PARTITION** (page 408)

**DUMP\_PARTITION\_KEYS** (page 414)

**DUMP\_PROJECTION\_PARTITION\_KEYS** (page 415)

**DUMP\_TABLE\_PARTITION\_KEYS** (page 416)

**PARTITION\_PROJECTION** (page 440)

Partitioning Tables in the Administrator's Guide

## PATTERN\_ID

Returns an integer value that is a partition-wide unique identifier for the instance of the pattern that matched.

## Syntax

```
PATTERN_ID()
```

## Notes

Pattern matching functions must be used in **MATCH clause** (page 763) syntax; for example, if call PATTERN\_ID() on its own, Vertica returns the following error message:

```
=> SELECT pattern_id();
ERROR:  query with pattern matching function pattern_id must include a MATCH clause
```

## Example

**Note:** This example uses the schema defined in Event Series Pattern Matching in the Programmer's Guide. For a more detailed example, see that topic.

The following statement analyzes users' browsing history on website2.com and identifies patterns where the user landed on website2.com from another Web site (Entry) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs values for PATTERN\_ID(), which represents the partition-wide identifier for the instance of the pattern that matched.

```
SELECT uid,
       sid,
       ts,
       refurl,
       pageurl,
       action,
       pattern_id()
FROM clickstream_log
MATCH
(PARTITION BY uid, sid ORDER BY ts
DEFINE
  Entry    AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
  Onsite   AS PageURL ILIKE '%website2.com%' AND Action='V',
  Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'
```

```

PATTERN
P AS (Entry Onsite* Purchase)
RESULTS ALL ROWS);
uid | sid | ts | refurl | pageurl | action | pattern_id
-----+-----+-----+-----+-----+-----+-----
 2 | 100 | 12:10:00 | website1.com | website2.com/home | V | 1
 2 | 100 | 12:11:00 | website2.com/home | website2.com/forks | V | 1
 2 | 100 | 12:13:00 | website2.com/forks | website2.com/buy | P | 1
 1 | 100 | 12:00:00 | website1.com | website2.com/home | V | 1
 1 | 100 | 12:01:00 | website2.com/home | website2.com/floby | V | 1
 1 | 100 | 12:02:00 | website2.com/floby | website2.com/shamwow | V | 1
 1 | 100 | 12:03:00 | website2.com/shamwow | website2.com/buy | P | 1
(7 rows)

```

**See Also****MATCH Clause** (page 763)**EVENT\_NAME** (page 419)**MATCH\_ID** (page 437)

Event Series Pattern Matching in the Programmer's Guide

**PURGE**

Purges all projections in the physical schema. Permanently removes deleted data from physical storage so that the disk space can be reused. You can purge historical data up to and including the epoch in which the Ancient History Mark is contained.

**Syntax**

PURGE ( )

**Notes**

- PURGE() was formerly named PURGE\_ALL\_PROJECTIONS. Vertica supports both function calls.
- Manual tuple mover operations, such as the PURGE() operations, often output a single storage container, combining any existing partitions. For example, if PURGE() is used on a non-partitioned table, all ROS containers are combined into a single container. Non-partitioned tables cannot be re-partitioned into multiple ROS containers. A purge operation on a partitioned table also results in a single ROS.
- To re-partition the data into multiple ROS containers, use the **PARTITION\_TABLE** (page 441)() function.

**Caution:** PURGE could temporarily take up significant disk space while the data is being purged.

**See Also****MERGE\_PARTITIONS** (page 439)**PARTITION\_TABLE** (page 441)**PURGE\_PROJECTION** (page 445)**PURGE\_TABLE** (page 445)

**STORAGE\_CONTAINERS** (page 890)

Purging Deleted Data in the Administrator's Guide

**PURGE\_PROJECTION**

Purges the specified projection. Permanently removes deleted data from physical storage so that the disk space can be reused. You can purge historical data up to and including the epoch in which the Ancient History Mark is contained.

**Caution:** PURGE\_PROJECTION could temporarily take up significant disk space while the data is being purged.

**Syntax**

```
PURGE_PROJECTION ( [ schema-name.] projection_name )
```

**Parameters**

<i>projection_name</i>	Is the name of a specific projection. When using more than one schema, specify the schema that contains the projection.
------------------------	---

**Notes**

See **PURGE** (page 444) for notes about the outcome of purge operations.

**See Also**

**MERGE\_PARTITIONS** (page 439)

**PURGE\_TABLE** (page 445)

**STORAGE\_CONTAINERS** (page 890)

Purging Deleted Data in the Administrator's Guide

**PURGE\_TABLE**

**Note:** This function was formerly named `PURGE_TABLE_PROJECTIONS()`. Vertica still supports the former function name.

Purges all projections of the specified table. Permanently removes deleted data from physical storage so that the disk space can be reused. You can purge historical data up to and including the epoch in which the Ancient History Mark is contained.

**Syntax**

```
PURGE_TABLE ( [ schema_name.] table_name )
```

**Parameters**

<i>[schema_name.]table_name</i>	Is the name of a specific table in the optionally-specified logical schema. When using more than one schema, specify the schema that contains the projection.
---------------------------------	--

**Caution:** `PURGE_TABLE` could temporarily take up significant disk space while the data is being purged.

### Example

The following example purges all projections for the store sales fact table located in the Vmart schema:

```
=> SELECT PURGE_TABLE('store.store_sales_fact');
```

### See Also

**PURGE** (page 444) for notes about the outcome of purge operations.

**MERGE\_PARTITIONS** (page 439)

**PURGE\_TABLE** (page 445)

**STORAGE\_CONTAINERS** (page 890)

Purging Deleted Data in the Administrator's Guide

### REENABLE\_DUPLICATE\_KEY\_ERROR

Restores the default behavior of error reporting by reversing the effects of `DISABLE_DUPLICATE_KEY_ERROR`. Effects are session scoped.

### Syntax

```
REENABLE_DUPLICATE_KEY_ERROR();
```

### Examples

For examples and usage see **DISABLE\_DUPLICATE\_KEY\_ERROR** (page 403).

### See Also

**ANALYZE\_CONSTRAINTS** (page 377)

### REFRESH

Performs a synchronous, optionally-targeted refresh of a specified table's projections.

Information about a refresh operation—whether successful or unsuccessful—is maintained in the **PROJECTION\_REFRESHES** (page 863) system table until either the **CLEAR\_PROJECTION\_REFRESHES** (page 395)() function is executed or the storage quota for the table is exceeded. The `PROJECTION_REFRESHES.IS_EXECUTING` column returns a boolean value that indicates whether the refresh is currently running (t) or occurred in the past (f).

### Syntax

```
REFRESH ( [schema_name.]table_name [ , ... ] )
```

### Parameters

<code>[schema_name.]table_name</code>	In the optionally-specified schema, <code>table_name</code> is the name of a specific table that contains the projections to be refreshed. When using more than one schema, specify the schema that
---------------------------------------	---

	contains the table.
--	---------------------

## Returns

Column Name	Description
Projection Name	The name of the projection that is targeted for refresh.
Anchor Table	The name of the projection's associated anchor table.
Status	The status of the projection: <ul style="list-style-type: none"> <li>▪ Queued — Indicates that a projection is queued for refresh.</li> <li>▪ Refreshing — Indicates that a refresh for a projection is in process.</li> <li>▪ Refreshed — Indicates that a refresh for a projection has successfully completed.</li> <li>▪ Failed — Indicates that a refresh for a projection did not successfully complete.</li> </ul>
Refresh Method	The method used to refresh the projection: <ul style="list-style-type: none"> <li>▪ Buddy – Uses the contents of a buddy to refresh the projection. This method maintains historical data. This enables the projection to be used for historical queries.</li> <li>▪ Scratch – Refreshes the projection without using a buddy. This method does not generate historical data. This means that the projection cannot participate in historical queries from any point before the projection was refreshed.</li> </ul>
Error Count	The number of times a refresh failed for the projection.
Duration (sec)	The length of time that the projection refresh ran in seconds.

## Notes

- Unlike START\_REFRESH(), which runs in the background, REFRESH() runs in the foreground of the caller's session.
- The REFRESH() function refreshes only the projections in the specified table.
- If you run REFRESH() without arguments, it refreshes all non up-to-date projections. If the function returns a header string with no results, then no projections needed refreshing.

## Example

The following command refreshes the projections in tables `t1` and `t2`:

```
=> SELECT REFRESH('t1, t2');
refresh
```

```
-----
Refresh completed with the following outcomes:
Projection Name: [Anchor Table] [Status] [Refresh Method] [Error Count] [Duration (sec)]
-----
"public"."t1_p": [t1] [refreshed] [scratch] [0] [0]
"public"."t2_p": [t2] [refreshed] [scratch] [0] [0]
```

This next command shows that only the projection on table `t` was refreshed:

```
=> SELECT REFRESH('allow, public.deny, t');"
refresh
```

```
-----
Refresh completed with the following outcomes:
Projection Name: [Anchor Table] [Status] [Refresh Method] [Error Count] [Duration (sec)]
-----
"n/a"."n/a": [n/a] [failed: insufficient permissions on table "allow"] [] [1] [0]
"n/a"."n/a": [n/a] [failed: insufficient permissions on table "public.deny"] [] [1] [0]
"public"."t_p1": [t] [refreshed] [scratch] [0] [0]
```

**See Also**

**CLEAR\_PROJECTION\_REFRESHES** (page 395)

**PROJECTION\_REFRESHES** (page 863)

**START\_REFRESH** (page 470)

Clearing PROJECTION\_REFRESHES History in the Administrator's Guide

**REGEXP\_COUNT**

Returns the number times a regular expression matches a string.

**Syntax**

```
REGEXP_COUNT( string, pattern [, position [, regexp_modifier ] ] )
```

**Parameters**

<i>string</i>	The string to be searched for matches.
<i>pattern</i>	The regular expression to search for within the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <b><i>Perl Regular Expressions Documentation</i></b> ( <a href="http://perldoc.perl.org/perlre.html">http://perldoc.perl.org/perlre.html</a> ) for details.
<i>position</i>	The number of characters from the start of the string where the function should start searching for matches. The default value, 1, means to start searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 starts searching for a match to the pattern that many characters into the string.

<i>regexp_modifier</i>	<p>A string containing one or more single-character flags that change how the regular expression is matched against the string:</p> <ul style="list-style-type: none"> <li>b      Treat strings as binary octets rather than UTF-8 characters.</li> <li>c      Forces the match to be case sensitive (the default).</li> <li>i      Forces the match to be case insensitive.</li> <li>m      Treats the string being matched as multiple lines. With this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Ordinarily, these operators only match the start and end of the string.</li> <li>n      Allows the single character regular expression operator (.) to match a newline (\n). Normally, the . operator will match any character except a newline.</li> <li>x      Allows you to document your regular expressions. It causes all unescaped space characters and comments in the regular expression to be ignored. Comments start with a hash character (#) and end with a newline. All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.</li> </ul>
------------------------	--

## Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

## Examples

Count the number of occurrences of the substring "an" in the string "A man, a plan, a canal, Panama."

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', 'an');
       REGEXP_COUNT
```

```
-----
                4
(1 row)
```

Find the number of occurrences of the substring "an" in the string "a man, a plan, a canal: Panama" starting with the fifth character.

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', 'an',5);
       REGEXP_COUNT
```

```
-----
              3
(1 row)
```

Find the number of occurrences of a substring containing a lower-case character followed by "an." In the first example, the query does not have a modifier. In the second example, the "i" query modifier is used to force the regular expression to ignore case.

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', '[a-z]an');
       REGEXP_COUNT
```

```
-----
              3
(1 row)
```

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', '[a-z]an', 1, 'i');
       REGEXP_COUNT
```

```
-----
              4
```

## REGEXP\_INSTR

Returns the starting or ending position in a string where a regular expression matches. This function returns 0 if no match for the regular expression is found in the string.

### Syntax

```
REGEXP_INSTR( string, pattern [, position [, occurrence
... [, return_position [, regexp_modifier ] ] ] ] )
```

### Parameters

<i>string</i>	The string to search for the pattern.
<i>pattern</i>	The regular expression to search for within the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <b>Perl Regular Expressions Documentation</b> ( <a href="http://perldoc.perl.org/perlre.html">http://perldoc.perl.org/perlre.html</a> ) for details.
<i>position</i>	The number of characters from the start of the string where the function should start searching for matches. The default value, 1, means to start searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 starts searching for a match to the pattern that many characters into the string.
<i>occurrence</i>	Controls which occurrence of a match between the string and the pattern is returned. With the default value (1), the function returns the position of the first substring that matches the pattern. You can use this parameter to find the position of additional matches between the string and the pattern. For example, set this parameter to 3 to find the position of the third substring that matched the pattern.
<i>return_position</i>	Sets the position within the string that is returned. When set to the default value (0), this function returns the

	position in the string of the first character of the substring that matched the pattern. If you set this value to 1, the function returns the position of the first character after the end of the matching substring.
<i>regexp_modifier</i>	<p>A string containing one or more single-character flags that change how the regular expression is matched against the string:</p> <ul style="list-style-type: none"> <li>b        Treat strings as binary octets rather than UTF-8 characters.</li> <li>c        Forces the match to be case sensitive (the default).</li> <li>i        Forces the match to be case insensitive.</li> <li>m        Treats the string being matched as multiple lines. With this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Ordinarily, these operators only match the start and end of the string.</li> <li>n        Allows the single character regular expression operator (.) to match a newline (\n). Normally, the . operator will match any character except a newline.</li> <li>x        Allows you to document your regular expressions. It causes all unescaped space characters and comments in the regular expression to be ignored. Comments start with a hash character (#) and end with a newline. All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.</li> </ul>

## Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

## Examples

Find the first occurrence of a sequence of letters starting with the letter e and ending with the letter y in the phrase "easy come, easy go."

```
=> SELECT REGEXP_INSTR('easy come, easy go','e\w*y');
```

```
REGEXP_INSTR
-----
1
(1 row)
```

Find the first occurrence of a sequence of letters starting with the letter e and ending with the letter y starting at the second character in the string "easy come, easy go."

```
=> SELECT REGEXP_INSTR('easy come, easy go', 'e\w*y', 2);
REGEXP_INSTR
-----
                12
(1 row)
```

Find the second sequence of letters starting with the letter e and ending with the letter y in the string "easy come, easy go" starting at the first character.

```
=> SELECT REGEXP_INSTR('easy come, easy go', 'e\w*y', 1, 2);
REGEXP_INSTR
-----
                12
(1 row)
```

Find the position of the first character after the first whitespace in the string "easy come, easy go."

```
=> SELECT REGEXP_INSTR('easy come, easy go', '\s', 1, 1, 1);
REGEXP_INSTR
-----
                6
(1 row)
```

## REGEXP\_LIKE

Returns true if the string matches the regular expression. This function is similar to the **LIKE-predicate** (page 59), except that it uses regular expressions rather than simple wildcard character matching.

### Syntax

```
REGEXP_LIKE( string, pattern [, modifiers ] )
```

### Parameters

<i>string</i>	The string to match against the regular expression.
<i>pattern</i>	A string containing the regular expression to match against the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <b>Perl Regular Expressions Documentation</b> ( <a href="http://perldoc.perl.org/perlre.html">http://perldoc.perl.org/perlre.html</a> ) for details.
<i>modifiers</i>	A string containing one or more single-character flags that change how the regular expression is matched against the string: <ul style="list-style-type: none"><li>b      Treat strings as binary octets rather than UTF-8 characters.</li><li>c      Forces the match to be case sensitive (the default).</li><li>i      Forces the match to be case insensitive.</li></ul>

m	Treats the string being matched as multiple lines. With this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Ordinarily, these operators only match the start and end of the string.
n	Allows the single character regular expression operator (.) to match a newline (\n). Normally, the . operator will match any character except a newline.
x	Allows you to document your regular expressions. It causes all unescaped space characters and comments in the regular expression to be ignored. Comments start with a hash character (#) and end with a newline. All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.

## Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

## Examples

This example creates a table containing several strings to demonstrate regular expressions.

```
=> create table t (v varchar);
CREATE TABLE
=> create projection t1 as select * from t;
CREATE PROJECTION
=> COPY t FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> aaa
>> Aaa
>> abc
>> abc1
>> 123
>> \.
=> SELECT * FROM t;
  v
-----
aaa
Aaa
abc
abc1
123
```

(5 rows)

Select all records in the table that contain the letter "a."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'a');
      v
-----
Aaa
aaa
abc
abc1
(4 rows)
```

Select all of the rows in the table that start with the letter "a."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, '^a');
      v
-----
aaa
abc
abc1
(3 rows)
```

Select all rows that contain the substring "aa."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'aa');
      v
-----
Aaa
aaa
(2 rows)
```

Select all rows that contain a digit.

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, '\d');
      v
-----
123
abc1
(2 rows)
```

Select all rows that contain the substring "aaa."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'aaa');
      v
-----
aaa
(1 row)
```

Select all rows that contain the substring "aaa" using case insensitive matching.

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'aaa', 'i');
      v
-----
```

```
Aaa
aaa
(2 rows)
```

Select rows that contain the substring "a b c."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'a b c');
v
-----
(0 rows)
```

Select rows that contain the substring "a b c" ignoring space within the regular expression.

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'a b c', 'x');
v
-----
abc
abc1
(2 rows)
```

Add multi-line rows to demonstrate using the "m" modifier.

```
=> COPY t FROM stdin RECORD TERMINATOR '!';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> Record 1 line 1
>> Record 1 line 2
>> Record 1 line 3!
>> Record 2 line 1
>> Record 2 line 2
>> Record 2 line 3!
>> \.
```

Select rows that start with the substring "Record" and end with the substring "line 2."

```
=> SELECT v from t WHERE REGEXP_LIKE(v, '^Record.*line 2$');
v
-----
(0 rows)
```

Select rows that start with the substring "Record" and end with the substring "line 2," treating multiple lines as separate strings.

```
=> SELECT v from t WHERE REGEXP_LIKE(v, '^Record.*line 2$', 'm');
v
-----
Record 2 line 1
Record 2 line 2
Record 2 line 3
Record 1 line 1
Record 1 line 2
Record 1 line 3
(2 rows)
```

## REGEXP\_REPLACE

Replace all occurrences of a substring that match a regular expression with another substring. It is similar to the **REPLACE** (page 346) function, except it uses a regular expression to select the substring to be replaced.

### Syntax

```
REGEXP_REPLACE( string, target [, replacement [, position [, occurrence
... [, regexp_modifiers ] ] ] ] )
```

### Parameters

<i>string</i>	The string whose to be searched and replaced.
<i>target</i>	The regular expression to search for within the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <b>Perl Regular Expressions Documentation</b> ( <a href="http://perldoc.perl.org/perlre.html">http://perldoc.perl.org/perlre.html</a> ) for details.
<i>replacement</i>	The string to replace matched substrings. If not supplied, the matched substrings are deleted. This string can contain backreferences for substrings captured by the regular expression. The first captured substring is inserted into the replacement string using <code>\1</code> , the second <code>\2</code> , and so on.
<i>position</i>	The number of characters from the start of the string where the function should start searching for matches. The default value, 1, means to start searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 starts searching for a match to the pattern that many characters into the string.
<i>occurrence</i>	Controls which occurrence of a match between the string and the pattern is replaced. With the default value (0), the function replaces all matching substrings with the replacement string. For any value above zero, the function replaces just a single occurrence. For example, set this parameter to 3 to replace the third substring that matched the pattern.
<i>regexp_modifier</i>	A string containing one or more single-character flags that change how the regular expression is matched against the string: <ul style="list-style-type: none"> <li>b      Treat strings as binary octets rather than UTF-8 characters.</li> <li>c      Forces the match to be case sensitive (the default).</li> <li>i      Forces the match to be case insensitive.</li> </ul>

m	Treats the string being matched as multiple lines. With this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Ordinarily, these operators only match the start and end of the string.
n	Allows the single character regular expression operator (.) to match a newline (\n). Normally, the . operator will match any character except a newline.
x	Allows you to document your regular expressions. It causes all unescaped space characters and comments in the regular expression to be ignored. Comments start with a hash character (#) and end with a newline. All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.

## Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

## Examples

Find groups of "word characters" (letters, numbers and underscore) ending with "thy" in the string "healthy, wealthy, and wise" and replace them with nothing.

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise', '\w+thy');
       REGEXP_REPLACE
-----
, , and wise
(1 row)
```

Find groups of word characters ending with "thy" and replace with the string "something."

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise', '\w+thy', 'something');
       REGEXP_REPLACE
-----
something, something, and wise
(1 row)
```

Find groups of word characters ending with "thy" and replace with the string "something" starting at the third character in the string.

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise', '\w+thy', 'something', 3);
       REGEXP_REPLACE
-----
```

```
hesomething, something, and wise
(1 row)
```

Replace the second group of word characters ending with "thy" with the string "something."

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise', '\w+thy', 'something', 1,
2);
          REGEXP_REPLACE
-----
healthy, something, and wise
(1 row)
```

Find groups of word characters ending with "thy" capturing the letters before the "thy", and replace with the captured letters plus the letters "ish."

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise', '(\w+)thy', '\1ish');
          REGEXP_REPLACE
-----
healish, wealish, and wise
(1 row)
```

Create a table to demonstrate replacing strings in a query.

```
=> CREATE TABLE customers (name varchar(50), phone varchar(11));
CREATE TABLE
=> CREATE PROJECTION customers1 AS SELECT * FROM customers;
CREATE PROJECTION
=> COPY customers FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> Able, Adam|17815551234
>> Baker,Bob|18005551111
>> Chu,Cindy|16175559876
>> Dodd,Dinara|15083452121
>> \.
```

Query the customers, using REGEXP\_REPLACE to format the phone numbers.

```
=> SELECT name, REGEXP_REPLACE(phone, '(\d)(\d{3})(\d{3})(\d{4})', '\1-(\2)
\3-\4') as phone FROM customers;
   name   |      phone
-----+-----
Able, Adam | 1-(781) 555-1234
Baker,Bob  | 1-(800) 555-1111
Chu,Cindy  | 1-(617) 555-9876
Dodd,Dinara | 1-(508) 345-2121
(4 rows)
```

## REGEXP\_SUBSTR

Returns the substring that matches a regular expression within a string. If no matches are found, this function returns NULL. This is different than an empty string, which can be returned by this function if the regular expression matches a zero-length string.

### Syntax

```
REGEXP_SUBSTR( string, pattern [, position [, occurrence
... [, regexp_modifier ] ] ] )
```

### Parameters

<i>string</i>	The string to search for the pattern.
<i>pattern</i>	The regular expression to find the substring to be extracted. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <b><i>Perl Regular Expressions Documentation</i></b> ( <a href="http://perldoc.perl.org/perlre.html">http://perldoc.perl.org/perlre.html</a> ) for details.
<i>position</i>	The character in the string where the search for a match should start. The default value, 1, starts the search at the beginning of the string. If you supply a value larger than 1 for this parameter, the function will start searching that many characters into the string.
<i>occurrence</i>	Controls which matching substring is returned by the function. When given the default value (1), the function will return the first matching substring it finds in the string. By setting this value to a number greater than 1, this function will return subsequent matching substrings. For example, setting this parameter to 3 will return the third substring that matches the regular expression within the string.
<i>regexp_modifier</i>	<p>A string containing one or more single-character flags that change how the regular expression is matched against the string:</p> <ul style="list-style-type: none"> <li>b      Treat strings as binary octets rather than UTF-8 characters.</li> <li>c      Forces the match to be case sensitive (the default).</li> <li>i      Forces the match to be case insensitive.</li> <li>m      Treats the string being matched as multiple lines. With this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Ordinarily, these operators only match the start and end of the string.</li> <li>n      Allows the single character regular expression operator (.) to match a newline (\n). Normally, the . operator will match any character except a newline.</li> </ul>

	x	Allows you to document your regular expressions. It causes all unescaped space characters and comments in the regular expression to be ignored. Comments start with a hash character (#) and end with a newline. All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.
--	---	--

## Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

## Examples

Select the first substring of letters that end with "thy."

```
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise', '\w+thy');
REGEXP_SUBSTR
-----
healthy
(1 row)
```

Select the first substring of letters that ends with "thy" starting at the second character in the string.

```
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise', '\w+thy', 2);
REGEXP_SUBSTR
-----
ealthy
(1 row)
```

Select the second substring of letters that ends with "thy."

```
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise', '\w+thy', 1, 2);
REGEXP_SUBSTR
-----
wealthy
(1 row)
```

## REMOVE\_DATABASE\_SNAPSHOT

Removes a snapshot from the catalog directory.

### Syntax

```
REMOVE_DATABASE_SNAPSHOT( name )
```

**Parameters**

<i>name</i>	The name of the snapshot to remove
-------------	------------------------------------

**See Also**

***DATABASE\_SNAPSHOT*** (page 493)

***DATABASE\_SNAPSHOTS*** (page 842) system table

Creating Database Snapshots in the Administrator's Guide

**RESTORE\_LOCATION**

Restores the retired location specified.

**Syntax**

```
RESTORE_LOCATION ( 'path' , 'node' )
```

**Parameters**

<i>path</i>	Specifies where the retired storage location is mounted.
<i>node</i>	Is the Vertica node where the retired location is available.

**Notes**

Once restored, Vertica re-ranks the storage locations and use the restored location to process queries as determined by its rank.

**Example**

The following example restores the retired storage location on node3:

```
=> SELECT RESTORE_LOCATION ('/thirdVerticaStorageLocation/' , 'node3');
```

**See Also**

***ADD\_LOCATION*** (page 374)

***RETIRE\_LOCATION*** (page 461)

Modifying Storage Locations in the Administrator's Guide

**RETIRE\_LOCATION**

Makes the specified storage location inactive.

**Syntax**

```
RETIRE_LOCATION ( 'path' , 'node' )
```

**Parameters**

<i>path</i>	Specifies where the storage location to retire is mounted.
-------------	--

<i>node</i>	Is the Vertica node where the location is available.
-------------	--

### Notes

- Before retiring a location, be sure that at least one location remains for storing data and temp files. Data and temp files can be stored in either one storage location or separate storage locations.
- Once retired, no new data can be stored on the location unless the location is restored through the **RESTORE\_LOCATION()** (page 461) function.
- If the storage location stored data, the data is not moved. Instead, it is removed through one or more mergeouts. Therefore, the location cannot be dropped.
- If the storage site was used to store only temp files, it can be dropped. See Dropping Storage Locations in the Administrators Guide and the **DROP\_LOCATION()** (page 407) function.

### Example

```
=> SELECT RETIRE_LOCATION ('/secondVerticaStorageLocation/' , 'node2');
```

### See Also

**ADD\_LOCATION** (page 374)

**RESTORE\_LOCATION** (page 461)

Retiring Storage Locations in the Administrator's Guide

### SAVE\_QUERY\_REPOSITORY

Triggers Vertica to save query data to the query repository immediately.

### Syntax

```
SAVE_QUERY_REPOSITORY()
```

### Notes

- Vertica saves data based on the established query repository configuration parameters. For example, it will use the value of the QueryRepoRetentionTime parameter to determine the maximum number of days worth of queries to save. (See Configuring Query Repository in the Troubleshooting Guide.)
- Before you can save a query repository, you have to enable it:

```
SELECT SET_CONFIG_PARAMETER('QueryRepositoryEnabled',1);
```

### Example

```
=> SELECT SAVE_QUERY_REPOSITORY();
SAVE_QUERY_REPOSITORY
-----
Query Repository Saved
(1 row)
```

### See Also

**V\_MONITOR.QUERY\_METRICS** (page 868)

**V\_MONITOR.QUERY\_PROFILES** (page 870)

Collecting Query Information in the Troubleshooting Guide

**SET\_AHM\_EPOCH**

Sets the Ancient History Mark (AHM) to the specified epoch. This function allows deleted data up to and including the AHM epoch to be purged from physical storage.

SET\_AHM\_EPOCH is normally used for testing purposes. Consider **SET\_AHM\_TIME** (page 464) instead, which is easier to use.

**Syntax**

```
SET_AHM_EPOCH ( epoch, [ true ] )
```

**Parameters**

<i>epoch</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>▪ The number of the epoch in which to set the AHM</li> <li>▪ Zero (0) (the default) disables <b>purge</b> (page 444)</li> </ul>
<i>true</i>	Optionally allows the AHM to advance when nodes are down. <b>Note:</b> If the AHM is advanced after the last good epoch of the failed nodes, those nodes must recover all data from scratch. Use with care.

**Notes**

If you use SET\_AHM\_EPOCH , the number of the specified epoch must be:

- Greater than the current AHM epoch
- Less than the current epoch
- Less than or equal to the cluster last good epoch (the minimum of the last good epochs of the individual nodes in the cluster)
- Less than or equal to the cluster refresh epoch (the minimum of the refresh epochs of the individual nodes in the cluster)

Use the **SYSTEM** (page 898) table to see current values of various epochs related to the AHM; for example:

```
=> SELECT * from SYSTEM;
-[ RECORD 1 ]-----+-----
current_timestamp   | 2009-08-11 17:09:54.651413
current_epoch       | 1512
ahm_epoch           | 961
last_good_epoch     | 1510
refresh_epoch       | -1
designed_fault_tolerance | 1
node_count          | 4
node_down_count     | 0
current_fault_tolerance | 1
catalog_revision_number | 1590
wos_used_bytes      | 0
wos_row_count       | 0
```

```
ros_used_bytes          | 41490783
ros_row_count           | 1298104
total_used_bytes        | 41490783
total_row_count         | 1298104
```

All nodes must be up. You cannot use `SET_AHM_EPOCH` when any node in the cluster is down, except by using the optional `true` parameter.

When a node is down and you issue `SELECT MAKE_AHM_NOW()`, the following error is printed to the `vertica.log`:

```
Some nodes were excluded from setAHM. If their LGE is before the AHM they will perform full recovery.
```

### Examples

The following command sets the AHM to a specified epoch of 12:

```
=> SELECT SET_AHM_EPOCH(12);
```

The following command sets the AHM to a specified epoch of 2 and allows the AHM to advance despite a failed node:

```
=> SELECT SET_AHM_EPOCH(2, true);
```

### See Also

***MAKE\_AHM\_NOW*** (page 434)

***SET\_AHM\_TIME*** (page 464)

***SYSTEM*** (page 898)

### SET\_AHM\_TIME

Sets the Ancient History Mark (AHM) to the epoch corresponding to the specified time on the initiator node. This function allows historical data up to and including the AHM epoch to be purged from physical storage.

### Syntax

```
SET_AHM_TIME ( time , [ true ] )
```

### Parameters

<i>time</i>	Is a <b><i>TIMESTAMP</i></b> (page 90) value that is automatically converted to the appropriate epoch number.
<i>true</i>	[Optional] Allows the AHM to advance when nodes are down. <b>Note:</b> If the AHM is advanced after the last good epoch of the failed nodes, those nodes must recover all data from scratch.

### Notes

- `SET_AHM_TIME` returns a `TIMESTAMP WITH TIME ZONE` value representing the end point of the AHM epoch.
- You cannot change the AHM when any node in the cluster is down, except by using the optional `true` parameter.

- When a node is down and you issue `SELECT MAKE_AHM_NOW()`, the following error is printed to the vertica.log:

```
Some nodes were excluded from setAHM. If their LGE is before the AHM they
will perform full recovery.
```

## Examples

Epochs depend on a configured epoch advancement interval. If an epoch includes a three-minute range of time, the purge operation is accurate only to within minus three minutes of the specified timestamp:

```
=> SELECT SET_AHM_TIME('2008-02-27 18:13');
       set_ahm_time
```

```
-----
AHM set to '2008-02-27 18:11:50-05'
(1 row)
```

**Note:** The -05 part of the output string is a time zone value, an offset in hours from UTC (Universal Coordinated Time, traditionally known as Greenwich Mean Time, or GMT).

In the above example, the actual AHM epoch ends at 18:11:50, roughly one minute before the specified timestamp. This is because `SET_AHM_TIME` selects the epoch that ends at or before the specified timestamp. It does not select the epoch that ends after the specified timestamp because that would purge data deleted as much as three minutes after the AHM.

For example, using only hours and minutes, suppose that epoch 9000 runs from 08:50 to 11:50 and epoch 9001 runs from 11:50 to 15:50. `SET_AHM_TIME('11:51')` chooses epoch 9000 because it ends roughly one minute before the specified timestamp.

In the next example, if given an environment variable set as `date = `date``; the following command fails if a node is down:

```
=> SELECT SET_AHM_TIME('$date');
```

In order to force the AHM to advance, issue the following command instead:

```
=> SELECT SET_AHM_TIME('$date', true);
```

## See Also

**MAKE\_AHM\_NOW** (page 434)

**SET\_AHM\_EPOCH** (page 463) for a description of the range of valid epoch numbers.

**SET\_DATESTYLE** (page 767) for information about specifying a **TIMESTAMP** (page 90) value.

## SET\_AUDIT\_TIME

Sets the time that Vertica performs automatic database size audit to determine if the size of the database is compliant with the raw data allowance in your Vertica license. You may want to use this function if the audits are currently scheduled to occur during your database's peak activity time. This is normally not a concern, since the automatic audit has little impact on database performance.

---

**Note:** Audits are scheduled by the preceding audit, so changing the audit time does not affect the next scheduled audit. For example, if your next audit is scheduled to take place at

---

11:59PM and you use `SET_AUDIT_TIME` to change the audit schedule 3AM, the previously scheduled 11:59PM audit still runs. As that audit finishes, it schedules the next audit to occur at 3AM.

If you want to prevent the next scheduled audit from running at its scheduled time, you can change the scheduled time using `SET_AUDIT_TIME` then manually trigger an audit to run immediately using **`AUDIT_LICENSE_SIZE`** (page 394). As the manually-triggered audit finishes, it schedules the next audit to occur at the time you set using `SET_AUDIT_TIME` (effectively overriding the previously scheduled audit).

## Syntax

```
SET_AUDIT_TIME(time)
```

<i>time</i>	A string containing the time in 'HH:MM AM/PM' format (for example, '1:00 AM') when the audit should run daily.
-------------	--

## Example

```
=> SELECT SET_AUDIT_TIME('3:00 AM');
           SET_AUDIT_TIME
```

-----  
 The scheduled audit time will be set to 3:00 AM after the next audit.  
 (1 row)

## SET\_DATA\_COLLECTOR\_POLICY

Sets the retention policy for the specified component on all nodes. Failed nodes receive the setting when they rejoin the cluster.

## Syntax

```
SET_DATA_COLLECTOR_POLICY('component', 'memoryKB', 'diskKB' )
```

## Parameters

<i>component</i>	Returns the retention policy for the specified component.
<i>memoryKB</i>	Specifies the memory size to retain in kilobytes.
<i>diskKB</i>	Specifies the disk size in kilobytes.

## Notes

- Only the superuser can configure the Data Collector.
- Before you change a retention policy, you can see what the current setting is by calling the `GET_DATA_COLLECTOR_POLICY()` function.
- For a list of all current component names, query the **`V_MONITOR.DATA_COLLECTOR`** (page 837) system table; for example:

```
=> SELECT DISTINCT component, description FROM data_collector
      ORDER BY 1 ASC;
```

## Example

The following example returns the retention policy for the ResourceAcquisitions component:

```
=> SELECT get_data_collector_policy('ResourceAcquisitions');
           get_data_collector_policy
-----
1000KB kept in memory, 10000KB kept on disk.
(1 row)
```

The next command changes the memory and disk setting for ResourceAcquisitions to 1500KB and 25000KB, respectively:

```
=> SELECT set_data_collector_policy('ResourceAcquisitions', '1500', '25000');
           set_data_collector_policy
-----
SET
(1 row)
```

To verify the setting you just made, call the GET\_DATA\_COLLECTOR\_POLICY() function on the specified component:

```
=> SELECT get_data_collector_policy('ResourceAcquisitions');
           get_data_collector_policy
-----
1500KB kept in memory, 25000KB kept on disk.
(1 row)
```

## See Also

**V\_MONITOR.DATA\_COLLECTOR** (page 837)

Retaining Monitoring Information in the Troubleshooting Guide

## SET\_LOCATION\_PERFORMANCE

Sets disk performance for the location specified.

### Syntax

```
SET_LOCATION_PERFORMANCE ( 'path' , 'node' , 'throughput' , 'average_latency' )
```

### Parameters

<i>node</i>	Is the Vertica node where the location to be set is available. If this parameter is omitted, <i>node</i> defaults to the initiator.
<i>path</i>	Specifies where the storage location to set is mounted.
<i>throughput</i>	Specifies the throughput for the location, which must be 1 or more.
<i>average_latency</i>	Specifies the average latency for the location. The <i>average_latency</i> must be 1 or more.

## Notes

To obtain the throughput and average latency for the location, run the **MEASURE\_LOCATION\_PERFORMANCE()** (page 438) function before you attempt to set the location's performance.

## Example

The following example sets the performance of a storage location on node2 to a throughput of 122 megabytes per second and a latency of 140 seeks per second.

```
=> SELECT MEASURE_LOCATION_PERFORMANCE ('node2','/secondVerticaStorageLocation/', '122', '140');
```

## See Also

**ADD\_LOCATION** (page 374)

**MEASURE\_LOCATION\_PERFORMANCE** (page 438)

Measuring Location Performance and Setting Location Performance in the Administrator's Guide

## SET\_SCALING\_FACTOR

Sets the scaling factor used to determine the size of the storage containers used when rebalancing the database and when using local data segmentation is enabled. See Cluster Scaling for details.

## Syntax

```
SET_SCALING_FACTOR(factor)
```

## Parameters

<i>factor</i>	An integer value between 1 and 32. Vertica uses this value to calculate the number of storage containers each projection is broken into when rebalancing or when local data segmentation is enabled.
---------------	--

**Note:** Setting this value too high can cause nodes to create too many small container files, greatly reducing efficiency and potentially causing a Too Many ROS containers error. In general, you should only set this value manually to enable data scaling on a database where it was previously disabled, or if you have found the amount of data skew (uneven distribution of data between the nodes) in your database to be unacceptable. You may find this to be the case if you have expanded your cluster from a power of two number of nodes to a non-power of two number of nodes.

## Example

```
=> SELECT SET_SCALING_FACTOR(12);
   SET_SCALING_FACTOR
-----
      SET
(1 row)
```

## SHUTDOWN

Forces a database to shut down, even if there are users connected.

### Syntax

```
SHUTDOWN ( [ 'false' | 'true' ] )
```

### Parameters

<i>false</i>	[Default] Returns a message if users are connected. Has the same effect as supplying no parameters.
<i>true</i>	Performs a moveout operation and forces the database to shut down, disallowing further connections.

### Notes

- Quotes around the `true` or `false` arguments are optional.
- Issuing the shutdown command without arguments or with the default (`false`) argument returns a message if users are connected, and the shutdown fails. If no users are connected, the database performs a moveout operation and shuts down.
- Issuing the `SHUTDOWN('true')` command forces the database to shut down whether users are connected or not.
- You can check the status of the shutdown operation in the `vertica.log` file:  
2010-03-09 16:51:52.625 unknown:0x7fc6d6d2e700 [Init] <INFO> Shutdown complete. Exiting.
- As an alternative to `SHUTDOWN()`, you can also temporarily set `MaxClientSessions` to 0 and then use `CLOSE_ALL_SESSIONS()`. New client connections cannot connect unless they connect using the `dbadmin` account. See **`CLOSE_ALL_SESSIONS`** (page 399) for details.

### Examples

The following command attempts to shut down the database. Because users are connected, the command fails:

```
=> SELECT SHUTDOWN('false');
NOTICE:  Cannot shut down while users are connected
        SHUTDOWN
-----
Shutdown: aborting shutdown
(1 row)
```

Note that `SHUTDOWN()` and `SHUTDOWN('false')` perform the same operation:

```
=> SELECT SHUTDOWN();
NOTICE:  Cannot shut down while users are connected
        SHUTDOWN
-----
Shutdown: aborting shutdown
(1 row)
```

Using the `'true'` parameter forces the database to shut down, even though clients might be connected:

```
=> SELECT SHUTDOWN('true');
       SHUTDOWN
```

```
-----
Shutdown: moveout complete
(1 row)
```

### See Also

**SESSIONS** (page 888)

### START\_REFRESH

Transfers data to projections that are not able to participate in query execution due to missing or out-of-date data.

### Syntax

```
START_REFRESH()
```

### Notes

- When a design is deployed through the Database Designer, it is automatically refreshed. See Deploying Designs in the Administrator's Guide.
- All nodes must be up in order to start a refresh.
- START\_REFRESH() has no effect if a refresh is already running.
- A refresh is run asynchronously.
- Shutting down the database ends the refresh.
- To view the progress of the refresh, see the **PROJECTION\_REFRESHES** (page 863) and **PROJECTIONS** (page 811) system tables.
- If a projection is updated from scratch, the data stored in the projection represents the table columns as of the epoch in which the refresh commits. As a result, the query optimizer might not choose the new projection for AT EPOCH queries that request historical data at epochs older than the refresh epoch of the projection. Projections refreshed from buddies retain history and can be used to answer historical queries.

Vertica internal operations (mergeout, refresh, and recovery) maintain partition separation except in certain cases:

- Recovery of a projection when the buddy projection from which the partition is recovering is identically sorted. If the projection is undergoing a full rebuild, it is recovered one ROS container at a time. The projection ends up with a storage layout identical to its buddy and is, therefore, properly segmented.

**Note:** In the case of a partial rebuild, all recovered data goes into a single ROS container and must be partitioned manually.

- Manual tuple mover operations often output a single storage container, combining any existing partitions; for example, after executing any of the PURGE() operations.

### Example

The following command starts the refresh operation:

```
=> SELECT START_REFRESH();
       start_refresh
```

-----  
Starting refresh background process.

### See Also

**CLEAR\_PROJECTION\_REFRESHES** (page 395)

**MARK\_DESIGN\_KSAFE** (page 436)

**PROJECTION\_REFRESHES** (page 863)

**PROJECTIONS** (page 811)

Clearing PROJECTION\_REFRESHES History in the Administrator's Guide

## TS\_FIRST\_VALUE

Processes the data that belongs to each time slice. A time series aggregate (TSA) function, `TS_FIRST_VALUE` returns the value at the start of the time slice, where an interpolation scheme is applied if the timeslice is missing, in which case the value is determined by the values corresponding to the previous (and next) timeslices based on the interpolation scheme of `const` (linear). There is one value per time slice per partition.

### Behavior Type

Immutable

### Syntax

```
TS_FIRST_VALUE ( expression [ IGNORE NULLS ]
... [, { 'CONST' | 'LINEAR' } ] )
```

### Parameters

<i>expression</i>	Is the argument expression on which to aggregate and interpolate. <i>expression</i> is data type INTEGER or FLOAT.
IGNORE NULLS	The IGNORE NULLS behavior changes depending on a CONST or LINEAR interpolation scheme. See When Time Series Data Contains Nulls in the Programmer's Guide for details.
'CONST'   'LINEAR'	Optionally specifies the interpolation value as either constant or linear. The default is constant. If omitted, Vertica defaults to CONST, but you can also specify CONST.

### Notes

- The function returns one output row per time slice or one output row per partition per time slice if partition expressions are specified.

- Multiple time series aggregate functions can exist in the same query. They share the same gap-filling policy as defined by the ***TIMESERIES clause*** (page 761); however, each time series aggregate function can specify its own interpolation policy. For example:

```
SELECT slice_time, symbol,
       TS_FIRST_VALUE (bid, 'const') fv_c,
       TS_FIRST_VALUE (bid, 'linear') fv_l,
       TS_LAST_VALUE (bid, 'const') lv_c
FROM TickStore
TIMESERIES slice_time AS '3 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```

- You must use an ORDER BY clause with a timestamp column.

### Example

For detailed examples, see Gap Filling and Interpolation and When Time Series Data Contains Nulls in the Programmer's Guide.

### See Also

***TIMESERIES Clause*** (page 761) and ***TS\_LAST\_VALUE*** (page 472)

Using Time Series Analytics in the Programmer's Guide

## TS\_LAST\_VALUE

Processes the data that belongs to each time slice. A time series aggregate (TSA) function, ***TS\_LAST\_VALUE*** returns the value at the end of the time slice, where an interpolation scheme is applied if the timeslice is missing, in which case the value is determined by the values corresponding to the previous (and next) timeslices based on the interpolation scheme of ***const*** (***linear***). There is one value per time slice per partition.

### Behavior Type

Immutable

### Syntax

```
TS_LAST_VALUE ( expression [ IGNORE NULLS ]
... [, { 'CONST' | 'LINEAR' } ] )
```

### Parameters

<i>expression</i>	Is the argument expression on which to aggregate and interpolate. <i>expression</i> is data type INTEGER or FLOAT.
IGNORE NULLS	The IGNORE NULLS behavior changes depending on a CONST or LINEAR interpolation scheme. See When Time Series Data Contains Nulls in the Programmer's Guide for details.
'CONST'   'LINEAR'	Optionally specifies the interpolation value as either constant or linear. The default is constant. If omitted, Vertica defaults to CONST, but you can also specify CONST.

## Notes

- The function returns one output row per time slice or one output row per partition per time slice if partition expressions are specified.
- Multiple time series aggregate functions can exist in the same query. They share the same gap-filling policy as defined by the ***TIMESERIES clause*** (page 761); however, each time series aggregate function can specify its own interpolation policy. For example:

```
SELECT slice_time, symbol,
       TS_FIRST_VALUE (bid, 'const') fv_c,
       TS_FIRST_VALUE (bid, 'linear') fv_l,
       TS_LAST_VALUE (bid, 'const') lv_c
FROM TickStore
TIMESERIES slice_time AS 3 seconds OVER (PARTITION BY symbol ORDER BY ts);
```

- If you use the `window_order_clause`, you can order by a `TIMESTAMP` column only, not, for example, an `INTEGER` column.

## Example

For detailed examples, see Gap Filling and Interpolation and When Time Series Data Contains Nulls in the Programmer's Guide.

## See Also

***TIMESERIES Clause*** (page 761) and ***TS\_FIRST\_VALUE*** (page 471)

Using Time Series Analytics in the Programmer's Guide

## Catalog Management Functions

This section contains catalog management functions specific to Vertica.

## DUMP\_CATALOG

Returns an internal representation of the Vertica catalog. This function is used for diagnostic purposes.

### Syntax

```
DUMP_CATALOG()
```

### Notes

To obtain an internal representation of the Vertica catalog for diagnosis, run the query:

```
SELECT DUMP_CATALOG();
```

The output is written to the specified file:

```
\o /tmp/catalog.txt
SELECT DUMP_CATALOG();
\o
```

Send the output to **Technical Support** (on page 1).

## EXPORT\_CATALOG

Generates a SQL script that can be used to recreate a physical schema design in its current state on a different cluster.

### Syntax

```
EXPORT_CATALOG ( [ 'destination' ] , [ 'type' ] )
```

### Parameters

<i>destination</i>	Specifies the path and name of the SQL output file. An empty string ( ' ' ), which is the default, dumps the script to standard output. A user who is not a DBA can only specify an empty string.
<i>type</i>	Determines what is exported: <ul style="list-style-type: none"> <li>▪ DESIGN — Exports schemas, tables, constraints, views, and projections to which the user has access. This is the default value.</li> <li>▪ DESIGN_ALL — Exports all the design objects plus system objects created in Database Designer (for example, design contexts and their tables). The objects that are exported are only the ones to which the user has access.</li> <li>▪ TABLES — Exports all tables, constraints, and projections for those tables for which the user has permissions. See also <b>EXPORT_TABLES</b> (page 424).</li> </ul>

### Notes

- Exporting a design is useful for quickly moving a design to another cluster.

- The script generated by this function:
  - Creates only the non-virtual objects for which the user has access.
  - Automatically runs **MARK\_DESIGN\_KSAFE()** (page 436) with the correct K-safety value to ensure the design copy has the same same K-safety value as the original design.
  - Exports catalog objects in their Oid order.
- If a projection is created with no sort order, Vertica implicitly assigns a sort order based on the SELECT columns in the projection definition. The sort order is explicitly defined in the exported script.

### Restrictions

The export script Vertica generates is portable as long as all the projections were generated using UNSEGMENTED ALL NODES or SEGMENTED ALL NODES. Projections might not exist on ALL NODES for the following reasons:

- A projection was dropped from a node.
- A projection was created only on a subset of nodes.
- An additional node was added since the projection set was created and the design wasn't extended through Database Designer deployment.

### Example

The following example exports the design to standard output:

```
=> SELECT EXPORT_CATALOG(' ', 'DESIGN');
```

### EXPORT\_OBJECTS

Generates a SQL script that can be used to recreate catalog objects on a different cluster.

#### Syntax

```
EXPORT_OBJECTS( [ 'destination' ] , [ 'scope' ] , [ 'bool_value' ] )
```

#### Parameters

<i>destination</i>	Specifies the path and name of the SQL output file. An empty string (''), which is the default, dumps the script to standard output. A user who is not a DBA can only specify an empty string.
--------------------	--

<i>scope</i>	<p>Determines the set of catalog objects to be exported where scope is one of the following:</p> <ul style="list-style-type: none"> <li>▪ an empty string ( ' ')—exports all non-virtual objects to which the user has access, including constraints. (Note that constraints are not objects which can be passed as individual arguments.) This is the default if no scope is specified.</li> <li>▪ a comma-delimited list of items in which each item can be one of the following:</li> <li>▪ —'&lt;schema&gt;.&lt;obj&gt;'—matches the named object. The named object can be a table, projection, or view.</li> <li>▪ —'&lt;obj&gt;'—matches the named object within the current search path. The named object can be a schema, table, projection, or view. If the named object is a schema, Vertica exports all non-virtual objects to which the user has access within that schema. If a schema and table both have the same name, the schema takes precedence.</li> </ul> <p>EXPORT_OBJECTS returns an error if:</p> <ul style="list-style-type: none"> <li>▪ an explicitly-specified object does not exist.</li> <li>▪ the user has no access to the specified object.</li> </ul>
<i>bool-value</i>	<p>Use one of the following:</p> <ul style="list-style-type: none"> <li>▪ true—incorporates a MAKE_DESIGN_KSAFE statement with the correct K-safety value for the database at the end of the output script.</li> <li>▪ false—omits the MAKE_DESIGN_KSAFE statement from the script.</li> </ul> <p>Adding the MAKE_DESIGN_KSAFE statement is useful if you are planning to import the script into a new database and you want the new database to inherit the K-safety value from the original database.</p> <p>By default, this parameter is true.</p>

## Notes

- The script generated by this function:
  - Creates only the non-virtual objects for which the user has access.
  - Exports catalog objects in their Oid order.
- None of the parameters for EXPORT\_OBJECTS accepts a NULL value as input.

## Example

The following example exports the all the non-virtual objects to which the user has access to standard output. It does not incorporate the MAKE\_DESIGN\_KSAFE statement at the end of the file.

```
=> SELECT EXPORT_OBJECTS(' ', ' ', false);
```

## INSTALL\_LICENSE

Installs the license key in the global catalog.

description

**Syntax**

```
INSTALL_LICENSE( 'filename' )
```

**Parameters**

<i>filename</i>	specifies the absolute pathname of a valid license file.
-----------------	--

**Notes**

See Managing Your License Key in the Administrator's Guide for more information about license keys.

**Examples**

```
SELECT INSTALL_LICENSE('/tmp/vlicense.dat');
```

## MARK\_DESIGN\_KSAFE

Enables or disables high availability in your environment, in case of a failure. Before enabling recovery, MARK\_DESIGN\_KSAFE queries the catalog to determine whether a cluster's physical schema design meets the following requirements:

- Dimension tables are replicated on all nodes.
- Fact table superprojections are segmented with each segment on a different node.
- Each fact table projection has at least one buddy projection for K-safety=1 (or two buddy projections for K-safety=2).

Buddy projections are also segmented across database nodes, but the distribution is modified so that segments that contain the same data are distributed to different nodes. See High Availability Through Projections in the Concepts Guide.

**Note:** Projections are considered to be buddies if they contain the same columns and have the same segmentation. They can have different sort orders.

MARK\_DESIGN\_KSAFE does not change the physical schema in any way.

### Syntax

```
SELECT MARK_DESIGN_KSAFE ( k )
```

### Parameters

<i>k</i>	<p>2 enables high availability if the schema design meets requirements for K-safety=2</p> <p>1 enables high availability if the schema design meets requirements for K-safety=1</p> <p>0 disables high availability</p>
----------	---

If you specify a *k* value of one (1) or two (2), Vertica returns one of the following messages.

#### Success:

```
Marked design n-safe
```

#### Failure:

```
The schema does not meet requirements for K=n.
Fact table projection projection-name
has insufficient "buddy" projections.
```

*n* in the message is 1 or 2 and represents the *k* value.

### Notes

- The database's internal recovery state persists across database restarts but it is not checked at startup time.
- If a database has automatic recovery enabled, you must temporarily disable automatic recovery before creating a new table.
- When one node fails on a system marked K-safe=1, the remaining nodes are available for DML operations.

## Examples

```
=> SELECT MARK_DESIGN_KSAFE(1);
      mark_design_ksafe
```

```
-----
      Marked design 1-safe
      (1 row)
```

If the physical schema design is not K-Safe, messages indicate which projections do not have a buddy:

```
=> SELECT MARK_DESIGN_KSAFE(1);
The given K value is not correct; the schema is 0-safe
Projection pp1 has 0 buddies, which is smaller than the given K of 1
Projection pp2 has 0 buddies, which is smaller than the given K of 1
.
.
.
(1 row)
```

## See Also

**SYSTEM** (page 898)

High Availability and Recovery in the Concepts Guide

**SQL System Tables (Monitoring APIs)** (page 794) topic in the Administrator's Guide

Using Identically Segmented Projections in the Programmer's Guide

Failure Recovery in the Troubleshooting Guide

## Cluster Scaling Functions

This section contains functions that control how the cluster organizes data for rebalancing.

### DISABLE\_LOCAL\_SEGMENTS

Disable local data segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes. See Local Data Segmentation in the Administrator's Guide for details.

#### Syntax

```
DISABLE_LOCAL_SEGMENTS()
```

#### Example

```
=> SELECT DISABLE_LOCAL_SEGMENTS();
      DISABLE_LOCAL_SEGMENTS
      -----
      DISABLED
      (1 row)
```

## ENABLE\_LOCAL\_SEGMENTS

Enables local storage segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes. See Local Data Segmentation in the Administrator's Guide for more information.

### Syntax

```
ENABLE_LOCAL_SEGMENTS()
```

### Example

```
=> SELECT ENABLE_LOCAL_SEGMENTS();
   ENABLE_LOCAL_SEGMENTS
-----
   ENABLED
(1 row)
```

## SET\_SCALING\_FACTOR

Sets the scaling factor used to determine the size of the storage containers used when rebalancing the database and when using local data segmentation is enabled. See Cluster Scaling for details.

### Syntax

```
SET_SCALING_FACTOR(factor)
```

### Parameters

<i>factor</i>	An integer value between 1 and 32. Vertica uses this value to calculate the number of storage containers each projection is broken into when rebalancing or when local data segmentation is enabled.
---------------	--

**Note:** Setting this value too high can cause nodes to create too many small container files, greatly reducing efficiency and potentially causing a Too Many ROS containers error. In general, you should only set this value manually to enable data scaling on a database where it was previously disabled, or if you have found the amount of data skew (uneven distribution of data between the nodes) in your database to be unacceptable. You may find this to be the case if you have expanded your cluster from a power of two number of nodes to a non-power of two number of nodes.

### Example

```
=> SELECT SET_SCALING_FACTOR(12);
   SET_SCALING_FACTOR
-----
   SET
(1 row)
```

## Constraint Management Functions

This section contains constraint management functions specific to Vertica.

See also SQL system table *V\_CATALOG.TABLE\_CONSTRAINTS* (page 818)

## ANALYZE\_CONSTRAINTS

Analyzes and reports on constraint violations within the current schema search path.

You can check for constraint violations by passing an empty argument (which returns violations on all tables within the current schema), by passing a single table argument, or by passing two arguments containing a table name and a column or list of columns.

### Syntax

```
ANALYZE_CONSTRAINTS [ ( ' ' )
... | ( schema.table )
... | [ ( schema.table , column ) ]
```

### Parameters

<code>('')</code>	Analyzes and reports on all tables within the current schema search path.
<code>table</code>	Analyzes and reports on all constraints referring to the specified table.
<code>column</code>	Analyzes and reports on all constraints referring to specified table that contains the specified columns.

### Notes

- `ANALYZE_CONSTRAINTS()`, takes locks in the same way that `SELECT * FROM t1` holds a lock on table `t1`. See **LOCKS** (page 858) for additional information.
- Use **COPY** (page 607) with `NO COMMIT` keywords to incorporate detection of constraint violations into the load process. Vertica checks for constraint violations when queries are run, not when data is loaded. To avoid constraint violations, load data without committing it and then perform a post-load check of your data using the `ANALYZE_CONSTRAINTS` function. If the function finds constraint violations, you can roll back the load because you have not committed it.
- `ANALYZE_CONSTRAINTS()` fails if the database cannot perform constraint checks, such as when the system is out of resources. Vertica returns an error that identifies the specific condition that caused the failure.
- When `ANALYZE_CONSTRAINTS` finds violations, such as when you insert a duplicate value into a primary key, you can correct errors using the following functions. Effects last until the end of the session only:
  - `SELECT DISABLE_DUPLICATE_KEY_ERROR` (page 403)
  - `SELECT REENABLE_DUPLICATE_KEY_ERROR` (page 446)
- If you specify the wrong table, the system returns an error message:
 

```
SELECT ANALYZE_CONSTRAINTS('abc');
ERROR: 'abc' is not a table in the current search path
```
- If you issue the function using incorrect syntax, the system returns an error message with a hint:
 

```
ANALYZE ALL CONSTRAINT;
Or
ANALYZE CONSTRAINT abc;
ERROR: ANALYZE CONSTRAINT is not supported.
```

HINT: You may consider using `analyze_constraints()`.

- **ANALYZE\_CONSTRAINTS** returns an error if run from a non-default locale; for example:

```
=> \locale LEN
INFO: Canonical locale: 'en'
INFO: English
INFO: Standard collation: 'LEN'
=> SELECT ANALYZE_CONSTRAINTS('t1');
ERROR: ANALYZE_CONSTRAINTS is currently not supported in non-default
       locales
HINT: Set the locale in this session to en_US@collation=binary using
       the
       command "\locale en_US@collation=binary"
```

### Return Values

**ANALYZE\_CONSTRAINTS** returns results in a structured set (see table below) that lists the schema name, table name, column name, constraint name, constraint type, and the column values that caused the violation.

If the result set is empty, then no constraint violations exist; for example:

```
SELECT ANALYZE_CONSTRAINTS ('public.product_dimension', 'product_key');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
(0 rows)
```

The following result set, on the other hand, shows a primary key violation, along with the value that caused the violation ('10'):

```
SELECT ANALYZE_CONSTRAINTS ('');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
store       | t1         | c1           | pk_t1           | PRIMARY         | ('10')
(1 row)
```

The result set columns are described in further detail in the following table:

Column Name	Data Type	Description
Schema Name	VARCHAR	The name of the schema.
Table Name	VARCHAR	The name of the table, if specified.
Column Names	VARCHAR	Names of columns containing constraints. Multiple columns are in a comma-separated list: <code>store_key,</code> <code>store_key, date_key,</code>
Constraint Name	VARCHAR	The given name of the primary key, foreign key, unique, or not null constraint, if specified.
Constraint Type	VARCHAR	Identified by one of the following strings: 'PRIMARY KEY', 'FOREIGN KEY', 'UNIQUE', or 'NOT NULL'.
Column Values	VARCHAR	Value of the constraint column, in the same order in which <code>Column Names</code> contains the value of that column in the violating row.

		When interpreted as SQL, the value of this column forms a list of values of the same type as the columns in Column Names; for example: ( '1' ), ( '1', 'z' )
--	--	--

## Examples

Given the following inputs, Vertica returns one row, indicating one violation, because the same primary key value (10) was inserted into table t1 twice:

```
CREATE TABLE t1(c1 INT);
ALTER TABLE t1 ADD CONSTRAINT pk_t1 PRIMARY KEY (c1);
CREATE PROJECTION t1_p (c1) AS SELECT * FROM t1 UNSEGMENTED ALL NODES;
INSERT INTO t1 values (10);
INSERT INTO t1 values (10); --Duplicate primary key value
SELECT ANALYZE_CONSTRAINTS('t1');
```

Schema Name	Table Name	Column Names	Constraint Name	Constraint Type	Column Values
public	t1	c1	pk_t1	PRIMARY	('10')

(1 row)

If the second INSERT statement above had contained any different value, the result would have been 0 rows (no violations).

In this example, create a table that contains 3 integer columns, one a unique key and one a primary key:

```
CREATE TABLE fact_1(
  f INTEGER,
  f_UK INTEGER UNIQUE,
  f_PK INTEGER PRIMARY KEY
);
```

Try issuing a command that refers to a nonexistent column:

```
SELECT ANALYZE_CONSTRAINTS('f_BB', 'f2');
ERROR: 'f_BB' is not a table name in the current search path
```

Insert some values into table fact\_1 and commit the changes:

```
INSERT INTO fact_1 values (1, 1, 1);
COMMIT;
```

Now issue the ANALYZE\_CONSTRAINTS command on table fact\_1. No constraint violations are expected and none are found:

```
SELECT ANALYZE_CONSTRAINTS('fact_1');
```

Schema Name	Table Name	Column Names	Constraint Name	Constraint Type	Column Values
-------------	------------	--------------	-----------------	-----------------	---------------

(0 rows)

Now insert duplicate unique and primary key values and run ANALYZE\_CONSTRAINTS on table fact\_1 again. The system shows two violations: one against the primary key and one against the unique key:

```
INSERT INTO fact_1 VALUES (1, 1, 1);
COMMIT;
SELECT ANALYZE_CONSTRAINTS('fact_1');
```

Schema Name	Table Name	Column Names	Constraint Name	Constraint Type	Column Values
public	fact_1	f_pk	-	PRIMARY	('1')
public	fact_1	f_uk	-	UNIQUE	('1')

(2 rows)

The following command looks for constraint validation on only the unique key in table `fact_1`:

```
SELECT ANALYZE_CONSTRAINTS('fact_1', 'f_UK');
Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
public      | fact_1     | f_uk         | C_UNIQUE        | UNIQUE         | ('1')
(1 row)
```

The following example shows that you can specify the same column more than once; the function, however, returns the violation once only:

```
SELECT ANALYZE_CONSTRAINTS('fact_1', 'f_PK, F_PK');
Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
public      | fact_1     | f_pk         | C_PRIMARY       | PRIMARY        | ('1')
(1 row)
```

The following example creates a new dimension table, `dim_1`, and inserts a foreign key and different (character) data types:

```
CREATE TABLE dim_1 (b VARCHAR(3), b_PK VARCHAR(4), b_FK INTEGER REFERENCES fact_1(f_PK));
```

Alter the table to create a multicolumn unique key and multicolumn foreign key and create superprojections:

```
ALTER TABLE dim_1 ADD CONSTRAINT dim_1_multiuk PRIMARY KEY (b, b_PK);
```

The following command inserts a missing foreign key (0) in table `dim_1` and commits the changes:

```
INSERT INTO dim_1 VALUES ('r1', 'Xpk1', 0);
COMMIT;
```

Checking for constraints on table `dim_1` detects a foreign key violation:

```
SELECT ANALYZE_CONSTRAINTS('dim_1');
Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
public      | dim_1     | b_fk         | C_FOREIGN       | FOREIGN         | ('0')
(1 row)
```

Now add a duplicate value into the unique key and commit the changes:

```
INSERT INTO dim_1 values ('r2', 'Xpk1', 1);
INSERT INTO dim_1 values ('r1', 'Xpk1', 1);
COMMIT;
```

Checking for constraint violations on table `dim_1` detects the duplicate unique key error:

```
SELECT ANALYZE_CONSTRAINTS('dim_1');
Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
public      | dim_1     | b, b_pk     | dim_1_multiuk  | PRIMARY        | ('r1', 'Xpk1')
public      | dim_1     | b_fk        | C_FOREIGN       | FOREIGN         | ('0')
(2 rows)
```

Now create a table with multicolumn foreign key and create the superprojections:

```
CREATE TABLE dim_2(z_fk1 VARCHAR(3), z_fk2 VARCHAR(4));
ALTER TABLE dim_2 ADD CONSTRAINT dim_2_multiuk FOREIGN KEY (z_fk1, z_fk2) REFERENCES dim_1(b, b_PK);
```

Now insert a foreign key that matches a foreign key in table `dim_1` and commit the changes:

```
INSERT INTO dim_2 VALUES ('r1', 'Xpk1');
COMMIT;
```

Checking for constraints on table `dim_2` detects no violations:

```
SELECT ANALYZE_CONSTRAINTS('dim_2');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
(0 rows)
```

Add a value that does not match and commit the change:

```
INSERT INTO dim_2 values ('r1', 'NONE');
COMMIT;
```

Checking for constraints on table `dim_2` detects a foreign key violation:

```
SELECT ANALYZE_CONSTRAINTS('dim_2');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | dim_2      | z_fk1, z_fk2 | dim_2_multifk  | FOREIGN        | ('r1', 'NONE')
(1 row)
```

Now analyze all constraints on all tables:

```
SELECT ANALYZE_CONSTRAINTS('');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | dim_1      | b, b_pk      | dim_1_multiuk  | PRIMARY        | ('r1', 'Xpk1')
 public      | dim_1      | b_fk         | C_FOREIGN      | FOREIGN        | ('0')
 public      | dim_2      | z_fk1, z_fk2 | dim_2_multifk  | FOREIGN        | ('r1', 'NONE')
 public      | fact_1     | f_pk         | C_PRIMARY      | PRIMARY        | ('1')
 public      | fact_1     | f_uk         | C_UNIQUE       | UNIQUE         | ('1')
(5 rows)
```

To quickly clean up your database, issue the following command:

```
DROP TABLE fact_1 cascade;
DROP TABLE dim_1 cascade;
DROP TABLE dim_2 cascade;
```

To learn how to remove violating rows, see the ***DISABLE\_DUPLICATE\_KEY\_ERROR*** (page 403) function.

## See Also

Adding Constraints in the Administrator's Guide

***COPY*** (page 607)

***ALTER TABLE*** (page 589)

***CREATE TABLE*** (page 664)

## DISABLE\_DUPLICATE\_KEY\_ERROR

Disables error messaging when Vertica finds duplicate PRIMARY KEY/UNIQUE KEY values at run time. Queries execute as though no constraints are defined on the schema. Effects are session scoped.

**CAUTION:** When called, `DISABLE_DUPLICATE_KEY_ERROR()` suppresses data integrity checking and can lead to incorrect query results. Use this function only after you insert duplicate primary keys into a dimension table in the presence of a prejoin projection. Then correct the violations and turn integrity checking back on with ***REENABLE\_DUPLICATE\_KEY\_ERROR*** (page 446)().

## Syntax

```
DISABLE_DUPLICATE_KEY_ERROR();
```

## Notes

The following series of commands create a table named `dim` and the corresponding projection:

```
CREATE TABLE dim (pk INTEGER PRIMARY KEY, x INTEGER);
CREATE PROJECTION dim_p (pk, x) AS SELECT * FROM dim ORDER BY x UNSEGMENTED ALL
NODES;
```

The next two statements create a table named `fact` and the pre-join projection that joins `fact` to `dim`.

```
CREATE TABLE fact(fk INTEGER REFERENCES dim(pk));
CREATE PROJECTION prejoin_p (fk, pk, x) AS SELECT * FROM fact, dim WHERE pk=fk ORDER
BY x;
```

The following statements load values into table `dim`. Notice the last statement inserts a duplicate primary key value of 1:

```
INSERT INTO dim values (1,1);
INSERT INTO dim values (2,2);
INSERT INTO dim values (1,2); --Constraint violation
COMMIT;
```

Table `dim` now contains duplicate primary key values, but you cannot delete the violating row because of the presence of the pre-join projection. Any attempt to delete the record results in the following error message:

```
ROLLBACK: Duplicate primary key detected in FK-PK join Hash-Join (x dim_p), value
1
```

In order to remove the constraint violation (`pk=1`), use the following sequence of commands, which puts the database back into the state just before the duplicate primary key was added.

To remove the violation:

- 1 First save the original `dim` rows that match the duplicated primary key.

```
CREATE TEMP TABLE dim_temp(pk integer, x integer);
INSERT INTO dim_temp SELECT * FROM dim WHERE pk=1 AND x=1; -- original
dim row
```

- 2 Temporarily disable error messaging on duplicate constraint values:

```
SELECT DISABLE_DUPLICATE_KEY_ERROR();
```

**Caution:** Remember that issuing this command suppresses the enforcement of data integrity checking.

- 3 Remove the the original row that contains duplicate values:

```
DELETE FROM dim WHERE pk=1;
```

- 4 Allow the database to resume data integrity checking:

```
SELECT REENABLE_DUPLICATE_KEY_ERROR();
```

- 5 Reinsert the original values back into the dimension table:

```
INSERT INTO dim SELECT * from dim_temp;
```

```
COMMIT;
```

## 6 Validate your dimension and fact tables.

If you receive the following error message, it means that the duplicate records you want to delete are not identical. That is, the records contain values that differ in at least one column that is not a primary key; for example, (1,1) and (1,2).

```
ROLLBACK: Delete: could not find a data row to delete (data integrity violation?)
```

The difference between this message and the rollback message in the previous example is that a fact row contains a foreign key that matches the duplicated primary key, which has been inserted. Thus, a row with values from the fact and dimension table is now in the pre-join projection. In order for the DELETE statement (Step 3 in the following example) to complete successfully, extra predicates are required to identify the original dimension table values (the values that are in the pre-join).

This example is nearly identical to the previous example, except that an additional INSERT statement joins the fact table to the dimension table by a primary key value of 1:

```
INSERT INTO dim values (1,1);
INSERT INTO dim values (2,2);
INSERT INTO fact values (1); -- New insert statement joins fact with dim on
primary key value=1
INSERT INTO dim values (1,2); -- Duplicate primary key value=1
COMMIT;
```

To remove the violation:

### 1 First save the original dim and fact rows that match the duplicated primary key:

```
CREATE TEMP TABLE dim_temp(pk integer, x integer);
CREATE TEMP TABLE fact_temp(fk integer);
INSERT INTO dim_temp SELECT * FROM dim WHERE pk=1 AND x=1; -- original
dim row
INSERT INTO fact_temp SELECT * FROM fact WHERE fk=1;
```

### 2 Temporarily suppresses the enforcement of data integrity checking:

```
SELECT DISABLE_DUPLICATE_KEY_ERROR();
```

### 3 Remove the duplicate primary keys. These steps implicitly remove all fact rows with the matching foreign key, as well.

#### a) Remove the the original row that contains duplicate values:

```
DELETE FROM dim WHERE pk=1 AND x=1;
```

Note: The extra predicate (x=1) specifies removal of the original (1, 1) row, rather than the newly inserted (1, 2) values that caused the violation.

#### b) Remove all remaining rows:

```
DELETE FROM dim WHERE pk=1;
```

### 4 Turn on integrity checking:

```
SELECT REENABLE_DUPLICATE_KEY_ERROR();
```

### 5 Reinsert the original values back into the fact and dimension table:

```
INSERT INTO dim SELECT * from dim_temp;
INSERT INTO fact SELECT * from fact_temp;
COMMIT;
```

6 Validate your dimension and fact tables.

### See Also

**ANALYZE\_CONSTRAINTS** (page 377)

**REENABLE\_DUPLICATE\_KEY\_ERROR** (page 446)

### LAST\_INSERT\_ID

Returns the last value of a column whose value is automatically incremented through the `AUTO_INCREMENT` or `IDENTITY` **column-constraint** (page 674). If multiple sessions concurrently load the same table, the returned value is the last value generated for an `AUTO_INCREMENT` column by an insert in that session.

### Behavior Type

Volatile

### Syntax

```
LAST_INSERT_ID()
```

### Notes

- This function works only with `AUTO_INCREMENT` and `IDENTITY` columns. See **column-constraints** (page 674) for the **CREATE TABLE** (page 664) statement.
- `LAST_INSERT_ID` does not work with sequence generators created through the **CREATE SEQUENCE** (page 658) statement.

### Examples

Create a sample table called `customer4`.

```
=> CREATE TABLE customer4(  
    ID IDENTITY(2,2),  
    lname VARCHAR(25),  
    fname VARCHAR(25),  
    membership_card INTEGER  
);  
=> INSERT INTO customer4(lname, fname, membership_card)  
VALUES ('Gupta', 'Saleem', 475987);
```

Notice that the `IDENTITY` column has a seed of 2, which specifies the value for the first row loaded into the table, and an increment of 2, which specifies the value that is added to the `IDENTITY` value of the previous row.

Query the table you just created:

```
=> SELECT * FROM customer4;  
ID | lname | fname | membership_card  
-----+-----+-----+-----  
 2 | Gupta | Saleem |          475987  
(1 row)
```

Insert some additional values:

```
=> INSERT INTO customer4(lname, fname, membership_card)
VALUES ('Lee', 'Chen', 598742);
```

Call the `LAST_INSERT_ID` function:

```
=> SELECT LAST_INSERT_ID();
last_insert_id
```

```
-----
                4
(1 row)
```

Query the table again:

```
=> SELECT * FROM customer4;
ID | lname | fname | membership_card
-----+-----+-----+-----
  2 | Gupta | Saleem |          475987
  4 | Lee   | Chen   |          598742
(2 rows)
```

Add another row:

```
=> INSERT INTO customer4(lname, fname, membership_card)
VALUES ('Davis', 'Bill', 469543);
```

Call the `LAST_INSERT_ID` function:

```
=> SELECT LAST_INSERT_ID();
LAST_INSERT_ID
```

```
-----
                6
(1 row)
```

Query the table again:

```
=> SELECT * FROM customer4;
ID | lname | fname | membership_card
-----+-----+-----+-----
  2 | Gupta | Saleem |          475987
  4 | Lee   | Chen   |          598742
  6 | Davis | Bill   |          469543
(3 rows)
```

## See Also

***ALTER SEQUENCE*** (page 586)

***CREATE SEQUENCE*** (page 658)

***DROP SEQUENCE*** (page 708)

***V\_CATALOG.SEQUENCES*** (page 815)

Using Sequences and Sequence Privileges in the Administrator's Guide

## REENABLE\_DUPLICATE\_KEY\_ERROR

Restores the default behavior of error reporting by reversing the effects of `DISABLE_DUPLICATE_KEY_ERROR`. Effects are session scoped.

## Syntax

```
REENABLE_DUPLICATE_KEY_ERROR();
```

## Examples

For examples and usage see **DISABLE\_DUPLICATE\_KEY\_ERROR** (page 403).

## See Also

**ANALYZE\_CONSTRAINTS** (page 377)

## Data Collector Functions

The Vertica data collector collects and retains monitoring information. It works in conjunction with the Workload Analyzer, an advisor tool that intelligently monitors the performance of SQL queries and workloads and recommends tuning actions based on observations of the actual workload history.

## See Also

**V\_MONITOR.DATA\_COLLECTOR** (page 837)

Retaining Monitoring Information in the Troubleshooting Guide.

## GET\_DATA\_COLLECTOR\_POLICY

Retrieves a brief statement about the retention policy for the specified component.

## Syntax

```
GET_DATA_COLLECTOR_POLICY( 'component' )
```

## Parameters

<i>component</i>	Returns the retention policy for the specified component.
------------------	---

## Notes

For a current list of component names, query the **V\_MONITOR.DATA\_COLLECTOR** (page 837) system table. For example:

```
=> SELECT DISTINCT component, description FROM data_collector ORDER BY 1 ASC;
```

## Example

The following example returns the history of all resource acquisitions (the ResourceAcquisitions component):

```
=> SELECT get_data_collector_policy('ResourceAcquisitions');
       get_data_collector_policy
```

```
-----
1000KB kept in memory, 10000KB kept on disk.
(1 row)
```

**See Also****V\_MONITOR.DATA\_COLLECTOR** (page 837)

Retaining Monitoring Information in the Troubleshooting Guide

**SET\_DATA\_COLLECTOR\_POLICY**

Sets the retention policy for the specified component on all nodes. Failed nodes receive the setting when they rejoin the cluster.

**Syntax**

```
SET_DATA_COLLECTOR_POLICY('component', 'memoryKB', 'diskKB' )
```

**Parameters**

<i>component</i>	Returns the retention policy for the specified component.
<i>memoryKB</i>	Specifies the memory size to retain in kilobytes.
<i>diskKB</i>	Specifies the disk size in kilobytes.

**Notes**

- Only the superuser can configure the Data Collector.
- Before you change a retention policy, you can see what the current setting is by calling the `GET_DATA_COLLECTOR_POLICY()` function.
- For a list of all current component names, query the **V\_MONITOR.DATA\_COLLECTOR** (page 837) system table; for example:

```
=> SELECT DISTINCT component, description FROM data_collector
      ORDER BY 1 ASC;
```

**Example**

The following example returns the retention policy for the ResourceAcquisitions component:

```
=> SELECT get_data_collector_policy('ResourceAcquisitions');
       get_data_collector_policy
```

```
-----
1000KB kept in memory, 10000KB kept on disk.
(1 row)
```

The next command changes the memory and disk setting for ResourceAcquisitions to 1500KB and 25000KB, respectively:

```
=> SELECT set_data_collector_policy('ResourceAcquisitions', '1500', '25000');
       set_data_collector_policy
```

```
-----
SET
(1 row)
```

To verify the setting you just made, call the `GET_DATA_COLLECTOR_POLICY()` function on the specified component:

```
=> SELECT get_data_collector_policy('ResourceAcquisitions');
       get_data_collector_policy
```

```
-----  
1500KB kept in memory, 25000KB kept on disk.  
(1 row)
```

**See Also**

***V\_MONITOR.DATA\_COLLECTOR*** (page 837)

Retaining Monitoring Information in the Troubleshooting Guide

**Database Management Functions**

This section contains the database management functions specific to Vertica.

**CLEAR\_QUERY\_REPOSITORY**

Triggers Vertica to clear query data from the query repository immediately.

**Syntax**

```
CLEAR_QUERY_REPOSITORY()
```

**Notes**

Before using this function:

- 1 Note the value of the QueryRepoRetentionTime parameter.
- 2 Set the QueryRepoRetentionTime parameter to zero (0). (See Configuring Query Repository in the Troubleshooting Guide.)

```
=> SELECT SET_CONFIG_PARAMETER('QueryRepoRetentionTime','0');
```

Once you have cleared the query repository, set the QueryRepoRetentionTime parameter back to the original value (before you changed it to zero). The default value is 100.

**Example**

```
SELECT CLEAR_QUERY_REPOSITORY();  
CLEAR_QUERY_REPOSITORY
```

```
-----  
Query Repository Cleaned  
(1 row)
```

**See Also**

***V\_MONITOR.QUERY\_METRICS*** (page 868)

***V\_MONITOR.QUERY\_PROFILES*** (page 870)

Collecting Query Information in the Troubleshooting Guide

Configuration Parameters in the Administrator's Guide

## CLEAR\_RESOURCE\_REJECTIONS

Clears the content of the **RESOURCE\_REJECTIONS** (page 882) and **DISK\_RESOURCE\_REJECTIONS** (page 843) system tables. Normally, these tables are only cleared during a node restart. This function lets you clear the tables whenever you need. For example, you may want to clear the tables after having resolved a disk space issue that caused disk resource rejections.

## DATABASE\_SNAPSHOT

Creates a snapshot of the database's data. The snapshot is stored on the nodes, in subdirectories under the database's catalog directory. See Creating Database Snapshots in the Administrator's Guide for more information.

Note: The DATABASE\_SNAPSHOT function commits any pre-existing transaction and starts a new one.

### Syntax

```
DATABASE_SNAPSHOT( name, durable )
```

### Parameters

<i>name</i>	The name to use for the snapshot. This controls the name of the directory where the snapshots are stored.
<i>durable</i>	A Boolean value indicating whether the snapshot is durable or not. A durable snapshot is permanent, where a non-durable snapshot is only valid for as long the duration of the current transaction.

## DUMP\_LOCKTABLE

Returns information about deadlocked clients and the resources they are waiting for.

### Syntax

```
DUMP_LOCKTABLE ( )
```

### Notes

Use DUMP\_LOCKTABLE if Vertica becomes unresponsive:

- 1 Open an additional vsql connection.
  - 2 Execute the query:  

```
SELECT DUMP_LOCKTABLE ( ) ;
```

The output is written to vsql. See [Monitoring the Log Files](#).
  - 3 Copy the output and send it to **Technical Support** (on page 1).
- You can also see who is connected using the following command:

```
SELECT * FROM SESSIONS;
```

Close all sessions using the following command:

```
SELECT CLOSE_ALL_SESSIONS ( ) ;
```

Close a single session using the following command:

How to close a single session:

```
SELECT CLOSE_SESSION('session_id');
```

You get the session\_id value from the **V\_MONITOR.SESSIONS** (page 888) system table.

### See Also

**CLOSE\_ALL\_SESSIONS** (page 399)

**CLOSE\_SESSION** (page 396)

**V\_MONITOR.LOCKS** (page 858)

**V\_MONITOR.SESSIONS** (page 888)

## DUMP\_PARTITION\_KEYS

Dumps the partition keys of all projections in the system.

### Syntax

```
DUMP_PARTITION_KEYS ( )
```

### Example

```
SELECT DUMP_PARTITION_KEYS ( ) ;
```

### See Also

**DO\_TM\_TASK** (page 406)

**DROP\_PARTITION** (page 408)

**DUMP\_PROJECTION\_PARTITION\_KEYS** (page 415)

**DUMP\_TABLE\_PARTITION\_KEYS** (page 416)

**PARTITIONS** (page 862) system table

**PARTITION\_PROJECTION** (page 440)

**PARTITION\_TABLE** (page 441)

**V\_MONITOR.PARTITIONS** (page 862)

Partitioning Tables in the Administrator's Guide

## EXPORT\_TABLES

Generates a SQL script that can be used to recreate a logical schema (schemas, tables, constraints, and views) on a different cluster.

### Syntax

```
EXPORT_TABLES ( [ 'destination' ] , [ 'scope' ] )
```

### Parameters

<i>destination</i>	Specifies the path and name of the SQL output file. An empty string ( ' ' ), which is the default, dumps the script to standard output. A user who is not a DBA can only specify an empty string.
<i>scope</i>	<p>Determines the tables to be exported where scope is one of the following:</p> <ul style="list-style-type: none"> <li>▪ an empty string ( ' ' )—exports all non-virtual objects to which the user has access, including constraints. (Note that constraints are not objects which can be passed as individual arguments.) This is the default if no scope is specified.</li> <li>▪ a comma-delimited list of items in which each item can be one of the following: <ul style="list-style-type: none"> <li>▪ —'&lt;schema&gt;.&lt;obj&gt;'—matches the named object. The named object can be a table or view.</li> <li>▪ —'&lt;obj&gt;'—matches the named object within the current search path. The named object can be a schema, table, or view. If the named object is a schema, Vertica exports all non-virtual objects to which the user has access within that schema. If a schema and table both have the same name, the schema takes precedence.</li> </ul> </li> </ul> <p>EXPORT_TABLES returns an error if:</p> <ul style="list-style-type: none"> <li>▪ an explicitly-specified object does not exist.</li> <li>▪ The user has no access to the specified object.</li> </ul>

## Notes

- The script generated by this function:
  - Creates only the non-virtual objects for which the user has access.
  - Exports catalog objects in their Oid order.
- If projections are specified in the scope parameter, they are ignored.
- None of the parameters for EXPORT\_TABLES accepts a NULL value as input.

## Example

The following example exports the store.store\_orders\_fact table to standard output:

```
=> SELECT EXPORT_TABLES(' ', 'store.store_orders_fact');
```

## REMOVE\_DATABASE\_SNAPSHOT

Removes a snapshot from the catalog directory.

### Syntax

```
REMOVE_DATABASE_SNAPSHOT( name )
```

### Parameters

<i>name</i>	The name of the snapshot to remove
-------------	------------------------------------

### See Also

**DATABASE\_SNAPSHOT** (page 493)

**DATABASE\_SNAPSHOTS** (page 842) system table

Creating Database Snapshots in the Administrator's Guide

## SAVE\_QUERY\_REPOSITORY

Triggers Vertica to save query data to the query repository immediately.

### Syntax

```
SAVE_QUERY_REPOSITORY()
```

### Notes

- Vertica saves data based on the established query repository configuration parameters. For example, it will use the value of the QueryRepoRetentionTime parameter to determine the maximum number of days worth of queries to save. (See Configuring Query Repository in the Troubleshooting Guide.)
- Before you can save a query repository, you have to enable it:

```
SELECT SET_CONFIG_PARAMETER('QueryRepositoryEnabled', 1);
```

### Example

```
=> SELECT SAVE_QUERY_REPOSITORY();
```

```
SAVE_QUERY_REPOSITORY
```

```
-----
Query Repository Saved
(1 row)
```

### See Also

***V\_MONITOR.QUERY\_METRICS*** (page 868)

***V\_MONITOR.QUERY\_PROFILES*** (page 870)

Collecting Query Information in the Troubleshooting Guide

## SET\_CONFIG\_PARAMETER

Use SET\_CONFIG\_PARAMETER to set a configuration parameter.

**Note:** Vertica is designed to operate with minimal configuration changes. Use this function sparingly and carefully follow any documented guidelines for that parameter.

### Syntax

```
SELECT SET_CONFIG_PARAMETER('parameter', value)
```

### Parameters

<i>parameter</i>	Specifies the name of the parameter value being set
<i>value</i>	<ul style="list-style-type: none"> <li>Specifies the value of the parameter. Data type is variable.</li> </ul>

### Notes

- The syntax of value will vary depending upon the parameter and its expected data type. For strings, it must be enclosed in single quotes, for integers, it is unquoted. See Configuration Parameters for a list of parameters, their function and examples of usage.
- Caution:** If a node is down when this function is issued, the changes will only be done on the UP nodes. You must re-issue the function after the node is recovered in order for the changes to take effect there. Alternatively, use the Administration Tools to copy the files - See Distributing Configuration Files to the New Host.

### See Also

***V\_MONITOR.CONFIGURATION\_PARAMETERS*** (page 832)

## SET\_LOGLEVEL

Use SET\_LOGLEVEL to set the logging level in the Vertica database log files.

### Syntax

```
SELECT SET_LOGLEVEL(n)
```

### Parameters

<i>n</i>	Logging Level	Description
0	DISABLE	No logging
1	CRITICAL	Errors requiring database recovery
2	WARNING	Errors indicating problems of lesser magnitude
3	INFO	Informational messages
4	DEBUG	Debugging messages
5	TRACE	Verbose debugging messages
6	TIMING	Verbose debugging messages

## SHUTDOWN

Forces a database to shut down, even if there are users connected.

### Syntax

```
SHUTDOWN ( [ 'false' | 'true' ] )
```

### Parameters

<i>false</i>	[Default] Returns a message if users are connected. Has the same effect as supplying no parameters.
<i>true</i>	Performs a moveout operation and forces the database to shut down, disallowing further connections.

### Notes

- Quotes around the `true` or `false` arguments are optional.
- Issuing the shutdown command without arguments or with the default (`false`) argument returns a message if users are connected, and the shutdown fails. If no users are connected, the database performs a moveout operation and shuts down.
- Issuing the `SHUTDOWN('true')` command forces the database to shut down whether users are connected or not.
- You can check the status of the shutdown operation in the `vertica.log` file:  

```
2010-03-09 16:51:52.625 unknown:0x7fc6d6d2e700 [Init] <INFO> Shutdown complete. Exiting.
```

- As an alternative to SHUTDOWN(), you can also temporarily set MaxClientSessions to 0 and then use CLOSE\_ALL\_SESSIONS(). New client connections cannot connect unless they connect using the dbadmin account. See **CLOSE\_ALL\_SESSIONS** (page 399) for details.

### Examples

The following command attempts to shut down the database. Because users are connected, the command fails:

```
=> SELECT SHUTDOWN('false');
NOTICE: Cannot shut down while users are connected
        SHUTDOWN
-----
Shutdown: aborting shutdown
(1 row)
```

**Note that SHUTDOWN() and SHUTDOWN('false') perform the same operation:**

```
=> SELECT SHUTDOWN();
NOTICE: Cannot shut down while users are connected
        SHUTDOWN
-----
Shutdown: aborting shutdown
(1 row)
```

Using the 'true' parameter forces the database to shut down, even though clients might be connected:

```
=> SELECT SHUTDOWN('true');
        SHUTDOWN
-----
Shutdown: moveout complete
(1 row)
```

### See Also

**SESSIONS** (page 888)

## Epoch Management Functions

This section contains the epoch management functions specific to Vertica.

## ADVANCE\_EPOCH

Manually closes the current epoch and begins a new epoch.

### Syntax

```
ADVANCE_EPOCH ( [ integer ] )
```

### Parameters

<i>integer</i>	Specifies the number of epochs to advance.
----------------	--

### Note

This function is primarily maintained for backward compatibility with earlier versions of Vertica.

### Example

The following command increments the epoch number by 1:

```
=> SELECT ADVANCE_EPOCH(1);
```

### See Also

**ALTER PROJECTION** (page 580)

## GET\_AHM\_EPOCH

Returns the number of the epoch in which the Ancient History Mark is located. Data deleted up to and including the AHM epoch can be purged from physical storage.

### Syntax

```
GET_AHM_EPOCH()
```

**Note:** The AHM epoch is 0 (zero) by default (purge is disabled).

### Examples

```
SELECT GET_AHM_EPOCH();
      get_ahm_epoch
-----
Current AHM epoch: 0
(1 row)
```

## GET\_AHM\_TIME

Returns a **TIMESTAMP** value representing the Ancient History Mark. Data deleted up to and including the AHM epoch can be purged from physical storage.

### Syntax

```
GET_AHM_TIME()
```

### Examples

```
SELECT GET_AHM_TIME();
      GET_AHM_TIME
```

```
-----
Current AHM Time: 2010-05-13 12:48:10.532332-04
(1 row)
```

**See Also**

**SET DATESTYLE** (page 767) for information about valid **TIMESTAMP** (page 90) values.

**GET\_CURRENT\_EPOCH**

Returns the number of the current epoch. The epoch into which data (COPY, INSERT, UPDATE, and DELETE operations) is currently being written. The current epoch advances automatically every three minutes.

**Syntax**

```
GET_CURRENT_EPOCH()
```

**Examples**

```
SELECT GET_CURRENT_EPOCH();
       GET_CURRENT_EPOCH
-----
                        683
(1 row)
```

**GET\_LAST\_GOOD\_EPOCH**

Returns the number of the last good epoch. A term used in manual recovery, LGE (Last Good Epoch) refers to the most recent epoch that can be recovered.

**Syntax**

```
GET_LAST_GOOD_EPOCH()
```

**Examples**

```
SELECT GET_LAST_GOOD_EPOCH();
       GET_LAST_GOOD_EPOCH
-----
                        682
(1 row)
```

**MAKE\_AHM\_NOW**

Sets the Ancient History Mark (AHM) to the greatest allowable value, and lets you drop any projections that existed before the issue occurred.

**Caution:** This function is intended for use by Administrators only.

**Syntax**

```
MAKE_AHM_NOW ( [ true ] )
```

**Parameters**

<code>true</code>	[Optional] Allows AHM to advance when nodes are down. <b>Note:</b> If the AHM is advanced after the last good epoch of the failed nodes, those nodes must recover all data from
-------------------	--

	scratch. Use with care.
--	-------------------------

## Notes

- The MAKE\_AHM\_NOW function performs the following operations:
  - Advances the epoch.
  - Performs a moveout operation on all projections.
  - Sets the AHM to LGE — at least to the current epoch at the time MAKE\_AHM\_NOW() was issued.
- All history is lost and you cannot perform historical queries prior to the current epoch.

## Example

```
=> SELECT MAKE_AHM_NOW();
       MAKE_AHM_NOW
-----
AHM set (New AHM Epoch: 683)
(1 row)
```

The following command allows the AHM to advance, even though node 2 is down:

```
=> SELECT MAKE_AHM_NOW(true);
WARNING: Received no response from v_vmartdb_node0002 in get cluster LGE
WARNING: Received no response from v_vmartdb_node0002 in get cluster LGE
WARNING: Received no response from v_vmartdb_node0002 in set AHM
       MAKE_AHM_NOW
-----
AHM set (New AHM Epoch: 684)
(1 row)
```

## See Also

**DROP PROJECTION** (page 704)

**MARK\_DESIGN\_KSAFE** (page 436)

**SET\_AHM\_EPOCH** (page 463)

**SET\_AHM\_TIME** (page 464)

## SET\_AHM\_EPOCH

Sets the Ancient History Mark (AHM) to the specified epoch. This function allows deleted data up to and including the AHM epoch to be purged from physical storage.

SET\_AHM\_EPOCH is normally used for testing purposes. Consider **SET\_AHM\_TIME** (page 464) instead, which is easier to use.

## Syntax

```
SET_AHM_EPOCH ( epoch, [ true ] )
```

## Parameters

<i>epoch</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>▪ The number of the epoch in which to set the AHM</li> <li>▪ Zero (0) (the default) disables <b>purge</b> (page 444)</li> </ul>
<i>true</i>	Optionally allows the AHM to advance when nodes are down. <b>Note:</b> If the AHM is advanced after the last good epoch of the failed nodes, those nodes must recover all data from scratch. Use with care.

## Notes

If you use SET\_AHM\_EPOCH , the number of the specified epoch must be:

- Greater than the current AHM epoch
- Less than the current epoch
- Less than or equal to the cluster last good epoch (the minimum of the last good epochs of the individual nodes in the cluster)
- Less than or equal to the cluster refresh epoch (the minimum of the refresh epochs of the individual nodes in the cluster)

Use the **SYSTEM** (page 898) table to see current values of various epochs related to the AHM; for example:

```
=> SELECT * from SYSTEM;
-[ RECORD 1 ]-----+-----
current_timestamp   | 2009-08-11 17:09:54.651413
current_epoch       | 1512
ahm_epoch           | 961
last_good_epoch     | 1510
refresh_epoch       | -1
designed_fault_tolerance | 1
node_count          | 4
node_down_count     | 0
current_fault_tolerance | 1
catalog_revision_number | 1590
wos_used_bytes      | 0
wos_row_count       | 0
ros_used_bytes      | 41490783
ros_row_count       | 1298104
total_used_bytes    | 41490783
total_row_count     | 1298104
```

All nodes must be up. You cannot use SET\_AHM\_EPOCH when any node in the cluster is down, except by using the optional *true* parameter.

When a node is down and you issue SELECT MAKE\_AHM\_NOW() , the following error is printed to the vertica.log:

```
Some nodes were excluded from setAHM. If their LGE is before the AHM they will perform full recovery.
```

## Examples

The following command sets the AHM to a specified epoch of 12:

```
=> SELECT SET_AHM_EPOCH(12);
```

The following command sets the AHM to a specified epoch of 2 and allows the AHM to advance despite a failed node:

```
=> SELECT SET_AHM_EPOCH(2, true);
```

**See Also**

**MAKE\_AHM\_NOW** (page 434)

**SET\_AHM\_TIME** (page 464)

**SYSTEM** (page 898)

**SET\_AHM\_TIME**

Sets the Ancient History Mark (AHM) to the epoch corresponding to the specified time on the initiator node. This function allows historical data up to and including the AHM epoch to be purged from physical storage.

**Syntax**

```
SET_AHM_TIME ( time , [ true ] )
```

**Parameters**

<i>time</i>	Is a <b>TIMESTAMP</b> (page 90) value that is automatically converted to the appropriate epoch number.
<i>true</i>	[Optional] Allows the AHM to advance when nodes are down. <b>Note:</b> If the AHM is advanced after the last good epoch of the failed nodes, those nodes must recover all data from scratch.

**Notes**

- SET\_AHM\_TIME returns a TIMESTAMP WITH TIME ZONE value representing the end point of the AHM epoch.
- You cannot change the AHM when any node in the cluster is down, except by using the optional *true* parameter.
- When a node is down and you issue SELECT MAKE\_AHM\_NOW(), the following error is printed to the vertica.log:  

```
Some nodes were excluded from setAHM. If their LGE is before the AHM they
will perform full recovery.
```

**Examples**

Epochs depend on a configured epoch advancement interval. If an epoch includes a three-minute range of time, the purge operation is accurate only to within minus three minutes of the specified timestamp:

```
=> SELECT SET_AHM_TIME('2008-02-27 18:13');
           set_ahm_time
-----
AHM set to '2008-02-27 18:11:50-05'
```

(1 row)

**Note:** The -05 part of the output string is a time zone value, an offset in hours from UTC (Universal Coordinated Time, traditionally known as Greenwich Mean Time, or GMT).

In the above example, the actual AHM epoch ends at 18:11:50, roughly one minute before the specified timestamp. This is because `SET_AHM_TIME` selects the epoch that ends at or before the specified timestamp. It does not select the epoch that ends after the specified timestamp because that would purge data deleted as much as three minutes after the AHM.

For example, using only hours and minutes, suppose that epoch 9000 runs from 08:50 to 11:50 and epoch 9001 runs from 11:50 to 15:50. `SET_AHM_TIME('11:51')` chooses epoch 9000 because it ends roughly one minute before the specified timestamp.

In the next example, if given an environment variable set as `date = `date``; the following command fails if a node is down:

```
=> SELECT SET_AHM_TIME('$date');
```

In order to force the AHM to advance, issue the following command instead:

```
=> SELECT SET_AHM_TIME('$date', true);
```

### See Also

**MAKE\_AHM\_NOW** (page 434)

**SET\_AHM\_EPOCH** (page 463) for a description of the range of valid epoch numbers.

**SET DATESTYLE** (page 767) for information about specifying a **TIMESTAMP** (page 90) value.

## License Management Functions

This section contains function that monitor Vertica license status and compliance.

### AUDIT

Estimates the raw data size of a database, a schema, a projection, or a table as it is counted in an audit of the database size. `AUDIT` estimates the size using the same data sampling method as the audit that Vertica performs to determine if a database is compliant with any database size allowances in its license. The results of this function are not considered when Vertica determines whether the size of the database complies with the Vertica license's data allowance. See *How Vertica Calculates Database Size in the Administrator's Guide* for details.

**Note:** This function can only audit the size of tables, schemas, and databases which the user has permission to access. If a non-superuser attempts to audit the entire database, the audit will only estimate the size of the data that the user is allowed to read.

### Syntax

```
AUDIT([name] [, granularity] [, error_tolerance [, confidence_level]])
```

### Parameters

<i>name</i>	A string containing the name of the schema, projection, or table to be audited. If this string is empty (""), the entire database is audited.
-------------	---

<i>granularity</i>	<p>A string containing the level at which the audit results are reported. The recognized levels are:</p> <ul style="list-style-type: none"> <li>▪ 'schema'</li> <li>▪ 'table'</li> <li>▪ 'projection'</li> </ul> <p>By default, the granularity is the same level as <i>name</i>. For example, if <i>name</i> is a schema, then the size of the entire schema is reported. If you instead specify 'table' as the granularity, AUDIT reports the size of each table in the schema. The granularity must be finer than that of object: specifying 'schema' for an audit of a table has no effect.</p> <p>The results of an audit with a granularity are reported in the V_CATALOG.USER_AUDITS system table.</p>
<i>error_tolerance</i>	<p>The percent margin of error allowed in the estimation, as a decimal number between 0 and 100. The default value is 5, for a 5% margin of error.</p> <p><b>Note:</b> The lower this value is, the more resources the audit uses since it will perform more data sampling. Setting this value to 0 results in a full audit of the database, which is very resource intensive, as all of the data in the database is analyzed. Doing a full audit of the database significantly impacts performance and is not recommended on a production database.</p>
<i>confidence_level</i>	<p>The statistical confidence level percentage of the estimate, as a decimal between 0 and 100. The default is 99, for a confidence level of 99%.</p> <p><b>Note:</b> The higher this value is, the more resources the function uses since it will perform more data sampling. Setting this value to 1 results in a full audit of the database, which is very resource intensive, as all of the database is analyzed. Doing a full audit of the database significantly impacts performance and is not recommended on a production database.</p>

### Notes

Due to the iterative sampling used in the auditing process, making the error tolerance a small fraction of a percent (0.00001, for example) can cause the AUDIT function to run for a longer period than a full database audit.

### Examples

To audit the entire database:

```
=> SELECT AUDIT('');
      AUDIT
-----
      76376696
(1 row)
```

To audit the database with a 25% error tolerance:

```
=> SELECT AUDIT('',25);
      AUDIT
-----
      75797126
(1 row)
```

To audit the database with a 25% level of tolerance and a 90% confidence level:

```
=> SELECT AUDIT('',25,90);
      AUDIT
-----
      76402672
(1 row)
```

To audit just the online\_sales schema in the VMart example database:

```
VMart=> SELECT AUDIT('online_sales');
      AUDIT
-----
      35716504
(1 row)
```

To audit the online\_sales schema and report the results by table:

```
=> SELECT AUDIT('online_sales','table');
      AUDIT
-----
      See table sizes in v_catalog.user_audits for schema online_sales
(1 row)
```

```
=> \x
Expanded display is on.
```

```
=> SELECT * FROM user_audits WHERE object_schema = 'online_sales';
-[ RECORD 1 ]-----+-----
size_bytes          | 64960
user_id             | 45035996273704962
user_name           | dbadmin
object_id           | 45035996273717636
object_type         | TABLE
object_schema       | online_sales
object_name         | online_page_dimension
audit_start_timestamp | 2011-04-05 09:24:48.224081-04
audit_end_timestamp  | 2011-04-05 09:24:48.337551-04
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling       | f
confidence_interval_lower_bound_bytes | 64960
confidence_interval_upper_bound_bytes | 64960
sample_count        | 0
```

```

cell_count | 0
-[ RECORD 2 ]-----+-----
size_bytes | 20197
user_id    | 45035996273704962
user_name  | dbadmin
object_id  | 45035996273717640
object_type | TABLE
object_schema | online_sales
object_name | call_center_dimension
audit_start_timestamp | 2011-04-05 09:24:48.340206-04
audit_end_timestamp   | 2011-04-05 09:24:48.365915-04
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling         | f
confidence_interval_lower_bound_bytes | 20197
confidence_interval_upper_bound_bytes | 20197
sample_count         | 0
cell_count          | 0
-[ RECORD 3 ]-----+-----
size_bytes          | 35614800
user_id            | 45035996273704962
user_name          | dbadmin
object_id          | 45035996273717644
object_type        | TABLE
object_schema      | online_sales
object_name        | online_sales_fact
audit_start_timestamp | 2011-04-05 09:24:48.368575-04
audit_end_timestamp   | 2011-04-05 09:24:48.379307-04
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling       | t
confidence_interval_lower_bound_bytes | 34692956
confidence_interval_upper_bound_bytes | 36536644
sample_count        | 10000
cell_count          | 9000000

```

## AUDIT\_LICENSE\_SIZE

Triggers an immediate audit of the database size to determine if it is in compliance the raw data storage allowance included in your Vertica license. The audit is performed in the background, so this function call returns immediately. To see the results of the audit when it is done, use the **GET\_COMPLIANCE\_STATUS** (page 425) function.

### Syntax

```
AUDIT_LICENSE_SIZE()
```

### Example

```
=> SELECT audit_license_size();
audit_license_size
-----
Service hurried
```

(1 row)

## AUDIT\_LICENSE\_TERM

Triggers an immediate audit to determine if the Vertica license has expired. The audit happens in the background, so this function returns immediately. To see the result of the audit, use the **GET\_COMPLIANCE\_STATUS** (page 425) function.

### Syntax

```
AUDIT_LICENSE_TERM()
```

### Example

```
=> SELECT AUDIT_LICENSE_TERM();
   AUDIT_LICENSE_TERM
-----
Service hurried
(1 row)
```

## GET\_AUDIT\_TIME

Reports the time when the automatic audit of database size occurs. Vertica performs this audit if your Vertica license includes a data size allowance. For details of this audit, see *Managing Your License Key* in the *Administrator's Guide*. To change the time the audit runs, use the **SET\_AUDIT\_TIME** (page 465) function.

### Syntax

```
GET_AUDIT_TIME()
```

### Example

```
=> SELECT get_audit_time();
           get_audit_time
-----
The audit is scheduled to run at 11:59 PM each day.
(1 row)
```

## GET\_COMPLIANCE\_STATUS

Displays whether your database is in compliance with your Vertica license agreement. This information includes the results of Vertica's most recent audit of the database size (if your license has a data allowance as part of its terms), and the license term (if your license has an end date).

The information displayed by **GET\_COMPLIANCE\_STATUS** includes:

- The estimated size of the database (see *How Vertica Calculates Database Size* in the *Administrator's Guide* for an explanation of the size estimate).
- The raw data size allowed by your Vertica license.
- The percentage of your allowance that your database is currently using.
- The date and time of the last audit.

- Whether your database complies with the data allowance terms of your license agreement.
- The end date of your license.
- How many days remain until your license expires.

**Note:** If your license does not have a data allowance or end date, some of the values may not appear in GET\_COMPLIANCE\_STATUS's output

If the audit shows your license is not in compliance with your data allowance, you should either delete data to bring the size of the database under the licensed amount, or upgrade your license. If your license term has expired, you should contact Vertica immediately to renew your license. See Managing Your License Key in the Administrator's Guide for further details.

### Syntax

```
GET_COMPLIANCE_STATUS()
```

### Example

```
-----
                                GET_COMPLIANCE_STATUS
-----
---
Raw Data Size: 0.071GB +/- 0.003GB
License Size : 500.000GB
Utilization  : 0%
Audit Time   : 2011-03-09 09:54:09.538704+00
Compliance Status : The database is in compliance with respect to raw data size.

License End Date: 04/06/2011
Days Remaining: 28.59
(1 row)
```

### DISPLAY\_LICENSE

Returns the terms of your Vertica license. This includes the start and end dates for which the license is valid (or "Perpetual" if the license has no expiration), and the amount of data your database can store.

### Syntax

```
DISPLAY_LICENSE()
```

### Examples

```
=> SELECT DISPLAY_LICENSE();
           DISPLAY_LICENSE
-----
Vertica Systems, Inc.
1/1/2011
12/31/2011
30
50TB

(1 row)
```

## SET\_AUDIT\_TIME

Sets the time that Vertica performs automatic database size audit to determine if the size of the database is compliant with the raw data allowance in your Vertica license. You may want to use this function if the audits are currently scheduled to occur during your database's peak activity time. This is normally not a concern, since the automatic audit has little impact on database performance.

---

**Note:** Audits are scheduled by the preceding audit, so changing the audit time does not affect the next scheduled audit. For example, if your next audit is scheduled to take place at 11:59PM and you use SET\_AUDIT\_TIME to change the audit schedule 3AM, the previously scheduled 11:59PM audit still runs. As that audit finishes, it schedules the next audit to occur at 3AM.

If you want to prevent the next scheduled audit from running at its scheduled time, you can change the scheduled time using SET\_AUDIT\_TIME then manually trigger an audit to run immediately using **AUDIT\_LICENSE\_SIZE** (page 394). As the manually-triggered audit finishes, it schedules the next audit to occur at the time you set using SET\_AUDIT\_TIME (effectively overriding the previously scheduled audit).

---

### Syntax

```
SET_AUDIT_TIME(time)
```

<i>time</i>	A string containing the time in 'HH:MM AM/PM' format (for example, '1:00 AM') when the audit should run daily.
-------------	--

### Example

```
=> SELECT SET_AUDIT_TIME('3:00 AM');
           SET_AUDIT_TIME
```

---

```
The scheduled audit time will be set to 3:00 AM after the next audit.
(1 row)
```

## Partition Management Functions

This section contains partition management functions specific to Vertica.

### DROP\_PARTITION

Forces the partition of projections (if needed) and then drops the specified partition.

#### Syntax

```
DROP_PARTITION [ ( table_name ) , ( partition_value ) ]
```

#### Parameters

<i>table-name</i>	Specifies the name of the table. Note: The specified <i>table_name</i> argument cannot be used as a dimension table in a pre-joined projection and cannot contain
-------------------	--

	projections that are not up to date (have not been refreshed)
<i>partition_value</i>	Must be specified as a string (within quotes) for all data types; for example: <code>DROP_PARTITION('trade', '2006');</code>

## Notes and Restrictions

In general, if a ROS container has data that belongs to  $n+1$  partitions and you want to drop a specific partition, the `DROP_PARTITION` operation:

- 1 Forces the partition of data into two containers where
  - one container holds the data that belongs to the partition that is to be dropped
  - another container holds the remaining  $n$  partitions
- 2 Drops the specified partition.

You can also use the ***MERGE\_PARTITIONS*** (page 439) function to merges ROS containers that have data belonging to partitions in a specified partition key range; for example, `[partitionKeyFrom, partitionKeyTo]`.

`DROP_PARTITION` forces a moveout if there is data in the WOS (WOS is not partition aware).

`DROP_PARTITION` acquires an exclusive lock on the table to prevent `DELETE | UPDATE | INSERT | COPY` statements from affecting the table, as well as any `SELECT` statements issued at `SERIALIZABLE` isolation level.

Users must be the table owner to drop a partition. They must have `MODIFY ( INSERT | UPDATE | DELETE )` permissions in order to:

- Partition a projection/table
- Merge partitions
- Run mergeout, moveout or purge operations on a projection

`DROP_PARTITION` operations cannot be performed on tables with projections that are not up to date (have not been refreshed).

## Examples

Using the example schema in Defining Partitions, the following command explicitly drops the 2006 partition key from table `trade`:

```
SELECT DROP_PARTITION('trade', 2006);
      DROP_PARTITION
-----
Partition dropped
(1 row)
```

Here, the partition key is specified:

```
SELECT DROP_PARTITION('trade', EXTRACT('year' FROM '2006-01-01'::date));
      DROP_PARTITION
-----
Partition dropped
(1 row)
```

The following example creates a table called `dates` and partitions the table by year:

```
CREATE TABLE dates (
    year INTEGER NOT NULL,
    month VARCHAR(8) NOT NULL)
PARTITION BY year * 12 + month;
```

The following statement drops the partition using a constant for Oct 2007 (2007\*12 + 10 = 24094):

```
SELECT DROP_PARTITION('dates', '24094');
DROP_PARTITION
```

```
-----
Partition dropped
(1 row)
```

Alternatively, the expression can be placed in line: `SELECT DROP_PARTITION('dates', 2007*12 + 10);`

### See Also

**ADVANCE EPOCH** (page 376)

**ALTER PROJECTION** (page 580)

**COLUMN\_STORAGE** (page 830)

**CREATE TABLE** (page 664)

**DO\_TM\_TASK** (page 406)

**DUMP\_PARTITION\_KEYS** (page 414)

**DUMP\_PROJECTION\_PARTITION\_KEYS** (page 415)

**DUMP\_TABLE\_PARTITION\_KEYS** (page 416)

**MERGE\_PARTITIONS** (page 439)

**PARTITION\_PROJECTION** (page 440)

**PARTITION\_TABLE** (page 441)

**PROJECTIONS** (page 811)

Dropping Partitions in the Administrator's Guide

### DUMP\_PROJECTION\_PARTITION\_KEYS

Dumps the partition keys of the specified projection.

#### Syntax

```
DUMP_PROJECTION_PARTITION_KEYS( 'projection_name' )
```

#### Parameters

<i>projection_name</i>	Specifies the name of the projection.
------------------------	---------------------------------------

#### Example

The following example creates a simple table called `states` and partitions the data by state:

```
CREATE TABLE states (
    year INTEGER NOT NULL,
    state VARCHAR NOT NULL)
PARTITION BY state;
CREATE PROJECTION states_p (state, year) AS SELECT * FROM states
ORDER BY state, year UNSEGMENTED ALL NODES;
```

Now drop the partition key of the specified projection:

```
SELECT DUMP_PROJECTION_PARTITION_KEYS( 'states_p_node0001' );
Partition keys on node helios_node0001
Projection 'states_p_node0001'
No of partition keys: 1
Partition keys on node helios_node0002
...
(1 row)
```

**See Also**

- DO\_TM\_TASK** (page 406)
- DROP\_PARTITION** (page 408)
- DUMP\_PARTITION\_KEYS** (page 414)
- DUMP\_TABLE\_PARTITION\_KEYS** (page 416)
- PARTITION\_PROJECTION** (page 440)
- PARTITION\_TABLE** (page 441)
- PROJECTIONS** (page 811) system table

Partitioning Tables in the Administrator's Guide

**DUMP\_TABLE\_PARTITION\_KEYS**

Dumps the partition keys of all projections anchored on the specified table.

**Syntax**

```
DUMP_TABLE_PARTITION_KEYS ( 'table_name' )
```

**Parameters**

<i>table_name</i>	Specifies the name of the table.
-------------------	----------------------------------

**Example**

The following example creates a simple table called `states` and partitions the data by state:

```
CREATE TABLE states (
    year INTEGER NOT NULL,
    state VARCHAR NOT NULL)
PARTITION BY state;
CREATE PROJECTION states_p (state, year) AS SELECT * FROM states
ORDER BY state, year UNSEGMENTED ALL NODES;
```

Now drop the partition keys of all projections anchored on table `states`:

```
SELECT DUMP_TABLE_PARTITION_KEYS( 'states' );
Partition keys on helios_node0001
  Projection 'states_p_node0004'
    No of partition keys: 1
  Projection 'states_p_node0003'
    No of partition keys: 1
  Projection 'states_p_node0002'
    No of partition keys: 1
  Projection 'states_p_node0001'
    No of partition keys: 1
Partition keys on helios_node0002
...
(1 row)
```

### See Also

***DO\_TM\_TASK*** (page 406)

***DROP\_PARTITION*** (page 408)

***DUMP\_PARTITION\_KEYS*** (page 415)

***DUMP\_PROJECTION\_PARTITION\_KEYS*** (page 416)

***PARTITION\_PROJECTION*** (page 440)

***PARTITION\_TABLE*** (page 441)

Partitioning Tables in the Administrator's Guide

## MERGE\_PARTITIONS

Merges ROS containers that have data belonging to partitions in a specified partition key range: [ `partitionKeyFrom`, `partitionKeyTo` ] .

### Syntax

```
MERGE_PARTITIONS [ ( table_name ) ,
... ( partition_key_from ) , ( partition_key_to ) ]
```

### Parameters

<i>table_name</i>	Specifies the name of the table
<i>partition_key_from</i>	Specifies the start point of the partition
<i>partition_key_to</i>	Specifies the end point of the partition

### Notes

- Partitioning functions take immutable functions only, in order that the same information be available across all nodes.

- The edge values are included in the range, and `partition_key_from` must be less than or equal to `partition_key_to`.
- Inclusion of partitions in the range is based on the application of less than(<)/greater than(>) operators of the corresponding data type.

**Note:** No restrictions are placed on a partition key's data type.

- If `partition_key_from` is the same as `partition_key_to`, all ROS containers of the partition key are merged into one ROS.

Users must be the table owner to drop a partition. They must have `MODIFY ( INSERT | UPDATE | DELETE )` permissions in order to:

- Partition a projection/table
- Merge partitions
- Run mergeout, moveout or purge operations on a projection

### Examples

```
=> SELECT MERGE_PARTITIONS('T1', '200', '400');
=> SELECT MERGE_PARTITIONS('T1', '800', '800');
=> SELECT MERGE_PARTITIONS('T1', 'CA', 'MA');
=> SELECT MERGE_PARTITIONS('T1', 'false', 'true');
=> SELECT MERGE_PARTITIONS('T1', '06/06/2008', '06/07/2008');
=> SELECT MERGE_PARTITIONS('T1', '02:01:10', '04:20:40');
=> SELECT MERGE_PARTITIONS('T1', '06/06/2008 02:01:10', '06/07/2008 02:01:10');
=> SELECT MERGE_PARTITIONS('T1', '8 hours', '1 day 4 hours 20 seconds');
```

## PARTITION\_PROJECTION

Forces a split of ROS containers of the specified projection.

### Syntax

```
PARTITION_PROJECTION ( projection_name )
```

### Parameters

<code><i>projection_name</i></code>	Specifies the name of the projection.
-------------------------------------	---------------------------------------

### Notes

Partitioning expressions take immutable functions only, in order that the same information be available across all nodes.

`PARTITION_PROJECTION()` is similar to ***PARTITION\_TABLE*** (page 441)(), except that `PARTITION_PROJECTION` works only on the specified projection, instead of the table.

Vertica internal operations (mergeout, refresh, and recovery) maintain partition separation except in certain cases:

- Recovery of a projection when the buddy projection from which the partition is recovering is identically sorted. If the projection is undergoing a full rebuild, it is recovered one ROS container at a time. The projection ends up with a storage layout identical to its buddy and is, therefore, properly segmented.  
**Note:** In the case of a partial rebuild, all recovered data goes into a single ROS container and must be partitioned manually.
- Manual tuple mover operations often output a single storage container, combining any existing partitions; for example, after executing any of the `PURGE()` operations.

Users must be the table owner to drop a partition. They must have `MODIFY ( INSERT | UPDATE | DELETE )` permissions in order to:

- Partition a projection/table
- Merge partitions
- Run mergeout, moveout or purge operations on a projection

`PARTITION_PROJECTION()` purges data while partitioning ROS containers if deletes were applied before the AHM epoch.

### Example

The following command forces a split of ROS containers on the `states_p_node01` projection:

```
=> SELECT PARTITION_PROJECTION ('states_p_node01');
      partition_projection
-----
Projection partitioned
(1 row)
```

### See Also

***DO\_TM\_TASK*** (page 406)

***DROP\_PARTITION*** (page 408)

***DUMP\_PARTITION\_KEYS*** (page 414)

***DUMP\_PROJECTION\_PARTITION\_KEYS*** (page 415)

***DUMP\_TABLE\_PARTITION\_KEYS*** (page 416)

***PARTITION\_TABLE*** (page 441)

Partitioning Tables in the Administrator's Guide

### **PARTITION\_TABLE**

Forces the system to break up any ROS containers that contain multiple distinct values of the partitioning expression. Only ROS containers with more than one distinct value participate in the split.

### Syntax

```
PARTITION_TABLE ( 'table_name' )
```

## Parameters

<code>table_name</code>	Specifies the name of the table.
-------------------------	----------------------------------

## Notes

PARTITION\_TABLE is similar to **PARTITION\_PROJECTION** (page 440), except that PARTITION\_TABLE works on the specified table.

Vertica internal operations (mergeout, refresh, and recovery) maintain partition separation except in certain cases:

- Recovery of a projection when the buddy projection from which the partition is recovering is identically sorted. If the projection is undergoing a full rebuild, it is recovered one ROS container at a time. The projection ends up with a storage layout identical to its buddy and is, therefore, properly segmented.

**Note:** In the case of a partial rebuild, all recovered data goes into a single ROS container and must be partitioned manually.

- Manual tuple mover operations often output a single storage container, combining any existing partitions; for example, after executing any of the `PURGE()` operations.

Users must be the table owner to drop a partition. They must have `MODIFY ( INSERT | UPDATE | DELETE )` permissions in order to:

- Partition a projection/table
- Merge partitions
- Run mergeout, moveout or purge operations on a projection

Partitioning functions take immutable functions only, in order that the same information be available across all nodes.

## Example

The following example creates a simple table called `states` and partitions data by state.

```
=> CREATE TABLE states (  
    year INTEGER NOT NULL,  
    state VARCHAR NOT NULL)  
    PARTITION BY state;  
=> CREATE PROJECTION states_p (state, year) AS  
    SELECT * FROM states  
    ORDER BY state, year UNSEGMENTED ALL NODES;
```

Now issue the command to partition table `states`:

```
=> SELECT PARTITION_TABLE('states');  
PARTITION_TABLE  
-----  
partition operation for projection 'states_p_node0004'  
partition operation for projection 'states_p_node0003'  
partition operation for projection 'states_p_node0002'  
partition operation for projection 'states_p_node0001'
```

---

(1 row)

### See Also

***DO\_TM\_TASK*** (page 406)

***DROP\_PARTITION*** (page 408)

***DUMP\_PARTITION\_KEYS*** (page 414)

***DUMP\_PROJECTION\_PARTITION\_KEYS*** (page 415)

***DUMP\_TABLE\_PARTITION\_KEYS*** (page 416)

***PARTITION\_PROJECTION*** (page 440)

Partitioning Tables in the Administrator's Guide

## Pattern Matching Functions

Used with the ***MATCH clause*** (page 763), the Vertica pattern matching functions return additional data about the patterns found/output. For example, you can use these functions to return values representing the name of the event or pattern that matched the input row, the sequential number of the match, or a partition-wide unique identifier for the instance of the pattern that matched.

Pattern matching is particularly useful for clickstream analysis where you might want to identify users' actions based on their Web browsing behavior (page clicks). A typical online clickstream funnel is:

Company home page -> product home page -> search -> results -> purchase online

Using the above clickstream funnel, you can search for a match on the user's sequence of web clicks and identify that the user:

- landed on the company home page
- navigated to the product page
- ran a search
- clicked a link from the search results
- made a purchase

For examples that use this clickstream model, see Event Series Pattern Matching in the Programmer's Guide.

### See Also

***MATCH Clause*** (page 763)

Event Series Pattern Matching in the Programmer's Guide

## EVENT\_NAME

Returns a VARCHAR value representing the name of the event that matched the row.

### Syntax

EVENT\_NAME ( )

## Notes

Pattern matching functions must be used in **MATCH clause** (page 763) syntax; for example, if you call `EVENT_NAME()` on its own, Vertica returns the following error message:

```
=> SELECT event_name();
ERROR:  query with pattern matching function event_name must include a MATCH clause
```

## Example

**Note:** This example uses the schema defined in Event Series Pattern Matching in the Programmer's Guide. For a more detailed example, see that topic.

The following statement analyzes users' browsing history on website2.com and identifies patterns where the user landed on website2.com from another Web site (Entry) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs the values for `EVENT_NAME()`, which is the name of the event that matched the row.

```
SELECT uid,
       sid,
       ts,
       refurl,
       pageurl,
       action,
       event_name ()
FROM clickstream_log
MATCH
(PARTITION BY uid, sid ORDER BY ts
 DEFINE
  Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
  Onsite AS PageURL ILIKE '%website2.com%' AND Action='V',
  Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'
 PATTERN
  P AS (Entry Onsite* Purchase)
 RESULTS ALL ROWS);
```

uid	sid	ts	refurl	pageurl	action	event_name
1	100	12:00:00	website1.com	website2.com/home	V	Entry
1	100	12:01:00	website2.com/home	website2.com/floby	V	Onsite
1	100	12:02:00	website2.com/floby	website2.com/shamwow	V	Onsite
1	100	12:03:00	website2.com/shamwow	website2.com/buy	P	Purchase
2	100	12:10:00	website1.com	website2.com/home	V	Entry
2	100	12:11:00	website2.com/home	website2.com/forks	V	Onsite
2	100	12:13:00	website2.com/forks	website2.com/buy	P	Purchase

(7 rows)

## See Also

**MATCH Clause** (page 763)

**MATCH\_ID** (page 437)

**PATTERN\_ID** (page 443)

Event Series Pattern Matching in the Programmer's Guide

## MATCH\_ID

Returns an `INTEGER` value that is the sequential number of the match.

## Syntax

```
MATCH_ID()
```

## Notes

Pattern matching functions must be used in **MATCH clause** (page 763) syntax; for example, if you call MATCH\_ID() on its own, Vertica returns the following error message:

```
=> SELECT match_id();
ERROR:  query with pattern matching function match_id must include a MATCH clause
```

## Example

**Note:** This example uses the schema defined in Event Series Pattern Matching in the Programmer's Guide. For a more detailed example, see that topic.

The following statement analyzes users' browsing history on website2.com and identifies patterns where the user landed on website2.com from another Web site (Entry) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs values for the MATCH\_ID(), which represents a sequential number of the match.

```
SELECT uid,
       sid,
       ts,
       refurl,
       pageurl,
       action,
       match_id()
FROM clickstream_log
MATCH
(PARTITION BY uid, sid ORDER BY ts
DEFINE
  Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
  Onsite AS PageURL ILIKE '%website2.com%' AND Action='V',
  Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'
PATTERN
  P AS (Entry Onsite* Purchase)
RESULTS ALL ROWS);
```

uid	sid	ts	refurl	pageurl	action	match_id
2	100	12:10:00	website1.com	website2.com/home	V	1
2	100	12:11:00	website2.com/home	website2.com/forks	V	2
2	100	12:13:00	website2.com/forks	website2.com/buy	P	3
1	100	12:00:00	website1.com	website2.com/home	V	1
1	100	12:01:00	website2.com/home	website2.com/floby	V	2
1	100	12:02:00	website2.com/floby	website2.com/shamwow	V	3
1	100	12:03:00	website2.com/shamwow	website2.com/buy	P	4

(7 rows)

## See Also

**MATCH Clause** (page 763)

**EVENT\_NAME** (page 419)

**PATTERN\_ID** (page 443)

Event Series Pattern Matching in the Programmer's Guide

## PATTERN\_ID

Returns an integer value that is a partition-wide unique identifier for the instance of the pattern that matched.

### Syntax

```
PATTERN_ID()
```

### Notes

Pattern matching functions must be used in **MATCH clause** (page 763) syntax; for example, if call PATTERN\_ID() on its own, Vertica returns the following error message:

```
=> SELECT pattern_id();
ERROR:  query with pattern matching function pattern_id must include a MATCH clause
```

### Example

**Note:** This example uses the schema defined in Event Series Pattern Matching in the Programmer's Guide. For a more detailed example, see that topic.

The following statement analyzes users' browsing history on website2.com and identifies patterns where the user landed on website2.com from another Web site (Entry) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs values for PATTERN\_ID(), which represents the partition-wide identifier for the instance of the pattern that matched.

```
SELECT uid,
       sid,
       ts,
       refurl,
       pageurl,
       action,
       pattern_id()
FROM clickstream_log
MATCH
(PARTITION BY uid, sid ORDER BY ts
 DEFINE
  Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
  Onsite AS PageURL ILIKE '%website2.com%' AND Action='V',
  Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'
 PATTERN
  P AS (Entry Onsite* Purchase)
 RESULTS ALL ROWS);
```

uid	sid	ts	refurl	pageurl	action	pattern_id
2	100	12:10:00	website1.com	website2.com/home	V	1
2	100	12:11:00	website2.com/home	website2.com/forks	V	1
2	100	12:13:00	website2.com/forks	website2.com/buy	P	1
1	100	12:00:00	website1.com	website2.com/home	V	1
1	100	12:01:00	website2.com/home	website2.com/floby	V	1
1	100	12:02:00	website2.com/floby	website2.com/shamwow	V	1
1	100	12:03:00	website2.com/shamwow	website2.com/buy	P	1

(7 rows)

### See Also

**MATCH Clause** (page 763)

**EVENT\_NAME** (page 419)

***MATCH\_ID*** (page 437)

Event Series Pattern Matching in the Programmer's Guide

## Projection Management Functions

This section contains projection management functions specific to Vertica.

See also the following SQL system tables:

- ***V\_CATALOG.PROJECTIONS*** (page 811)
- ***V\_CATALOG.PROJECTION\_COLUMNS*** (page 805)
- ***V\_MONITOR.PROJECTION\_REFRESHES*** (page 863)
- ***V\_MONITOR.PROJECTION\_STORAGE*** (page 865)

### EVALUATE\_DELETE\_PERFORMANCE

Evaluates projections for potential ***DELETE*** (page 699) performance issues. If there are issues found, a warning message is displayed. For steps you can take to resolve delete and update performance issues, see *Optimizing Deletes and Updates for Performance* in the Administrator's Guide. This function uses data sampling to determine whether there are any issues with a projection. Therefore, it does not generate false-positives warnings, but it can miss some cases where there are performance issues.

**Note:** Optimizing for delete performance is the same as optimizing for update performance. So, you can use this function to help optimize a projection for updates as well as deletes.

#### Syntax

```
EVALUATE_DELETE_PERFORMANCE ( 'target' )
```

#### Parameters

<i>target</i>	<p>The name of a projection or table. If you supply the name of a projection, only that projection is evaluated for <b><i>DELETE</i></b> performance issues. If you supply the name of a table, then all of the projections anchored to the table will be evaluated for issues.</p> <p>If you do not provide a projection or table name, <b><i>EVALUATE_DELETE_PERFORMANCE</i></b> examines all of the projections that you can access for <b><i>DELETE</i></b> performance issues. Depending on the size of your database, this may take a long time.</p>
---------------	--

**Note:** When evaluating multiple projections, ***EVALUATE\_DELETE\_PERFORMANCE*** reports up to ten projections that have issues, and refers you to a table that contains the full list of issues it has found.

#### Example

The following example demonstrates how you can use ***EVALUATE\_DELETE\_PERFORMANCE*** to evaluate your projections for slow ***DELETE*** performance.

```
=> create table example (A int, B int,C int);
CREATE TABLE
=> create projection one_sort (A,B,C) as (select A,B,C from example) order by A;
CREATE PROJECTION
=> create projection two_sort (A,B,C) as (select A,B,C from example) order by A,B;
```

```
CREATE PROJECTION
=> select evaluate_delete_performance('one_sort');
       evaluate_delete_performance
-----
No projection delete performance concerns found.
(1 row)
=> select evaluate_delete_performance('two_sort');
       evaluate_delete_performance
-----
No projection delete performance concerns found.
(1 row)
```

The previous example showed that there was no structural issue with the projection that would cause poor DELETE performance. However, the data contained within the projection can create potential delete issues if the sorted columns do not uniquely identify a row or small number of rows. In the following example, Perl is used to populate the table with data using a nested series of loops. The inner loop populates column C, the middle loop populates column B, and the outer loop populates column A. The result is column A contains only three distinct values (0, 1, and 2), while column B slowly varies between 20 and 0 and column C changes in each row.

EVALUATE\_DELETE\_PERFORMANCE is run against the projections again to see if the data within the projections causes any potential DELETE performance issues.

```
=> \! perl -e 'for ($i=0; $i<3; $i++) { for ($j=0; $j<21; $j++) { for ($k=0; $k<19; $k++) { printf "%d,%d,%d\n", $i,$j,$k;}}}' | /opt/vertica/bin/vsql -c "copy example from stdin delimiter ',' direct;"
Password:
=> select * from example;
```

```
 A | B | C
-----
 0 | 20 | 18
 0 | 20 | 17
 0 | 20 | 16
 0 | 20 | 15
 0 | 20 | 14
 0 | 20 | 13
 0 | 20 | 12
 0 | 20 | 11
 0 | 20 | 10
 0 | 20 | 9
 0 | 20 | 8
 0 | 20 | 7
 0 | 20 | 6
 0 | 20 | 5
 0 | 20 | 4
 0 | 20 | 3
 0 | 20 | 2
 0 | 20 | 1
 0 | 20 | 0
 0 | 19 | 18
1157 rows omitted
 2 | 1 | 0
 2 | 0 | 18
 2 | 0 | 17
 2 | 0 | 16
 2 | 0 | 15
 2 | 0 | 14
 2 | 0 | 13
 2 | 0 | 12
 2 | 0 | 11
 2 | 0 | 10
 2 | 0 | 9
 2 | 0 | 8
 2 | 0 | 7
 2 | 0 | 6
 2 | 0 | 5
```

```

2 | 0 | 4
2 | 0 | 3
2 | 0 | 2
2 | 0 | 1
2 | 0 | 0

```

```

=> SELECT COUNT (*) FROM example;
COUNT
-----
    1197
(1 row)
=> SELECT COUNT (DISTINCT A) FROM example;
COUNT
-----
        3
(1 row)

=> select evaluate_delete_performance('one_sort');
          evaluate_delete_performance
-----
Projection exhibits delete performance concerns.
(1 row)
release=> select evaluate_delete_performance('two_sort');
          evaluate_delete_performance
-----
No projection delete performance concerns found.
(1 row)

```

The `one_sort` projection has potential delete issues since it only sorts on column A which has few distinct values. This means that each value in the sort column corresponds to many rows in the projection, which negatively impacts DELETE performance. Since the `two_sort` projection is sorted on columns A and B, each combination of values in the two sort columns identifies just a few rows, allowing deletes to be performed faster.

Not supplying a projection name results in all of the projections you can access being evaluated for DELETE performance issues.

```

=> select evaluate_delete_performance();
          evaluate_delete_performance
-----
The following projection exhibits delete performance concerns:
"public"."one_sort"
See v_catalog.projection_delete_concerns for more details.
(1 row)

```

## GET\_PROJECTION\_STATUS

Returns information relevant to the status of a projection.

### Syntax

```
GET_PROJECTION_STATUS ( [ schema-name. ]projection );
```

### Parameters

<code>[<i>schema-name</i>.]<i>projection</i></code>	Is the name of the projection for which to display status. When using more than one schema, specify the schema that contains the projection.
---	--

## Description

GET\_PROJECTION\_STATUS returns information relevant to the status of a projection:

- The current K-safety status of the database
- The number of nodes in the database
- Whether the projection is segmented
- The number and names of buddy projections
- Whether the projection is safe
- Whether the projection is up-to-date
- Whether statistics have been computed for the projection

## Notes

- You can use GET\_PROJECTION\_STATUS to monitor the progress of a projection data refresh. See **ALTER PROJECTION** (page 580).
- When using GET\_PROJECTION\_STATUS or GET\_PROJECTIONS you must provide the name and node (for example, ABC\_NODE01) instead of just ABC.
- To view a list of the nodes in a database, use the View Database Command in the Administration Tools.

## Examples

```
=> SELECT GET_PROJECTION_STATUS('public.customer_dimension_site01');
```

```
-----  
GET_PROJECTION_STATUS  
-----  
Current system K is 1.  
# of Nodes: 4.  
public.customer_dimension_site01 [Segmented: No] [Seg Cols: ] [K: 3]  
[public.customer_dimension_site04, public.customer_dimension_site03,  
public.customer_dimension_site02] [Safe: Yes] [UptoDate: Yes][Stats: Yes]
```

## See Also

**ALTER PROJECTION** (page 580)

**GET\_PROJECTIONS** (page 429)

## GET\_PROJECTIONS, GET\_TABLE\_PROJECTIONS

**Note:** This function was formerly named GET\_TABLE\_PROJECTIONS(). Vertica still supports the former function name.

Returns information relevant to the status of a table:

- The current K-safety status of the database
- The number of sites (nodes) in the database
- The number of projections for which the specified table is the anchor table
- For each projection:
  - The projection's buddy projections
  - Whether the projection is segmented

- Whether the projection is safe
- Whether the projection is up-to-date

## Syntax

```
GET_PROJECTIONS ( [ schema-name.] table )
```

## Parameters

<code>[schema-name.] table</code>	Is the name of the table for which to list projections. When using more than one schema, specify the schema that contains the table.
-----------------------------------	--

## Notes

- You can use GET\_PROJECTIONS to monitor the progress of a projection data refresh. See **ALTER PROJECTION** (page 580).
- When using GET\_PROJECTIONS or GET\_PROJECTION\_STATUS for replicated projections created using the ALL NODES syntax, you must provide the name and node (for example, ABC\_NODE01 instead of just ABC).
- To view a list of the nodes in a database, use the View Database Command in the Administration Tools.

## Examples

The following example gets information about the store\_dimension table in the VMart schema:

```
=> SELECT GET_PROJECTIONS('store.store_dimension');
-----
Current system K is 1.
# of Nodes: 4.
Table store.store_dimension has 4 projections.

Projection Name: [Segmented] [Seg Cols] [# of Buddies] [Buddy Projections] [Safe] [UptoDate]
-----
store.store_dimension_node0004 [Segmented: No] [Seg Cols: ] [K: 3] [store.store_dimension_node0003,
store.store_dimension_node0002, store.store_dimension_node0001] [Safe: Yes] [UptoDate: Yes][Stats:
Yes]
store.store_dimension_node0003 [Segmented: No] [Seg Cols: ] [K: 3] [store.store_dimension_node0004,
store.store_dimension_node0002, store.store_dimension_node0001] [Safe: Yes] [UptoDate: Yes][Stats:
Yes]
store.store_dimension_node0002 [Segmented: No] [Seg Cols: ] [K: 3] [store.store_dimension_node0004,
store.store_dimension_node0003, store.store_dimension_node0001] [Safe: Yes] [UptoDate: Yes][Stats:
Yes]
store.store_dimension_node0001 [Segmented: No] [Seg Cols: ] [K: 3] [store.store_dimension_node0004,
store.store_dimension_node0003, store.store_dimension_node0002] [Safe: Yes] [UptoDate: Yes][Stats:
Yes]
(1 row)
```

## See Also

**ALTER PROJECTION** (page 580)

**GET\_PROJECTION\_STATUS** (page 428)

## REFRESH

Performs a synchronous, optionally-targeted refresh of a specified table's projections.

Information about a refresh operation—whether successful or unsuccessful—is maintained in the **PROJECTION\_REFRESHES** (page 863) system table until either the **CLEAR\_PROJECTION\_REFRESHES** (page 395)() function is executed or the storage quota for the table is exceeded. The `PROJECTION_REFRESHES.IS_EXECUTING` column returns a boolean value that indicates whether the refresh is currently running (t) or occurred in the past (f).

### Syntax

```
REFRESH ( [schema_name.]table_name [ , ... ] )
```

### Parameters

<code>[schema_name.]table_name</code>	In the optionally-specified schema, <code>table_name</code> is the name of a specific table that contains the projections to be refreshed. When using more than one schema, specify the schema that contains the table.
---------------------------------------	---

### Returns

Column Name	Description
Projection Name	The name of the projection that is targeted for refresh.
Anchor Table	The name of the projection's associated anchor table.
Status	The status of the projection: <ul style="list-style-type: none"> <li>▪ Queued — Indicates that a projection is queued for refresh.</li> <li>▪ Refreshing — Indicates that a refresh for a projection is in process.</li> <li>▪ Refreshed — Indicates that a refresh for a projection has successfully completed.</li> <li>▪ Failed — Indicates that a refresh for a projection did not successfully complete.</li> </ul>
Refresh Method	The method used to refresh the projection: <ul style="list-style-type: none"> <li>▪ Buddy – Uses the contents of a buddy to refresh the projection. This method maintains historical data. This enables the projection to be used for historical queries.</li> <li>▪ Scratch – Refreshes the projection without using a buddy. This method does not generate historical data. This means that the projection cannot participate in historical queries from any point before the projection was refreshed.</li> </ul>
Error Count	The number of times a refresh failed for the projection.
Duration (sec)	The length of time that the projection refresh ran in seconds.

### Notes

- Unlike `START_REFRESH()`, which runs in the background, `REFRESH()` runs in the foreground of the caller's session.

- The REFRESH() function refreshes only the projections in the specified table.
- If you run REFRESH() without arguments, it refreshes all non up-to-date projections. If the function returns a header string with no results, then no projections needed refreshing.

### Example

The following command refreshes the projections in tables `t1` and `t2`:

```
=> SELECT REFRESH('t1, t2');
refresh
```

```
-----
Refresh completed with the following outcomes:
Projection Name: [Anchor Table] [Status] [Refresh Method] [Error Count] [Duration (sec)]
-----
"public"."t1_p": [t1] [refreshed] [scratch] [0] [0]
"public"."t2_p": [t2] [refreshed] [scratch] [0] [0]
```

This next command shows that only the projection on table `t` was refreshed:

```
=> SELECT REFRESH('allow, public.deny, t');"
refresh
```

```
-----
Refresh completed with the following outcomes:
Projection Name: [Anchor Table] [Status] [Refresh Method] [Error Count] [Duration (sec)]
-----
"/n/a"/n/a": [n/a] [failed: insufficient permissions on table "allow"] [] [1] [0]
"/n/a"/n/a": [n/a] [failed: insufficient permissions on table "public.deny"] [] [1] [0]
"public"."t_p1": [t] [refreshed] [scratch] [0] [0]
```

### See Also

**CLEAR\_PROJECTION\_REFRESHES** (page 395)

**PROJECTION\_REFRESHES** (page 863)

**START\_REFRESH** (page 470)

Clearing PROJECTION\_REFRESHES History in the Administrator's Guide

### START\_REFRESH

Transfers data to projections that are not able to participate in query execution due to missing or out-of-date data.

### Syntax

```
START_REFRESH()
```

### Notes

- When a design is deployed through the Database Designer, it is automatically refreshed. See Deploying Designs in the Administrator's Guide.
- All nodes must be up in order to start a refresh.
- START\_REFRESH() has no effect if a refresh is already running.
- A refresh is run asynchronously.
- Shutting down the database ends the refresh.
- To view the progress of the refresh, see the **PROJECTION\_REFRESHES** (page 863) and **PROJECTIONS** (page 811) system tables.

- If a projection is updated from scratch, the data stored in the projection represents the table columns as of the epoch in which the refresh commits. As a result, the query optimizer might not choose the new projection for AT EPOCH queries that request historical data at epochs older than the refresh epoch of the projection. Projections refreshed from buddies retain history and can be used to answer historical queries.

Vertica internal operations (mergeout, refresh, and recovery) maintain partition separation except in certain cases:

- Recovery of a projection when the buddy projection from which the partition is recovering is identically sorted. If the projection is undergoing a full rebuild, it is recovered one ROS container at a time. The projection ends up with a storage layout identical to its buddy and is, therefore, properly segmented.

**Note:** In the case of a partial rebuild, all recovered data goes into a single ROS container and must be partitioned manually.

- Manual tuple mover operations often output a single storage container, combining any existing partitions; for example, after executing any of the `PURGE()` operations.

### Example

The following command starts the refresh operation:

```
=> SELECT START_REFRESH();
      start_refresh
```

```
-----
Starting refresh background process.
```

### See Also

***CLEAR\_PROJECTION\_REFRESHES*** (page 395)

***MARK\_DESIGN\_KSAFE*** (page 436)

***PROJECTION\_REFRESHES*** (page 863)

***PROJECTIONS*** (page 811)

Clearing PROJECTION\_REFRESHES History in the Administrator's Guide

## Purge Functions

This section contains purge functions specific to Vertica.

### PURGE

Purges all projections in the physical schema. Permanently removes deleted data from physical storage so that the disk space can be reused. You can purge historical data up to and including the epoch in which the Ancient History Mark is contained.

## Syntax

PURGE ( )

## Notes

- PURGE() was formerly named PURGE\_ALL\_PROJECTIONS. Vertica supports both function calls.
- Manual tuple mover operations, such as the PURGE() operations, often output a single storage container, combining any existing partitions. For example, if PURGE() is used on a non-partitioned table, all ROS containers are combined into a single container. Non-partitioned tables cannot be re-partitioned into multiple ROS containers. A purge operation on a partitioned table also results in a single ROS.
- To re-partition the data into multiple ROS containers, use the **PARTITION\_TABLE** (page 441)() function.

**Caution:** PURGE could temporarily take up significant disk space while the data is being purged.

## See Also

**MERGE\_PARTITIONS** (page 439)

**PARTITION\_TABLE** (page 441)

**PURGE\_PROJECTION** (page 445)

**PURGE\_TABLE** (page 445)

**STORAGE\_CONTAINERS** (page 890)

Purging Deleted Data in the Administrator's Guide

## PURGE\_PROJECTION

Purges the specified projection. Permanently removes deleted data from physical storage so that the disk space can be reused. You can purge historical data up to and including the epoch in which the Ancient History Mark is contained.

**Caution:** PURGE\_PROJECTION could temporarily take up significant disk space while the data is being purged.

## Syntax

PURGE\_PROJECTION ( [ *schema-name.* ] *projection\_name* )

## Parameters

<i>projection_name</i>	Is the name of a specific projection. When using more than one schema, specify the schema that contains the projection.
------------------------	---

## Notes

See **PURGE** (page 444) for notes about the outcome of purge operations.

**See Also****MERGE\_PARTITIONS** (page 439)**PURGE\_TABLE** (page 445)**STORAGE\_CONTAINERS** (page 890)

Purging Deleted Data in the Administrator's Guide

**PURGE\_TABLE**

**Note:** This function was formerly named `PURGE_TABLE_PROJECTIONS()`. Vertica still supports the former function name.

Purges all projections of the specified table. Permanently removes deleted data from physical storage so that the disk space can be reused. You can purge historical data up to and including the epoch in which the Ancient History Mark is contained.

**Syntax**

```
PURGE_TABLE ( [ schema_name.] table_name )
```

**Parameters**

<code>[schema_name.] table_name</code>	Is the name of a specific table in the optionally-specified logical schema. When using more than one schema, specify the schema that contains the projection.
--	--

**Caution:** `PURGE_TABLE` could temporarily take up significant disk space while the data is being purged.

**Example**

The following example purges all projections for the store sales fact table located in the Vmart schema:

```
=> SELECT PURGE_TABLE('store.store_sales_fact');
```

**See Also****PURGE** (page 444) for notes about the outcome of purge operations.**MERGE\_PARTITIONS** (page 439)**PURGE\_TABLE** (page 445)**STORAGE\_CONTAINERS** (page 890)

Purging Deleted Data in the Administrator's Guide

## Regular Expression Functions

A regular expression lets you perform pattern matching on strings of characters. The regular expression syntax allows you to very precisely define the pattern used to match strings, giving you much greater control than the wildcard matching used in the **LIKE** (page 59) predicate. Vertica's regular expression functions let you perform tasks such as determining if a string value matches a pattern, extracting a portion of a string that matches a pattern, or counting the number of times a string matches a pattern.

Vertica uses the **Perl Compatible Regular Expression library** <http://www.pcre.org/> (PCRE) to evaluate regular expressions. As its name implies, PCRE's regular expression syntax is compatible with the syntax used by the Perl 5 programming language. You can read **PCRE's documentation on its regular expression syntax** <http://vcs.pcre.org/viewvc/code/trunk/doc/html/pcrepattern.html?view=co>. However, you might find the **Perl Regular Expressions Documentation** (<http://perldoc.perl.org/perlre.html>) to be a better introduction, especially if you are unfamiliar with regular expressions.

**Note:** The regular expression functions operate on UTF-8 strings by default. You may want to use the **ISUTF8** (page 331) function to ensure the strings you want to pass to the regular expression functions are actually valid UTF-8 strings.

### ISUTF8

Tests whether a string is a valid UTF-8 string. Returns true if the string conforms to UTF-8 standards, and false otherwise. This function is useful to test strings for UTF-8 compliance before passing them to one of the regular expression functions, such as **REGEXP\_LIKE** (page 452), which expect UTF-8 characters by default.

#### Syntax

```
ISUTF8( string );
```

#### Parameters

<i>string</i>	The string to test for UTF-8 compliance.
---------------	--

#### Examples

```
=> SELECT ISUTF8(E'\xc2\xbf'); -- UTF-8 INVERTED QUESTION MARK
       ISUTF8
```

```
-----
t
(1 row)
```

```
=> SELECT ISUTF8(E'\xc2\xc0'); -- UNDEFINED UTF-8 CHARACTER
       ISUTF8
```

```
-----
f
(1 row)
```

## REGEXP\_COUNT

Returns the number times a regular expression matches a string.

### Syntax

```
REGEXP_COUNT( string, pattern [, position [, regexp_modifier ] ] )
```

### Parameters

<i>string</i>	The string to be searched for matches.
<i>pattern</i>	The regular expression to search for within the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <b>Perl Regular Expressions Documentation</b> ( <a href="http://perldoc.perl.org/perlre.html">http://perldoc.perl.org/perlre.html</a> ) for details.
<i>position</i>	The number of characters from the start of the string where the function should start searching for matches. The default value, 1, means to start searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 starts searching for a match to the pattern that many characters into the string.
<i>regexp_modifier</i>	<p>A string containing one or more single-character flags that change how the regular expression is matched against the string:</p> <ul style="list-style-type: none"> <li>b        Treat strings as binary octets rather than UTF-8 characters.</li> <li>c        Forces the match to be case sensitive (the default).</li> <li>i        Forces the match to be case insensitive.</li> <li>m        Treats the string being matched as multiple lines. With this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Ordinarily, these operators only match the start and end of the string.</li> <li>n        Allows the single character regular expression operator (.) to match a newline (\n). Normally, the . operator will match any character except a newline.</li> <li>x        Allows you to document your regular expressions. It causes all unescaped space characters and comments in the regular expression to be ignored. Comments start with a hash character (#) and end with a newline. All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.</li> </ul>

## Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

## Examples

Count the number of occurrences of the substring "an" in the string "A man, a plan, a canal, Panama."

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', 'an');
   REGEXP_COUNT
-----
                4
(1 row)
```

Find the number of occurrences of the substring "an" in the string "a man, a plan, a canal: Panama" starting with the fifth character.

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', 'an',5);
   REGEXP_COUNT
-----
                3
(1 row)
```

Find the number of occurrences of a substring containing a lower-case character followed by "an." In the first example, the query does not have a modifier. In the second example, the "i" query modifier is used to force the regular expression to ignore case.

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', '[a-z]an');
   REGEXP_COUNT
-----
                3
(1 row)
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', '[a-z]an', 1, 'i');
   REGEXP_COUNT
-----
                4
```

## REGEXP\_INSTR

Returns the starting or ending position in a string where a regular expression matches. This function returns 0 if no match for the regular expression is found in the string.

### Syntax

```
REGEXP_INSTR( string, pattern [, position [, occurrence
... [, return_position [, regexp_modifier ] ] ] )
```

### Parameters

<i>string</i>	The string to search for the pattern.
---------------	---------------------------------------

<i>pattern</i>	The regular expression to search for within the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <b><i>Perl Regular Expressions Documentation</i></b> ( <a href="http://perldoc.perl.org/perlre.html">http://perldoc.perl.org/perlre.html</a> ) for details.
<i>position</i>	The number of characters from the start of the string where the function should start searching for matches. The default value, 1, means to start searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 starts searching for a match to the pattern that many characters into the string.
<i>occurrence</i>	Controls which occurrence of a match between the string and the pattern is returned. With the default value (1), the function returns the position of the first substring that matches the pattern. You can use this parameter to find the position of additional matches between the string and the pattern. For example, set this parameter to 3 to find the position of the third substring that matched the pattern.
<i>return_position</i>	Sets the position within the string that is returned. When set to the default value (0), this function returns the position in the string of the first character of the substring that matched the pattern. If you set this value to 1, the function returns the position of the first character after the end of the matching substring.
<i>regexp_modifier</i>	A string containing one or more single-character flags that change how the regular expression is matched against the string:  b        Treat strings as binary octets rather than UTF-8 characters.  c        Forces the match to be case sensitive (the default).  i        Forces the match to be case insensitive.  m        Treats the string being matched as multiple lines. With this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Ordinarily, these operators only match the start and end of the string.  n        Allows the single character regular expression operator (.) to match a newline (\n). Normally, the . operator will match any character except a newline.

x	Allows you to document your regular expressions. It causes all unescaped space characters and comments in the regular expression to be ignored. Comments start with a hash character (#) and end with a newline. All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.
---	--

## Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

## Examples

Find the first occurrence of a sequence of letters starting with the letter e and ending with the letter y in the phrase "easy come, easy go."

```
=> SELECT REGEXP_INSTR('easy come, easy go','e\w*y');
REGEXP_INSTR
-----
1
(1 row)
```

Find the first occurrence of a sequence of letters starting with the letter e and ending with the letter y starting at the second character in the string "easy come, easy go."

```
=> SELECT REGEXP_INSTR('easy come, easy go','e\w*y',2);
REGEXP_INSTR
-----
12
(1 row)
```

Find the second sequence of letters starting with the letter e and ending with the letter y in the string "easy come, easy go" starting at the first character.

```
=> SELECT REGEXP_INSTR('easy come, easy go','e\w*y',1,2);
REGEXP_INSTR
-----
12
(1 row)
```

Find the position of the first character after the first whitespace in the string "easy come, easy go."

```
=> SELECT REGEXP_INSTR('easy come, easy go','\s',1,1,1);
REGEXP_INSTR
-----
6
(1 row)
```

## REGEXP\_LIKE

Returns true if the string matches the regular expression. This function is similar to the **LIKE-predicate** (page 59), except that it uses regular expressions rather than simple wildcard character matching.

### Syntax

```
REGEXP_LIKE( string, pattern [, modifiers ] )
```

### Parameters

<i>string</i>	The string to match against the regular expression.
<i>pattern</i>	A string containing the regular expression to match against the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <b>Perl Regular Expressions Documentation</b> ( <a href="http://perldoc.perl.org/perlre.html">http://perldoc.perl.org/perlre.html</a> ) for details.
<i>modifiers</i>	<p>A string containing one or more single-character flags that change how the regular expression is matched against the string:</p> <ul style="list-style-type: none"><li>b      Treat strings as binary octets rather than UTF-8 characters.</li><li>c      Forces the match to be case sensitive (the default).</li><li>i      Forces the match to be case insensitive.</li><li>m      Treats the string being matched as multiple lines. With this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Ordinarily, these operators only match the start and end of the string.</li><li>n      Allows the single character regular expression operator (.) to match a newline (\n). Normally, the . operator will match any character except a newline.</li><li>x      Allows you to document your regular expressions. It causes all unescaped space characters and comments in the regular expression to be ignored. Comments start with a hash character (#) and end with a newline. All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.</li></ul>

## Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

## Examples

This example creates a table containing several strings to demonstrate regular expressions.

```
=> create table t (v varchar);
CREATE TABLE
=> create projection t1 as select * from t;
CREATE PROJECTION
=> COPY t FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> aaa
>> Aaa
>> abc
>> abc1
>> 123
>> \.
=> SELECT * FROM t;
      v
-----
aaa
Aaa
abc
abc1
123
(5 rows)
```

Select all records in the table that contain the letter "a."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'a');
      v
-----
Aaa
aaa
abc
abc1
(4 rows)
```

Select all of the rows in the table that start with the letter "a."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, '^a');
      v
-----
aaa
abc
abc1
(3 rows)
```

Select all rows that contain the substring "aa."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'aa');
      v
-----
     Aaa
     aaa
(2 rows)
```

Select all rows that contain a digit.

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, '\d');
      v
-----
     123
     abc1
(2 rows)
```

Select all rows that contain the substring "aaa."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'aaa');
      v
-----
     aaa
(1 row)
```

Select all rows that contain the substring "aaa" using case insensitive matching.

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'aaa', 'i');
      v
-----
     Aaa
     aaa
(2 rows)
```

Select rows that contain the substring "a b c."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'a b c');
      v
-----
(0 rows)
```

---

Select rows that contain the substring "a b c" ignoring space within the regular expression.

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'a b c', 'x');
      v
-----
     abc
     abc1
(2 rows)
```

Add multi-line rows to demonstrate using the "m" modifier.

```
=> COPY t FROM stdin RECORD TERMINATOR '!';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
```

```
>> Record 1 line 1
>> Record 1 line 2
>> Record 1 line 3!
>> Record 2 line 1
>> Record 2 line 2
>> Record 2 line 3!
>> \.
```

Select rows that start with the substring "Record" and end with the substring "line 2."

```
=> SELECT v from t WHERE REGEXP_LIKE(v, '^Record.*line 2$');
v
-----
(0 rows)
```

Select rows that start with the substring "Record" and end with the substring "line 2," treating multiple lines as separate strings.

```
=> SELECT v from t WHERE REGEXP_LIKE(v, '^Record.*line 2$', 'm');
v
-----
Record 2 line 1
Record 2 line 2
Record 2 line 3
Record 1 line 1
Record 1 line 2
Record 1 line 3
(2 rows)
```

## REGEXP\_REPLACE

Replace all occurrences of a substring that match a regular expression with another substring. It is similar to the **REPLACE** (page 346) function, except it uses a regular expression to select the substring to be replaced.

### Syntax

```
REGEXP_REPLACE( string, target [, replacement [, position [, occurrence
... [, regexp_modifiers ] ] ] ] )
```

### Parameters

<i>string</i>	The string whose to be searched and replaced.
<i>target</i>	The regular expression to search for within the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <b>Perl Regular Expressions Documentation</b> ( <a href="http://perldoc.perl.org/perlre.html">http://perldoc.perl.org/perlre.html</a> ) for details.
<i>replacement</i>	The string to replace matched substrings. If not supplied, the matched substrings are deleted. This string can contain backreferences for substrings captured by the regular expression. The first captured substring is

	inserted into the replacement string using \1, the second \2, and so on.
<i>position</i>	The number of characters from the start of the string where the function should start searching for matches. The default value, 1, means to start searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 starts searching for a match to the pattern that many characters into the string.
<i>occurrence</i>	Controls which occurrence of a match between the string and the pattern is replaced. With the default value (0), the function replaces all matching substrings with the replacement string. For any value above zero, the function replaces just a single occurrence. For example, set this parameter to 3 to replace the third substring that matched the pattern.
<i>regexp_modifier</i>	<p>A string containing one or more single-character flags that change how the regular expression is matched against the string:</p> <ul style="list-style-type: none"> <li>b      Treat strings as binary octets rather than UTF-8 characters.</li> <li>c      Forces the match to be case sensitive (the default).</li> <li>i      Forces the match to be case insensitive.</li> <li>m      Treats the string being matched as multiple lines. With this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Ordinarily, these operators only match the start and end of the string.</li> <li>n      Allows the single character regular expression operator (.) to match a newline (\n). Normally, the . operator will match any character except a newline.</li> <li>x      Allows you to document your regular expressions. It causes all unescaped space characters and comments in the regular expression to be ignored. Comments start with a hash character (#) and end with a newline. All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.</li> </ul>

## Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

## Examples

Find groups of "word characters" (letters, numbers and underscore) ending with "thy" in the string "healthy, wealthy, and wise" and replace them with nothing.

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise', '\w+thy');
       REGEXP_REPLACE
-----
, , and wise
(1 row)
```

Find groups of word characters ending with "thy" and replace with the string "something."

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise', '\w+thy', 'something');
       REGEXP_REPLACE
-----
something, something, and wise
(1 row)
```

Find groups of word characters ending with "thy" and replace with the string "something" starting at the third character in the string.

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise', '\w+thy', 'something', 3);
       REGEXP_REPLACE
-----
hesomething, something, and wise
(1 row)
```

Replace the second group of word characters ending with "thy" with the string "something."

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise', '\w+thy', 'something', 1,
2);
       REGEXP_REPLACE
-----
healthy, something, and wise
(1 row)
```

Find groups of word characters ending with "thy" capturing the letters before the "thy", and replace with the captured letters plus the letters "ish."

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise', '(\w+)thy', '\1ish');
       REGEXP_REPLACE
-----
healish, wealish, and wise
(1 row)
```

Create a table to demonstrate replacing strings in a query.

```
=> CREATE TABLE customers (name varchar(50), phone varchar(11));
CREATE TABLE
=> CREATE PROJECTION customers1 AS SELECT * FROM customers;
CREATE PROJECTION
```

```
=> COPY customers FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> Able, Adam|17815551234
>> Baker, Bob|18005551111
>> Chu, Cindy|16175559876
>> Dodd, Dinara|15083452121
>> \.
```

Query the customers, using REGEXP\_REPLACE to format the phone numbers.

```
=> SELECT name, REGEXP_REPLACE(phone, '(\d)(\d{3})(\d{3})(\d{4})', '\1-(\2)\3-\4') as phone FROM customers;
   name      |      phone
-----+-----
Able, Adam   | 1-(781) 555-1234
Baker, Bob   | 1-(800) 555-1111
Chu, Cindy   | 1-(617) 555-9876
Dodd, Dinara | 1-(508) 345-2121
(4 rows)
```

## REGEXP\_SUBSTR

Returns the substring that matches a regular expression within a string. If no matches are found, this function returns NULL. This is different than an empty string, which can be returned by this function if the regular expression matches a zero-length string.

### Syntax

```
REGEXP_SUBSTR( string, pattern [, position [, occurrence
... [, regexp_modifier ] ] ] )
```

### Parameters

<i>string</i>	The string to search for the pattern.
<i>pattern</i>	The regular expression to find the substring to be extracted. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <b>Perl Regular Expressions Documentation</b> ( <a href="http://perldoc.perl.org/perlre.html">http://perldoc.perl.org/perlre.html</a> ) for details.
<i>position</i>	The character in the string where the search for a match should start. The default value, 1, starts the search at the beginning of the string. If you supply a value larger than 1 for this parameter, the function will start searching that many characters into the string.
<i>occurrence</i>	Controls which matching substring is returned by the function. When given the default value (1), the function will return the first matching substring it finds in the string. By

	setting this value to a number greater than 1, this function will return subsequent matching substrings. For example, setting this parameter to 3 will return the third substring that matches the regular expression within the string.
<i>regexp_modifier</i>	<p>A string containing one or more single-character flags that change how the regular expression is matched against the string:</p> <p>b        Treat strings as binary octets rather than UTF-8 characters.</p> <p>c        Forces the match to be case sensitive (the default).</p> <p>i        Forces the match to be case insensitive.</p> <p>m        Treats the string being matched as multiple lines. With this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Ordinarily, these operators only match the start and end of the string.</p> <p>n        Allows the single character regular expression operator (.) to match a newline (\n). Normally, the . operator will match any character except a newline.</p> <p>x        Allows you to document your regular expressions. It causes all unescaped space characters and comments in the regular expression to be ignored. Comments start with a hash character (#) and end with a newline. All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.</p>

## Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

## Examples

Select the first substring of letters that end with "thy."

```
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise','\w+thy');
REGEXP_SUBSTR
-----
healthy
(1 row)
```

Select the first substring of letters that ends with "thy" starting at the second character in the string.

```
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise','\w+thy',2);
REGEXP_SUBSTR
-----
ealthy
(1 row)
```

Select the second substring of letters that ends with "thy."

```
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise','\w+thy',1,2);
REGEXP_SUBSTR
-----
wealthy
(1 row)
```

## Session Management Functions

This section contains session management functions specific to Vertica.

See also the SQL system table *V\_MONITOR.SESSIONS* (page 888)

### CANCEL\_REFRESH

Cancel refresh-related internal operations initiated by *START\_REFRESH()*.

#### Syntax

```
CANCEL_REFRESH()
```

#### Notes

- Refresh tasks run in a background thread in an internal session, so you cannot use *INTERRUPT\_STATEMENT* (page 430) to cancel those statements. Instead, use *CANCEL\_REFRESH* to cancel statements that are run by refresh-related internal sessions.
- Run *CANCEL\_REFRESH()* on the same node on which *START\_REFRESH()* was initiated.
- *CANCEL\_REFRESH()* cancels the refresh operation running on a node, waits for the cancelation to complete, and returns *SUCCESS*.
- Only one set of refresh operations runs on a node at any time.

#### See Also

*INTERRUPT\_STATEMENT* (page 430)

*SESSIONS* (page 888)

*START\_REFRESH* (page 470)

*PROJECTION\_REFRESHES* (page 863)

### CLOSE\_ALL\_SESSIONS

Closes all external sessions except the one issuing the *CLOSE\_ALL\_SESSIONS* functions.

#### Syntax

```
CLOSE_ALL_SESSIONS()
```

## Notes

Closing of the sessions is processed asynchronously. It might take some time for the session to be closed. Check the **SESSIONS** (page 888) table for the status.

Database shutdown is prevented if new sessions connect after the `CLOSE_SESSION` or `CLOSE_ALL_SESSIONS()` command is invoked (and before the database is actually shut down). See **Controlling Sessions** below.

## Message

```
close_all_sessions | Close all sessions command sent.
Check SESSIONS for progress.
```

## Examples

Two user sessions opened, each on a different node:

```
vmartdb=> SELECT * FROM sessions;
-[ RECORD 1
]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (select * from sessions;)
statement_start    | 2011-01-03 15:36:13.896288
statement_id       | 10
last_statement_duration_us | 14978
current_statement  | select * from sessions;
ssl_state          | None
authentication_method | Trust
-[ RECORD 2
]-----+-----
node_name          | v_vmartdb_node0002
user_name          | dbadmin
client_hostname    | 127.0.0.1:57174
client_pid         | 30117
login_timestamp    | 2011-01-03 15:33:00.842021-05
session_id         | stress05-27944:0xc1a
client_label       |
transaction_start  | 2011-01-03 15:34:46.538102
transaction_id     | -1
transaction_description | user dbadmin (COPY Mart_Fact FROM
'/data/mart_Fact.tbl'
                                DELIMITER '|' NULL '\\\n';)
statement_start    | 2011-01-03 15:34:46.538862
statement_id       |
last_statement_duration_us | 26250
current_statement  | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER
'|'
```

```

                                NULL '\\n';
ssl_state                       | None
authentication_method          | Trust
-[ RECORD 3
]-----+-----
node_name                       | v_vmartdb_node0003
user_name                      | dbadmin
client_hostname                 | 127.0.0.1:56367
client_pid                     | 1191
login_timestamp                 | 2011-01-03 15:31:44.939302-05
session_id                     | stress06-25663:0xbec
client_label                    |
transaction_start              | 2011-01-03 15:34:51.05939
transaction_id                  | 54043195528458775
transaction_description         | user dbadmin (COPY Mart_Fact FROM
'/data/Mart_Fact.tbl'
                                DELIMITER '|' NULL '\\n' DIRECT;);
statement_start                 | 2011-01-03 15:35:46.436748
statement_id                    |
last_statement_duration_us     | 1591403
current_statement               | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER
'|'
                                NULL '\\n' DIRECT;
ssl_state                       | None
authentication_method          | Trust

```

**Close all sessions:**

```

vmartdb=> \x
Expanded display is off.
vmartdb=> SELECT CLOSE_ALL_SESSIONS();
                                CLOSE_ALL_SESSIONS

```

```

-----
Close all sessions command sent. Check v_monitor.sessions for progress.
(1 row)

```

**Sessions contents after issuing the CLOSE\_ALL\_SESSIONS() command:**

```

=> SELECT * FROM SESSIONS;
-[ RECORD 1 ]-----+-----
node_name                       | v_vmartdb_node0001
user_name                      | dbadmin
client_hostname                 | 127.0.0.1:52110
client_pid                     | 4554
login_timestamp                 | 2011-01-03 14:05:40.252625-05
session_id                     | stress04-4325:0x14
client_label                    |
transaction_start              | 2011-01-03 14:05:44.325781
transaction_id                  | 45035996273728326
transaction_description         | user dbadmin (SELECT * FROM sessions;)
statement_start                 | 2011-01-03 16:19:56.720071
statement_id                    | 25
last_statement_duration_us     | 15605
current_statement               | SELECT * FROM SESSIONS;
ssl_state                       | None

```

```
authentication_method | Trust
```

## Controlling Sessions

The database administrator must be able to disallow new incoming connections in order to shut down the database. On a busy system, database shutdown is prevented if new sessions connect after the `CLOSE_SESSION` or `CLOSE_ALL_SESSIONS()` command is invoked — and before the database actually shuts down.

One option is for the administrator to issue the `SHUTDOWN('true')` command, which forces the database to shut down and disallow new connections. See **SHUTDOWN** (page 469) in the SQL Reference Manual.

Another option is to modify the `MaxClientSessions` parameter from its original value to 0, in order to prevent new non-dbadmin users from connecting to the database.

- 1 Determine the original value for the `MaxClientSessions` parameter by querying the `V_MONITOR.CONFIGURATIONS_PARAMETERS` (page 832) system table:

```
=> SELECT CURRENT_VALUE FROM CONFIGURATION_PARAMETERS WHERE
       parameter_name='MaxClientSessions';
       CURRENT_VALUE
-----
          50
(1 row)
```

- 2 Set the `MaxClientSessions` parameter to 0 to prevent new non-dbadmin connections:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 0);
```

**Note:** The previous command allows up to five administrators to log in.

- 3 Issue the `CLOSE_ALL_SESSIONS()` command to remove existing sessions:

```
=> SELECT CLOSE_ALL_SESSIONS();
```

- 4 Query the `SESSIONS` table:

```
=> SELECT * FROM SESSIONS;
```

When the session no longer appears in the `SESSIONS` table, disconnect and run the Stop Database command.

- 5 Restart the database.

- 6 Restore the `MaxClientSessions` parameter to its original value:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 50);
```

### See Also

**CLOSE\_SESSION** (page 396)

**CONFIGURATION\_PARAMETERS** (page 832)

**SHUTDOWN** (page 469)

**V\_MONITOR.SESSIONS** (page 888)

Managing Sessions and Configuration Parameters in the Administrator's Guide

## Shutdown Problems in the Troubleshooting Guide

**CLOSE\_SESSION**

Interrupts the specified external session, rolls back the current transaction, if any, and closes the socket.

**Syntax**

```
CLOSE_SESSION ( 'sessionid' )
```

**Parameters**

<i>sessionid</i>	A string that specifies the session to close. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
------------------	--

**Notes**

- Closing of the session is processed asynchronously. It could take some time for the session to be closed. Check the **SESSIONS** (page 888) table for the status.
- Database shutdown is prevented if new sessions connect after the `CLOSE_SESSION()` command is invoked (and before the database is actually shut down. See **Controlling Sessions** below.

**Messages**

The following are the messages you could encounter:

- For a badly formatted sessionID  

```
close_session | Session close command sent. Check SESSIONS for progress.
Error: invalid Session ID format
```
- For an incorrect sessionID parameter  

```
Error: Invalid session ID or statement key
```

**Examples**

User session opened. RECORD 2 shows the user session running COPY DIRECT statement.

```
vmartdb=> SELECT * FROM sessions;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (SELECT * FROM sessions;)
statement_start    | 2011-01-03 15:36:13.896288
statement_id       | 10
last_statement_duration_us | 14978
```

```

current_statement      | select * from sessions;
ssl_state              | None
authentication_method | Trust
-[ RECORD 2 ]-----+-----
node_name              | v_vmartdb_node0002
user_name              | dbadmin
client_hostname       | 127.0.0.1:57174
client_pid            | 30117
login_timestamp       | 2011-01-03 15:33:00.842021-05
session_id            | stress05-27944:0xc1a
client_label          |
transaction_start     | 2011-01-03 15:34:46.538102
transaction_id        | -1
transaction_description | user dbadmin (COPY ClickStream_Fact FROM
                        '/data/clickstream/lg/ClickStream_Fact.tbl'
                        DELIMITER '|' NULL '\\n' DIRECT;)
statement_start       | 2011-01-03 15:34:46.538862
statement_id          |
last_statement_duration_us | 26250
current_statement     | COPY ClickStream_Fact FROM '/data/clickstream
                        /lg/ClickStream_Fact.tbl' DELIMITER '|' NULL
                        '\\n' DIRECT;
ssl_state              | None
authentication_method | Trust

```

### Close user session stress05-27944:0xc1a

```

vmartdb=> \x
Expanded display is off.
vmartdb=> SELECT CLOSE_SESSION('stress05-27944:0xc1a');
                        CLOSE_SESSION

```

```

-----
Session close command sent. Check v_monitor.sessions for progress.
(1 row)

```

Query the sessions table again for current status, and you can see that the second session has been closed:

```

=> SELECT * FROM SESSIONS;
-[ RECORD 1 ]-----+-----
node_name              | v_vmartdb_node0001
user_name              | dbadmin
client_hostname       | 127.0.0.1:52110
client_pid            | 4554
login_timestamp       | 2011-01-03 14:05:40.252625-05
session_id            | stress04-4325:0x14
client_label          |
transaction_start     | 2011-01-03 14:05:44.325781
transaction_id        | 45035996273728326
transaction_description | user dbadmin (select * from SESSIONS;)
statement_start       | 2011-01-03 16:12:07.841298
statement_id          | 20
last_statement_duration_us | 2099
current_statement     | SELECT * FROM SESSIONS;
ssl_state              | None
authentication_method | Trust

```

## Controlling Sessions

The database administrator must be able to disallow new incoming connections in order to shut down the database. On a busy system, database shutdown is prevented if new sessions connect after the `CLOSE_SESSION` or `CLOSE_ALL_SESSIONS()` command is invoked — and before the database actually shuts down.

One option is for the administrator to issue the `SHUTDOWN('true')` command, which forces the database to shut down and disallow new connections. See **SHUTDOWN** (page 469) in the SQL Reference Manual.

Another option is to modify the `MaxClientSessions` parameter from its original value to 0, in order to prevent new non-dbadmin users from connecting to the database.

- 1 Determine the original value for the `MaxClientSessions` parameter by querying the `V_MONITOR.CONFIGURATIONS_PARAMETERS` (page 832) system table:

```
=> SELECT CURRENT_VALUE FROM CONFIGURATION_PARAMETERS WHERE
      parameter_name='MaxClientSessions';
      CURRENT_VALUE
      -----
             50
      (1 row)
```

- 2 Set the `MaxClientSessions` parameter to 0 to prevent new non-dbadmin connections:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 0);
```

**Note:** The previous command allows up to five administrators to log in.

- 3 Issue the `CLOSE_ALL_SESSIONS()` command to remove existing sessions:

```
=> SELECT CLOSE_ALL_SESSIONS();
```

- 4 Query the `SESSIONS` table:

```
=> SELECT * FROM SESSIONS;
```

When the session no longer appears in the `SESSIONS` table, disconnect and run the Stop Database command.

- 5 Restart the database.

- 6 Restore the `MaxClientSessions` parameter to its original value:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 50);
```

### See Also

**CLOSE\_ALL\_SESSIONS** (page 399), **CONFIGURATION\_PARAMETERS** (page 832), **SESSIONS** (page 888), **SHUTDOWN** (page 469)

Managing Sessions and Configuration Parameters in the Administrator's Guide

Shutdown Problems in the Troubleshooting Guide

## GET\_NUM\_ACCEPTED\_ROWS

Returns the number of rows loaded into the database for the last completed load for the current session.

### Syntax

```
GET_NUM_ACCEPTED_ROWS ( ) ;
```

### Notes

- Only loads from STDIN or a single file on the initiator are supported. This function cannot be called for multi-node loads.
- Information is not available for a load that is currently running. Check the system table **LOAD\_STREAMS** (page 856) for its status.
- Data regarding loads does not persist, and is dropped when a new load is initiated.
- GET\_NUM\_ACCEPTED\_ROWS is a meta-function, Do not use it as a value in an INSERT query.

## GET\_NUM\_REJECTED\_ROWS

Returns the number of rows that were rejected during the last completed load for the current session.

### Syntax

```
GET_NUM_REJECTED_ROWS ( ) ;
```

### Notes

- Only loads from STDIN or a single file on the initiator are supported. This function cannot be called for multi-node loads.
- Information is not available for a load that is currently running. Check the system table **LOAD\_STREAMS** (page 856) for its status.
- Data regarding loads does not persist, and is dropped when a new load is initiated.
- GET\_NUM\_REJECTED\_ROWS is a meta-function, Do not use it as a value in an INSERT query.

## INTERRUPT\_STATEMENT

Interrupts the specified statement (within an external session), rolls back the current transaction, and writes a success or failure message to the log file.

### Syntax

```
INTERRUPT_STATEMENT ( session_id , statement_id )
```

### Parameters

<i>session_id</i>	Specifies the session to interrupt. This identifier is unique within the cluster at any point in time.
<i>statement_id</i>	Specifies the statement to interrupt

## Notes

- Only statements run by external sessions can be interrupted.
- Sessions can be interrupted during statement execution.
- If the *statement\_id* is valid, the statement is interruptible. The command is successfully sent and returns a success message. Otherwise the system returns an error.

## Messages

The following list describes messages you might encounter:

Message	Meaning
Statement interrupt sent. Check SESSIONS for progress.	This message indicates success.
Session <id> could not be successfully interrupted: session not found.	The session ID argument to the interrupt command does not match a running session.
Session <id> could not be successfully interrupted: statement not found.	The statement ID does not match (or no longer matches) the ID of a running statement (if any).
No interruptible statement running	The statement is DDL or otherwise non-interruptible.
Internal (system) sessions cannot be interrupted.	The session is internal, and only statements run by external sessions can be interrupted.

## Examples

Two user sessions are open. RECORD 1 shows user session running `SELECT FROM SESSION`, and RECORD 2 shows user session running `COPY DIRECT`:

```
=> SELECT * FROM SESSIONS;
-[ RECORD 1
]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (select * from sessions;)
statement_start    | 2011-01-03 15:36:13.896288
statement_id       | 10
last_statement_duration_us | 14978
current_statement  | select * from sessions;
ssl_state          | None
authentication_method | Trust
```

```

-[ RECORD 2
]-----+-----
node_name          | v_vmartdb_node0003
user_name          | dbadmin
client_hostname    | 127.0.0.1:56367
client_pid         | 1191
login_timestamp    | 2011-01-03 15:31:44.939302-05
session_id         | stress06-25663:0xbec
client_label       |
transaction_start  | 2011-01-03 15:34:51.05939
transaction_id     | 54043195528458775
transaction_description | user dbadmin (COPY Mart_Fact FROM
'/data/Mart_Fact.tbl'
                                DELIMITER '|' NULL '\\n' DIRECT;)
statement_start    | 2011-01-03 15:35:46.436748
statement_id       | 5
last_statement_duration_us | 1591403
current_statement  | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER
'|'
                                NULL '\\n' DIRECT;
ssl_state          | None
authentication_method | Trust

```

Interrupt the COPY DIRECT statement running in stress06-25663:0xbec:

```

=> \x
Expanded display is off.
=> SELECT INTERRUPT_STATEMENT('stress06-25663:0x1537', 5);
                                interrupt_statement

```

```

-----+-----
Statement interrupt sent. Check v_monitor.sessions for progress.
(1 row)

```

Verify that the interrupted statement is no longer active by looking at the current\_statement column in the SESSIONS system table. This column becomes blank when the statement has been interrupted:

```

=> SELECT * FROM SESSIONS;

```

```

-[ RECORD 1
]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (select * from sessions;)
statement_start    | 2011-01-03 15:36:13.896288
statement_id       | 10
last_statement_duration_us | 14978
current_statement  | select * from sessions;
ssl_state          | None
authentication_method | Trust

```

```

-[ RECORD 2
]-----+-----
node_name          | v_vmartdb_node0003
user_name          | dbadmin
client_hostname    | 127.0.0.1:56367
client_pid         | 1191
login_timestamp    | 2011-01-03 15:31:44.939302-05
session_id         | stress06-25663:0xbec
client_label       |
transaction_start  | 2011-01-03 15:34:51.05939
transaction_id     | 54043195528458775
transaction_description | user dbadmin (COPY Mart_Fact FROM
'/data/Mart_Fact.tbl'
                    DELIMITER '|' NULL '\\n' DIRECT;)
statement_start    | 2011-01-03 15:35:46.436748
statement_id       | 5
last_statement_duration_us | 1591403
current_statement |
ssl_state          | None
authentication_method | Trust

```

**See Also****SESSIONS** (page 888)

Managing Sessions and Configuration Parameters in the Administrator's Guide

**Statistic Management Functions**

This section contains statistic management functions specific to Vertica.

**ANALYZE\_HISTOGRAM**

Works much like **ANALYZE\_STATISTICS()** (page 385) in that it collects and aggregates data samples and storage information from all nodes on which a projection is stored, then writes statistics to the catalog so that the data can be used by the query optimizer to recommend the best possible plan. The only difference is that **ANALYZE\_HISTOGRAM** lets you specify how much of the disk should be read, allowing you to optimize either for speed or accuracy during histogram collection.

**Syntax**

```
ANALYZE_HISTOGRAM ( ( ' | '[ schema.]table' | 'column-name' ) [, ( percent ) ] )
```

**Return Value**

- 0 - For success.
- 1 - For failure. Refer to `vertica.log` for details.

**Parameters**

''	Empty string. Collects statistics for all tables.
[schema.]table	Specifies the name of the table and optional schema and collects statistics for all projections of the specified table.

	<b>Note:</b> When using more than one schema, specify the schema that contains the projection.
<i>column-name</i>	Specifies the name of a single table column, typically a predicate column, and collects statistics for that column only.
<i>percent</i>	(Default 10) Specifies what fraction of the data to read from disk with a range of 1 to 100. The smaller the percentage, the faster the analysis; however, because less data is collected, sampling accuracy will be affected. <b>Note:</b> If you specify the percent parameter as 100, the entire projection is read from disk, and 128K rows are chosen at random. Otherwise, (row count) * (percent/100) rows are read from the projection in 100 contiguous bands with a minimum of 128K rows read. The one exception is if the column is first in the chosen projection's sort order, then all data is read from disk.

### Notes

- `ANALYZE_HISTOGRAM()` is a DDL operation that auto commits the current transaction, if any.
- You can cancel `ANALYZE_HISTOGRAM` mid analysis by issuing CTRL-C in a vsql shell or by invoking the **`INTERRUPT_STATEMENT()`** (page 430) function.
- **`ANALYZE_STATISTICS()`** (page 385) is an alias for `ANALYZE_HISTOGRAM()`; the only difference is `ANALYZE_HISTOGRAM` includes the percent parameter.
- To retrieve hints about underperforming queries and their root causes, use the **`ANALYZE_WORKLOAD()`** (page 386) function, which runs the Workload Analyzer and analyzes the workload history, identifying underperforming queries and their root causes. The function's result set returns a set of recommendations; for example `"run analyze_statistics on schema.table.column"`.

### Examples

In the following command, the system reads 10% of data from disk (default) and returns 0 for success:

```
=> SELECT ANALYZE_STATISTICS('shipping_dimension.shipping_key');
ANALYZE_STATISTICS
-----
                                0
(1 row)
```

The next command performs the same functionality as the previous `ANALYZE_STATISTICS()` command and returns 0 for success:

```
=> SELECT ANALYZE_HISTOGRAM('shipping_dimension.shipping_key');
ANALYZE_HISTOGRAM
-----
                                0
(1 row)
```

With the percent parameter specified as 100, the following command performs a full column scan and returns 0 for success:

```
=> SELECT ANALYZE_HISTOGRAM('shipping_dimension.shipping_key', 100);
ANALYZE_HISTOGRAM
-----
                0
(1 row)
```

In this command, only 0.1% (1/1000) of the disk is read:

```
=> SELECT ANALYZE_HISTOGRAM('shipping_dimension.shipping_key', 0.1);
ANALYZE_HISTOGRAM
-----
                0
(1 row)
```

For use cases, see [Collecting Statistics in the Administrator's Guide](#)

**See Also**

***ANALYZE\_STATISTICS*** (page 385)

***ANALYZE\_WORKLOAD*** (page 386)

***DROP\_STATISTICS*** (page 410)

***EXPORT\_STATISTICS*** (page 423)

***IMPORT\_STATISTICS*** (page 433)

***INTERRUPT\_STATEMENT*** (page 430)

## ANALYZE\_STATISTICS

Collects and aggregates data samples and storage information from all nodes on which a projection is stored, then writes statistics into the catalog so that the data can be used by the query optimizer to recommend the best possible plan. Without statistics, the query optimizer would assume uniform distribution of data values and equal storage usage for all projections. `ANALYZE_STATISTICS()` is an alias for **`ANALYZE_HISTOGRAM()`** (page 382).

### Syntax

```
ANALYZE_STATISTICS [ ( ' ) | ( '[schema.]table' ) | ( 'column-name' ) ]
```

### Return Value

- 0 - For success.
- 1 - For failure. Refer to `vertica.log` for details.

### Parameters

<code>' '</code>	Empty string. Collects statistics for all tables.
<code>[schema.]table</code>	Specifies the name of the table and optional schema and collects statistics for all projections of the specified table. <b>Note:</b> When using more than one schema, specify the schema that contains the projection.
<code>column-name</code>	Specifies the name of a single table column, typically a predicate column. and collects statistics for only that column.

### Notes

- `ANALYZE_STATISTICS` is a DDL operation that auto commits the current transaction, if any.
- Pre-join projection statistics are updated on any pre-joined tables.
- You can cancel `ANALYZE_STATISTICS` mid analysis by issuing CTRL-C in a vsql shell or by invoking the **`INTERRUPT_STATEMENT()`** (page 430) function.
- Always run `ANALYZE_STATISTICS` on a table rather than a projection.
- Invoking `ANALYZE_STATISTICS()` against very large tables with many projections could return results more slowly. To expedite things, you can run `ANALYZE_STATISTICS()` on a single column. For example, it is a good idea to run statistics on a single column after you alter a table to add or drop a column.
- By default, more than one column is analyzed in a single query execution plan (subject to resource limits) in order to reduce the latency (time delay) of plan execution and help speed up analysis of relatively small tables with a large number of columns.
- To retrieve hints about underperforming queries and their root causes, use the **`ANALYZE_WORKLOAD()`** (page 386) function, which runs the Workload Analyzer and analyzes the workload history, identifying underperforming queries and their root causes. The function's result set returns a set of recommendations; for example `run analyze_statistics on schema.table.column`. These tuning recommendations are provided so the database administrator can act upon them.

## Examples

The following command computes statistics on all projections in the Vmart database and returns 0 (success):

```
=> SELECT ANALYZE_STATISTICS ('');
analyze_statistics
-----
0
(1 row)
```

The following command computes statistics on a single table (`shipping_dimension`) and returns 0 (success):

```
=> SELECT ANALYZE_STATISTICS ('shipping_dimension');
analyze_statistics
-----
0
(1 row)
```

The following command computes statistics on a single important column (`shipping_key`) for all projections in the `shipping_dimension` table and returns 0 (success):

```
=> SELECT ANALYZE_STATISTICS('shipping_dimension.shipping_key');
analyze_statistics
-----
0
(1 row)
```

For use cases, see [Collecting Statistics in the Administrator's Guide](#)

## See Also

***ANALYZE\_WORKLOAD*** (page 386)

***DROP\_STATISTICS*** (page 410)

***EXPORT\_STATISTICS*** (page 423)

***IMPORT\_STATISTICS*** (page 433)

***INTERRUPT\_STATEMENT*** (page 430)

## DROP\_STATISTICS

Removes statistics for the specified table or type.

### Syntax

```
DROP_STATISTICS('[ schema.]table' [, {'BASE' | 'HISTOGRAMS' | 'EXTENDED' | 'ALL'}
]);
```

### Return Value

- 0 - For success.
- 1 - For failure. Refer to *vertica.log* for details about the failure.

## Parameters

<code>''</code>	Empty string. Drops statistics for all projections.
<code>[schema.]table</code>	Drops statistics for all projections within the specified table. When using more than one schema, specify the schema that contains the table with the projections you want to delete.
<code>projection</code>	Drops statistics for the specified projection.
<code>TYPE</code>	Specifies the type of statistics to drop: <ul style="list-style-type: none"> <li>▪ 'BASE' (default) drops histograms and row counts (min/max column values, histogram, etc.. Extended statistics remain.</li> <li>▪ 'HISTOGRAMS' drops only the histograms. Row counts and extended statistics remain.</li> <li>▪ 'EXTENDED' drops soft unique keys. Histograms and row counts remain.</li> <li>▪ 'ALL' drops all types of statistics.</li> </ul>

## Notes

Once dropped, statistics can be time consuming to regenerate.

## Example

The following command analyzes all statistics on the VMart schema database:

```
vmart=> SELECT ANALYZE_STATISTICS('');
ANALYZE_STATISTICS
-----
0
(1 row)
```

This command drops base statistics for table `store_sales_fact` in the `store` schema:

```
vmart=> SELECT DROP_STATISTICS('store.store_sales_fact', 'BASE');
drop_statistics
-----
0
(1 row)
```

Note that this command works the same as the previous command:

```
vmart=> SELECT DROP_STATISTICS('store.store_sales_fact');
DROP_STATISTICS
-----
0
(1 row)
```

The following example drops statistics for all projections in the database and returns 0 (success):

```
vmart=> SELECT DROP_STATISTICS ('', 'ALL');
DROP_STATISTICS
-----
0
(1 row)
```

This command also drops statistics for all table projections:

```
vmart=> SELECT DROP_STATISTICS ('');
        DROP_STATISTICS
-----
                0
(1 row)
```

The following command drops extended statistics for the shipping\_dimension table and returns 0 (success):

```
vmart=> SELECT DROP_STATISTICS ('shipping_dimension', 'EXTENDED');
        DROP_STATISTICS
-----
                0
(1 row)
```

The following command drops statistics for one of the shipping\_dimension table's projections and returns 0 (success):

```
vmart=> SELECT DROP_STATISTICS('shipping_dimension',
        'shipping_dimension_node02');
        DROP_STATISTICS
-----
                0
(1 row)
```

For use cases, see Collecting Statistics in the Administrator's Guide

### See Also

**ANALYZE\_STATISTICS** (page 385)

**EXPORT\_STATISTICS** (page 423)

**IMPORT\_STATISTICS** (page 433)

### EXPORT\_STATISTICS

Generates an XML file that contains statistics for the database.

**Note:** Before you export statistics for the database, run **ANALYZE\_STATISTICS()** (page 385) to collect and aggregate data samples and storage information. If you do not use **ANALYZE\_STATISTICS**, the Database Designer produces a suboptimal projection similar to projections created for temporary designs.

### Syntax

```
EXPORT_STATISTICS ( 'destination' )
```

### Parameters

<i>destination</i>	Specifies the path and name of the XML output file. An empty string returns the script to standard output.
--------------------	--

### Example

The following command exports statistics on the VMart example database to a file:

```
vmart=> SELECT EXPORT_STATISTICS('/vmart/statistics/vmart_stats');
        export_statistics
```

```
-----
Statistics exported successfully
(1 row)
```

### See Also

**ANALYZE\_STATISTICS** (page 385)

**DROP\_STATISTICS** (page 410)

**IMPORT\_STATISTICS** (page 433)

Collecting Statistics in the Administrator's Guide

## IMPORT\_STATISTICS

Imports statistics from the XML file generated by the EXPORT\_STATISTICS command.

### Syntax

```
IMPORT_STATISTICS ( 'destination' )
```

### Parameters

<i>destination</i>	Specifies the path and name of the XML input file (which is the output of EXPORT_STATISTICS function).
--------------------	--

### Notes

- Imported statistics override existing statistics for all projections on the specified table.
- For use cases, see Collecting Statistics in the Administrator's Guide

### See Also

**ANALYZE\_STATISTICS** (page 385)

**DROP\_STATISTICS** (page 410)

**EXPORT\_STATISTICS** (page 423)

## Storage Management Functions

This section contains storage management functions specific to Vertica.

### ADD\_LOCATION

Adds a location to store data.

### Syntax

```
ADD_LOCATION ( 'path' , [ 'node' , 'usage_string' ] )
```

### Parameters

<i>path</i>	Specifies where the storage location is mounted. Path must be an empty directory with write permissions for user,
-------------	--

	group, or all.
<i>node</i>	Is the Vertica node where the location is available. If this parameter is omitted, <i>node</i> defaults to the initiator.
<i>usage_string</i>	Is one of the following: <ul style="list-style-type: none"> <li>▪ DATA: Only data is stored in the location.</li> <li>▪ TEMP: Only temporary files that are created during loads or queries are stored in the location.</li> <li>▪ DATA,TEMP: Both types of files are stored in the location.</li> </ul> If this parameter is omitted, the default is DATA,TEMP.

### Notes

- By default, the location is used to store both data and temporary files.
- Locations can be added from any node to any node.
- Either *node* and *usage\_string* must both be specified or neither of them specified.
- Information about storage locations is visible **V\_MONITOR.DISK\_STORAGE** (page 844).
- A storage location annotation called CATALOG indicates the location is used to store the catalog and is visible in V\_MONITOR.DISK\_STORAGE. However, no new locations can be added, as CATALOG locations and existing CATALOG annotations cannot be removed.

### Example

This example adds a location that stores data and temporary files:

```
SELECT ADD_LOCATION('/secondVerticaStorageLocation/');
```

This example adds a location to store data only:

```
SELECT ADD_LOCATION('/secondVerticaStorageLocation/' , 'node2' , 'DATA');
```

### See Also

**ALTER\_LOCATION\_USE** (page 376)

**RETIRE\_LOCATION** (page 461)

### ALTER\_LOCATION\_USE

Alters the type of files stored in the specified storage location.

### Syntax

```
ALTER_LOCATION_USE ( 'path' , [ 'node' ] , 'usage_string' )
```

### Parameters

<i>path</i>	Specifies where the storage location is mounted.
<i>node</i>	[Optional] Is the Vertica node where the location is available. If this parameter is omitted, <i>node</i> defaults to the initiator.

<i>usage_string</i>	<p>Is one of the following:</p> <ul style="list-style-type: none"> <li>▪ DATA: Only data is stored in the location.</li> <li>▪ TEMP: Only temporary files that are created during loads or queries are stored in the location.</li> <li>▪ DATA,TEMP: Both types of files are stored in the location.</li> </ul>
---------------------	---

## Notes

- Altering the type of files stored in a particular location is useful if you create additional storage locations and you want to isolate execution engine temporary files from data files.
- After modifying the location's use, at least one location must remain for storing data and temp files. These files can be stored in the same storage location or separate storage locations.
- When a storage location is altered, it stores only the type of information indicated from that point forward. For example:
  - If you modify a storage location that previously stored both temp and data files so that it only stores temp files, the data is eventually merged out through the ATM. You can also merge it out manually.
  - If you modify a storage location that previously stored both temp and data files so that it only stores data files, all currently running statements that use these temp files, such as queries and loads, continue to run. Subsequent statements will no longer use this location.

## Example

The following example alters the storage location on node3 to store data only:

```
=> SELECT ALTER_LOCATION_USE ('/thirdVerticaStorageLocation/' , 'node3' ,
'DATA');
```

## See Also

**ADD\_LOCATION** (page 374)

**RETIRE\_LOCATION** (page 461)

Modifying Storage Locations in the Administrator's Guide

## DROP\_LOCATION

Removes the specified storage location.

## Syntax

```
DROP_LOCATION ( 'path' , 'site' )
```

## Parameters

<i>path</i>	Specifies where the storage location to drop is mounted.
<i>site</i>	Is the Vertica site where the location is available.

## Notes

- Dropping a storage location is a permanent operation and cannot be undone. Therefore, Vertica recommends that you retire a storage location before dropping it. This allows you to verify that you actually want to drop a storage location before doing so. Additionally, you can easily restore a retired storage location.
- Dropping storage locations is limited to locations that contain only temp files.
- If a location used to store data and you modified it to store only temp files, the location might still contain data files. If the storage location contains data files, Vertica does not allow you to drop it. You can manually merge out all the data in this location, wait for the ATM to merge out the data files automatically, or you can drop partitions. Deleting data files does not work.

## Example

The following example drops a storage location on node3 that was used to store temp files:

```
=> SELECT DROP_LOCATION('/secondVerticaStorageLocation/' , 'node3');
```

## See Also

- **RETIRE\_LOCATION** (page 461) in this SQL Reference Manual
- Dropping Storage Locations and Retiring Storage Locations in the Administrator's Guide

## MEASURE\_LOCATION\_PERFORMANCE

Measures disk performance for the location specified.

## Syntax

```
MEASURE_LOCATION_PERFORMANCE ( 'path' , 'node' )
```

## Parameters

<i>path</i>	Specifies where the storage location to measure is mounted.
<i>node</i>	Is the Vertica node where the location to be measured is available..

## Notes

- If you intend to create a tiered disk architecture in which projections, columns, and partitions are stored on different disks based on predicted or measured access patterns, you need to measure storage location performance for each location in which data is stored. You do not need to measure storage location performance for temp data storage locations because temporary files are stored based on available space.
- This method of measuring storage location performance applies only to configured clusters. If you want to measure a disk before configuring a cluster see Measuring Location Performance.
- Storage location performance equates to the amount of time it takes to read and write 1 MB of data from the disk. This time equates to:

IO time = Time to read/write 1MB + Time to seek = 1/Throughput + 1/Latency

Throughput is the average throughput of sequential reads/writes (units in MB per second)

Latency is for random reads only in seeks (units in seeks per second)

**Note:** The IO time of a faster storage location is less than a slower storage location.

### Example

The following example measures the performance of a storage location on node2:

```
=> SELECT MEASURE_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/' ,
'node2');
```

WARNING: measure\_location\_performance can take a long time. Please check logs for progress

```
measure_location_performance
```

```
-----
Throughput : 122 MB/sec. Latency : 140 seeks/sec
```

### See Also

**ADD\_LOCATION** (page 374)

**ALTER\_LOCATION\_USE** (page 376)

**RETIRE\_LOCATION** (page 461)

Measuring Location Performance in the Administrator's Guide

## RESTORE\_LOCATION

Restores the retired location specified.

### Syntax

```
RESTORE_LOCATION ( 'path' , 'node' )
```

### Parameters

<i>path</i>	Specifies where the retired storage location is mounted.
<i>node</i>	Is the Vertica node where the retired location is available.

### Notes

Once restored, Vertica re-ranks the storage locations and use the restored location to process queries as determined by its rank.

### Example

The following example restores the retired storage location on node3:

```
=> SELECT RESTORE_LOCATION ('/thirdVerticaStorageLocation/' , 'node3');
```

### See Also

**ADD\_LOCATION** (page 374)

**RETIRE\_LOCATION** (page 461)

Modifying Storage Locations in the Administrator's Guide

## RETIRE\_LOCATION

Makes the specified storage location inactive.

### Syntax

```
RETIRE_LOCATION ( 'path' , 'node' )
```

### Parameters

<i>path</i>	Specifies where the storage location to retire is mounted.
<i>node</i>	Is the Vertica node where the location is available.

### Notes

- Before retiring a location, be sure that at least one location remains for storing data and temp files. Data and temp files can be stored in either one storage location or separate storage locations.
- Once retired, no new data can be stored on the location unless the location is restored through the **RESTORE\_LOCATION()** (page 461) function.
- If the storage location stored data, the data is not moved. Instead, it is removed through one or more mergeouts. Therefore, the location cannot be dropped.
- If the storage site was used to store only temp files, it can be dropped. See Dropping Storage Locations in the Administrators Guide and the **DROP\_LOCATION()** (page 407) function.

### Example

```
=> SELECT RETIRE_LOCATION ('/secondVerticaStorageLocation/' , 'node2');
```

### See Also

**ADD\_LOCATION** (page 374)

**RESTORE\_LOCATION** (page 461)

Retiring Storage Locations in the Administrator's Guide

## SET\_LOCATION\_PERFORMANCE

Sets disk performance for the location specified.

### Syntax

```
SET_LOCATION_PERFORMANCE ( 'path' , 'node' , 'throughput' , 'average_latency' )
```

### Parameters

<i>node</i>	Is the Vertica node where the location to be set is available. If this parameter is omitted, <i>node</i> defaults to the initiator.
<i>path</i>	Specifies where the storage location to set is mounted.

<i>throughput</i>	Specifies the throughput for the location, which must be 1 or more.
<i>average_latency</i>	Specifies the average latency for the location. The <i>average_latency</i> must be 1 or more.

### Notes

To obtain the throughput and average latency for the location, run the ***MEASURE\_LOCATION\_PERFORMANCE()*** (page 438) function before you attempt to set the location's performance.

### Example

The following example sets the performance of a storage location on node2 to a throughput of 122 megabytes per second and a latency of 140 seeks per second.

```
=> SELECT MEASURE_LOCATION_PERFORMANCE('node2','/secondVerticaStorageLocation/','122','140');
```

### See Also

***ADD\_LOCATION*** (page 374)

***MEASURE\_LOCATION\_PERFORMANCE*** (page 438)

Measuring Location Performance and Setting Location Performance in the Administrator's Guide

## Timeseries Functions

Timeseries aggregate functions evaluate the values of a given set of variables over time and group those values into a window for analysis and aggregation.

One output row is produced per time slice—or per partition per time slice—if partition expressions are present.

### See Also

***TIMESERIES Clause*** (page 761)

***CONDITIONAL\_CHANGE\_EVENT*** (page 137) and ***CONDITIONAL\_TRUE\_EVENT*** (page 138)

Using Time Series Analytics in the Programmer's Guide

### TS\_FIRST\_VALUE

Processes the data that belongs to each time slice. A time series aggregate (TSA) function, ***TS\_FIRST\_VALUE*** returns the value at the start of the time slice, where an interpolation scheme is applied if the timeslice is missing, in which case the value is determined by the values corresponding to the previous (and next) timeslices based on the interpolation scheme of const (linear). There is one value per time slice per partition.

### Behavior Type

Immutable

## Syntax

```
TS_FIRST_VALUE ( expression [ IGNORE NULLS ]
... [, { 'CONST' | 'LINEAR' } ] )
```

## Parameters

<i>expression</i>	Is the argument expression on which to aggregate and interpolate. <i>expression</i> is data type INTEGER or FLOAT.
IGNORE NULLS	The IGNORE NULLS behavior changes depending on a CONST or LINEAR interpolation scheme. See When Time Series Data Contains Nulls in the Programmer's Guide for details.
'CONST'   'LINEAR'	Optionally specifies the interpolation value as either constant or linear. The default is constant. If omitted, Vertica defaults to CONST, but you can also specify CONST.

## Notes

- The function returns one output row per time slice or one output row per partition per time slice if partition expressions are specified.
- Multiple time series aggregate functions can exist in the same query. They share the same gap-filling policy as defined by the **TIMESERIES clause** (page 761); however, each time series aggregate function can specify its own interpolation policy. For example:

```
SELECT slice_time, symbol,
       TS_FIRST_VALUE (bid, 'const') fv_c,
       TS_FIRST_VALUE (bid, 'linear') fv_l,
       TS_LAST_VALUE (bid, 'const') lv_c
FROM TickStore
TIMESERIES slice_time AS '3 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```

- You must use an ORDER BY clause with a timestamp column.

## Example

For detailed examples, see Gap Filling and Interpolation and When Time Series Data Contains Nulls in the Programmer's Guide.

## See Also

**TIMESERIES Clause** (page 761) and **TS\_LAST\_VALUE** (page 472)

Using Time Series Analytics in the Programmer's Guide

## TS\_LAST\_VALUE

Processes the data that belongs to each time slice. A time series aggregate (TSA) function, **TS\_LAST\_VALUE** returns the value at the end of the time slice, where an interpolation scheme is applied if the timeslice is missing, in which case the value is determined by the values corresponding to the previous (and next) timeslices based on the interpolation scheme of const (linear). There is one value per time slice per partition.

## Behavior Type

Immutable

## Syntax

```
TS_LAST_VALUE ( expression [ IGNORE NULLS ]
... [, { 'CONST' | 'LINEAR' } ] )
```

## Parameters

<i>expression</i>	Is the argument expression on which to aggregate and interpolate. <i>expression</i> is data type INTEGER or FLOAT.
IGNORE NULLS	The IGNORE NULLS behavior changes depending on a CONST or LINEAR interpolation scheme. See When Time Series Data Contains Nulls in the Programmer's Guide for details.
'CONST'   'LINEAR'	Optionally specifies the interpolation value as either constant or linear. The default is constant. If omitted, Vertica defaults to CONST, but you can also specify CONST.

## Notes

- The function returns one output row per time slice or one output row per partition per time slice if partition expressions are specified.
- Multiple time series aggregate functions can exist in the same query. They share the same gap-filling policy as defined by the **TIMESERIES clause** (page 761); however, each time series aggregate function can specify its own interpolation policy. For example:

```
SELECT slice_time, symbol,
       TS_FIRST_VALUE (bid, 'const') fv_c,
       TS_FIRST_VALUE (bid, 'linear') fv_l,
       TS_LAST_VALUE (bid, 'const') lv_c
FROM TickStore
TIMESERIES slice_time AS 3 seconds OVER (PARTITION BY symbol ORDER BY ts);
```

- If you use the `window_order_clause`, you can order by a **TIMESTAMP** column only, not, for example, an **INTEGER** column.

## Example

For detailed examples, see Gap Filling and Interpolation and When Time Series Data Contains Nulls in the Programmer's Guide.

## See Also

**TIMESERIES Clause** (page 761) and **TS\_FIRST\_VALUE** (page 471)

Using Time Series Analytics in the Programmer's Guide

## Tuple Mover Functions

This section contains tuple mover functions specific to Vertica.

### DO\_TM\_TASK

Runs a Tuple Mover operation (moveout) on one or more projections defined on the specified table. You do not need to stop the Tuple Mover to run this function.

#### Syntax

```
DO_TM_TASK ( 'task' [ , '[ schema.]table' | 'projection' ] )
```

#### Parameters

<i>task</i>	Is one of the following tuple mover operations: <ul style="list-style-type: none"> <li>'moveout' — Moves out all projections on the specified table (if a particular projection is not specified).</li> <li>'analyze_row_count' — Automatically collects the number of rows in a projection every 60 seconds and aggregates row counts calculated during loads.</li> </ul>
<i>[schema.]table</i>	Runs a tuple mover operation for all projections within the specified table. When using more than one schema, specify the schema that contains the table with the projections you want to affect.
<i>projection</i>	If <i>projection</i> is not passed as an argument, all projections in the system are used. If <i>projection</i> is specified, DO_TM_TASK looks for a projection of that name and, if found, uses it; if a named projection is not found, the function looks for a table with that name and, if found, moves out all projections on that table.

#### Notes

DO\_TM\_TASK() is useful because you can move out all projections from a table or database without having to name each projection individually.

#### Examples

The following example performs a moveout of all projections for table t1:

```
=> SELECT DO_TM_TASK('moveout', 't1');
```

The following example performs a moveout for projections t1\_p:

```
=> SELECT DO_TM_TASK('moveout', 't1_p')
```

#### See Also

**COLUMN\_STORAGE** (page 830)

**DROP\_PARTITION** (page 408)

**DUMP\_PARTITION\_KEYS** (page 414)

***DUMP\_PROJECTION\_PARTITION\_KEYS*** (page 415)

***DUMP\_TABLE\_PARTITION\_KEYS*** (page 416)

***PARTITION\_PROJECTION*** (page 440)

Partitioning Tables in the Administrator's Guide

Collecting Statistics in the Administrator's Guide

## Workload Management Functions

This section contains workload management functions specific to Vertica.

### ANALYZE\_WORKLOAD

Runs the Workload Analyzer, a utility that analyzes the system workload history, resources, and configurations to identify underperforming queries, as well as the root causes for poor query performance. The function returns a set of tuning recommendations for all events in 'scope' since the time specified. If you don't specify a time, the function returns recommendations on events since the last recorded invocation of ANALYZE\_WORKLOAD(). The tuning recommendations are provided so the database administrator can act upon them.

#### Syntax 1

```
ANALYZE_WORKLOAD ( 'scope' , [ true ] );
```

#### Syntax 2

```
ANALYZE_WORKLOAD ( 'scope' , 'since_time' );
```

#### Parameters

<i>scope</i>	Specifies which catalog objects to analyze. Can be one of: <ul style="list-style-type: none"> <li>▪ " (empty string) returns recommendations for all objects in database</li> <li>▪ 'table_name' returns all recommendations related to a table</li> <li>▪ 'schema_name' returns recommendations on all objects in a schema</li> </ul>
<i>since_time</i>	Limits the recommendations from all events of 'scope' since the specified time up to the current system status. When omitted, the function returns the recommendations on events since the last recorded invocation of ANALYZE_WORKLOAD. <b>Note:</b> Strings used for this parameter must be explicitly cast to <code>TIMESTAMP</code> or <code>TIMESTAMP TZ</code> . See the Examples section below.
<i>true</i>	Specifies that this analysis invocation will be recorded in the system. The default value is false (do not record in system). If recorded, subsequent calls to ANALYZE_WORKLOAD analyze only the events that have occurred since this recorded time; all prior events are ignored.

## Return Value

ANALYZE\_WORKLOAD() returns aggregated tuning recommendations, as described in the following table. See the Notes section below this table for example output for the parameters:

Column	Data type	Description
<i>observation_count</i>	INTEGER	An integer for the total number of events observed for this tuning recommendation.
<i>first_observation_time</i>	TIMESTAMP TZ	The timestamp for when the event first occurred. If null, the tuning recommendation is from the current status of the system instead of from any event.
<i>last_observation_time</i>	TIMESTAMP TZ	The timestamp for when the event last occurred. If null, the tuning recommendation is from the current status of the system instead of from any event.
<i>tuning_parameter</i>	VARCHAR	The objects on which tuning action will be applied.
<i>tuning_description</i>	VARCHAR	Returns the tuning recommendation from the Workload Analyzer.
<i>tuning_command</i>	VARCHAR	Returns a command string if tuning action is a SQL command.
<i>tuning_cost</i>	VARCHAR	The cost of performing the recommended tuning operation. Is one of: <ul style="list-style-type: none"> <li>▪ LOW</li> <li>▪ MEDIUM</li> <li>▪ HIGH.</li> </ul>

## Notes

- If the 'scope' you enter is invalid, the system returns an error:  
ERROR: Invalid scope in ANALYZE\_WORKLOAD: schema or table my\_table does not exist
- An example return value for *tuning\_parameter* is `schema.table.column`; for example:  
`store.store_orders_fact.date_ordered`  
The above identifies the `date_ordered` column as the column that needs tuning. This is useful because it means the DBA can run ANALYZE\_STATISTICS on a single column, rather than on an entire table.
- Example values returned for the *tuning\_description* column are:
  - Run database designer on table `public.t`
  - Create replicated projection for table `public.vendor_dimension`
  - Consider query-specific design on query
  - Reset configuration parameter with `set_config_parameter()`
  - Re-segment projection `public.p_t` on high-cardinality column(s)

- Example return values for the `tuning_command` column are listed below. The first statement tells the user to update statistics on a particular `schema.table.column`:

```
SELECT ANALYZE_STATISTICS('public.tablename.column-name');
```

The next statement indicates a mismatched configuration parameter 'LockTimeout':

```
SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name =
'LockTimeout';
```

## Examples

The following statement returns recommendations for all objects in all schemas. Analysis is based on the events since the last recorded analysis:

```
=> SELECT ANALYZE_WORKLOAD('', true);
```

The `true` setting tells the Workload Analyzer to record this analysis invocation. The next invocation of `ANALYZE_WORKLOAD()` analyzes events from this point forward.

The next statement instructs the Workload Analyzer to save all events for table `t1`:

```
=> \x
Expanded display is on.
=> SELECT ANALYZE_WORKLOAD('t1');
```

```
-[ RECORD 1 ]-----+-----
observation_count      | 1
first_observation_time |
last_observation_time  |
tuning_parameter       | public.t1
tuning_description     | run database designer on table public.t1
tuning_command         |
tuning_cost            | HIGH
```

**Note:** Null observation times shown above indicate that the tuning recommendation is from the current status of the system instead of from any event.

The recommendation output can also be queried using the ***V\_MONITOR.TUNING\_RECOMMENDATIONS*** (page 899) system table:

```
=> SELECT * FROM tuning_recommendations;
```

```
-[ RECORD 1 ]-----+-----
observation_count      | 1
first_observation_time |
last_observation_time  |
tuning_parameter       | public.T1
tuning_description     | run database designer on table public.T1
tuning_command         |
tuning_cost            | HIGH
```

This statement analyzes workloads on all tables one week before today (now):

```
=> SELECT ANALYZE_WORKLOAD('', NOW() - INTERVAL '1 week');
```

```
-[ RECORD 1
]-----+-----
observation_count      | 4
```

```

first_observation_time | 2011-04-07 13:57:17.799003-04
last_observation_time  | 2011-04-12 12:05:26.856456-04
tuning_parameter       | store.store_orders_fact.date_ordered
tuning_description     | analyze statistics on table column
store.store_orders_fact.date_ordered
tuning_command         | select
analyze_statistics('store.store_orders_fact.date_ordered');
tuning_cost            | MEDIUM
-[ RECORD 14
]-----+-----
observation_count      | 2
first_observation_time | 2011-04-12 17:52:03.022644-04
last_observation_time  | 2011-04-12 17:52:03.02301-04
tuning_parameter       | SELECT x FROM t WHERE x > (SELECT SUM(DISTINCT x) FROM
                        | t GROUP BY y) OR x < 9;
tuning_description     | consider query-specific design on query
tuning_command         |
tuning_cost            | HIGH

```

Note that the Workload Analyzer found two issues in the above results:

- In RECORD 1 the date\_ordered column in the store.store\_orders\_fact table. The Workload Analyzer recommends that the DBA run ANALYZE\_STATISTICS() on that column. The function also returns the query the DBA can run, which is output in the tuning\_command column.
- In RECORD 2 there is an under performing query. The Workload Analyzer recommends that the DBA consider using the Database Designer to run a query-specific design on that particular query. The cost is HIGH, which lets the DBD know he or she might want to let the query run after hours.

The following four statements are some of the different ways to express the same query. All return the same results for workloads on table t1 since October 4, 2010.

```

=> SELECT ANALYZE_WORKLOAD('t1', TIMESTAMP '2010-10-04 11:18:15');
=> SELECT ANALYZE_WORKLOAD('t1', '2010-10-04 11:18:15'::TIMESTAMPTZ);
=> SELECT ANALYZE_WORKLOAD('t1', 'October 4, 2010'::TIMESTAMP);
=> SELECT ANALYZE_WORKLOAD('t1', '10-04-10'::TIMESTAMPTZ);

```

This statement analyzes workloads on all tables two days before today (now):

```

=> SELECT ANALYZE_WORKLOAD('', NOW() - '2 DAYS'::INTERVAL);

```

The results above indicate an issue with the customer\_state column in the customer\_dimension table and recommends that the DBA run ANALYZE\_STATISTICS() on that column.

```

-[ RECORD 1
]-----+-----
observation_count      | 2
first_observation_time | 2011-04-12 11:13:57.568675-04
last_observation_time  | 2011-04-12 12:05:08.200439-04
tuning_parameter       | public.customer_dimension.customer_state
tuning_description     | analyze statistics on table column public.customer_dimen
                        | sion.customer_state
tuning_command         | select
analyze_statistics('public.customer_dimension.cu
                        | stomer_state');

```

tuning\_cost | MEDIUM

**See Also**

***V\_MONITOR.TUNING\_RECOMMENDATIONS*** (page 899)

Analyzing Workloads in the Administrator's Guide

# SQL Statements

---

The primary structure of a SQL query is its statement. Multiple statements are separated by semicolons; for example:

```
CREATE TABLE fact ( ..., date_col date NOT NULL, ...);
CREATE TABLE fact(..., state VARCHAR NOT NULL, ...);
```

## ALTER FUNCTION

Alters a SQL Macro by providing a new function or different schema name.

### Syntax 1

```
ALTER FUNCTION
... [ schema_name.]function-name ( [ [ argname ] argtype [, ...] ] )
... RENAME TO new_name
```

### Syntax 2

```
ALTER FUNCTION
... [ schema_name.]function-name ( [ [ argname ] argtype [, ...] ] )
... SET SCHEMA new_schema
```

### Parameters

<code>[schema-name.]function-name</code>	Specifies a name for the SQL Macro (function body) to alter. If the function name is schema-qualified, the function is altered in the specified schema.
<code>argname</code>	Specifies the name of the argument.
<code>argtype</code>	Specifies the data type for argument that is passed to the function. Argument types must match Vertica type names. See <b>SQL Data Types</b> (page 64).
<code>new_name</code>	Specifies the new name of the function
<code>new_schema</code>	Specifies the new schema name where the function resides.

### Notes

Before you can alter a function, you must specify the argument type because there could be several functions that share the same name with different argument types.

### Permissions

Only the superuser or owner can alter the function.

### Example

This example creates a SQL Macro called `zeroifnull` that accepts an `INTEGER` argument and returns an `INTEGER` result.

```
=> CREATE FUNCTION zeroifnull(x INT) RETURN INT
    AS BEGIN
        RETURN (CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END);
    END;
```

This next command renames the `zeroifnull` function to `zerowhennull`:

```
=> ALTER FUNCTION zeroifnull(x INT) RENAME TO zerowhennull;
ALTER FUNCTION
```

This command moves the renamed function to a new schema called `macros`:

```
=> ALTER FUNCTION zerowhennull(x INT) SET SCHEMA macros;
ALTER FUNCTION
```

### See Also

***CREATE FUNCTION*** (page 629)

***DROP FUNCTION*** (page 700)

***GRANT (Function)*** (page 721)

***REVOKE (Function)*** (page 735)

***V\_CATALOG.USER\_FUNCTIONS*** (page 822)

Using SQL Macros in the Programmer's Guide

## ALTER PROJECTION RENAME

Initiates a rename operation on the specified projection:

### Syntax

```
ALTER PROJECTION projection-name RENAME TO new-projection-name
```

### Parameters

<i>projection-name</i>	Specifies the projection to change.
<i>new-projection-name</i>	Specifies the new projection name.

### Notes

The projection must exist before it can be renamed.

## ALTER PROFILE

Changes a profile. Only the database superuser can alter a profile.

### Syntax

```
ALTER PROFILE name LIMIT
... [PASSWORD_LIFE_TIME {life-limit | DEFAULT | UNLIMITED}]
... [PASSWORD_GRACE_TIME {grace-period | DEFAULT | UNLIMITED}]
... [FAILED_LOGIN_ATTEMPTS {login-limit | DEFAULT | UNLIMITED}]
... [PASSWORD_LOCK_TIME {lock-period | DEFAULT | UNLIMITED}]
... [PASSWORD_REUSE_MAX {reuse-limit | DEFAULT | UNLIMITED}]
... [PASSWORD_REUSE_TIME {reuse-period | DEFAULT | UNLIMITED}]
... [PASSWORD_MAX_LENGTH {max-length | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LENGTH {min-length | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LETTERS {min-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_UPPERCASE_LETTERS {min-cap-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LOWERCASE_LETTERS {min-lower-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_DIGITS {min-digits | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_SYMBOLS {min-symbols | DEFAULT | UNLIMITED}]
```

**Note:** For all parameters, the special value DEFAULT means the parameter is inherited from the DEFAULT profile.

### Parameters

Parameter Name	Description	Meaning of UNLIMITED value
<i>name</i>	The name of the profile to create	N/A
PASSWORD_LIFE_TIME <i>life-limit</i>	Integer number of days a password remains valid. After the time elapses, the user must change the password (or will be warned that their password has	Passwords never expire.

	expired if PASSWORD_GRACE_TIME is set to a value other than zero or UNLIMITED).	
PASSWORD_GRACE_TIME <i>grace-period</i>	Integer number of days the users are allowed to login (while being issued a warning message) after their passwords are older than the PASSWORD_LIFE_TIME. After this period expires, users are forced to change their passwords on login if they have not done so after their password expired.	No grace period (the same as zero)
FAILED_LOGIN_ATTEMPTS <i>login-limit</i>	The number of consecutive failed login attempts that result in a user's account being locked.	Accounts are never locked, no matter how many failed login attempts are made.
PASSWORD_LOCK_TIME <i>lock-period</i>	Integer value setting the number of days an account is locked after the user's account was locked by having too many failed login attempts. After the PASSWORD_LOCK_TIME has expired, the account is automatically unlocked.	Accounts locked because of too many failed login attempts are never automatically unlocked. They must be manually unlocked by the database superuser.
PASSWORD_REUSE_MAX <i>reuse-limit</i>	The number of password changes that need to occur before the current password can be reused.	Users are not required to change passwords a certain number of times before reusing an old password.
PASSWORD_REUSE_TIME <i>reuse-period</i>	The integer number of days that must pass after a password has been set before the before it can be reused.	Password reuse is not limited by time.
PASSWORD_MAX_LENGTH <i>max-length</i>	The maximum number of characters allowed in a password. Value must be in the range of 8 to 100.	Passwords are limited to 100 characters.
PASSWORD_MIN_LENGTH <i>min-length</i>	The minimum number of characters required in a password. Valid range is 0 to <i>max-length</i> .	Equal to <i>max-length</i> .
PASSWORD_MIN_LETTERS <i>min-of-letters</i>	Minimum number of letters (a-z and A-Z) that must be in a password. Valid ranged is 0 to <i>max-length</i> .	0 (no minimum).
PASSWORD_MIN_UPPERCASE_LETTERS <i>min-cap-letters</i>	Minimum number of capital letters (A-Z) that must be in a password. Valid range is is 0 to	0 (no minimum).

	<i>max-length</i> .	
PASSWORD_MIN_LOWERCASE_LETTERS <i>min-lower-letters</i>	Minimum number of lowercase letters (a-z) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).
PASSWORD_MIN_DIGITS <i>min-digits</i>	Minimum number of digits (0-9) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).
PASSWORD_MIN_SYMBOLS <i>min-symbols</i>	Minimum number of symbols (any printable non-letter and non-digit character, such as \$, #, @, and so on) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).

**Note:** Only the profile settings for how many failed login attempts trigger account locking and how long accounts are locked have an effect on external password authentication methods such as LDAP or Kerberos. All password complexity, reuse, and lifetime settings only have an effect on passwords managed by Vertica.

## ALTER PROFILE RENAME

Rename an existing profile.

### Syntax

```
ALTER PROFILE name RENAME TO newname;
```

### Parameters

<i>name</i>	The current name of the profile.
<i>newname</i>	The new name for the profile.

## ALTER RESOURCE POOL

Modifies a resource pool.

### Syntax

```
ALTER RESOURCE POOL pool-name MEMORYSIZE 'sizeUnits'
... [ MAXMEMORYSIZE 'sizeUnits' | NONE ]
... [ PRIORITY integer ]
... [ QUEUETIMEOUT integer | NONE ]
... [ PLANNEDCONCURRENCY integer ]
... [ SINGLEINITIATOR bool ]
... [ MAXCONCURRENCY integer | NONE ]
```

## Parameters

<i>pool-name</i>	Specifies the name of the resource pool to alter.
MEMORYSIZE ' <i>sizeUnits</i> '	<p>[Default 0%] Amount of memory allocated to the resource pool. See also MAXMEMORYSIZE parameter.</p> <p>Units can be one of the following:</p> <ul style="list-style-type: none"> <li>▪ % percentage of total memory available to the Resource Manager. (In this case size must be 0-100).</li> <li>▪ K Kilobytes</li> <li>▪ M Megabytes</li> <li>▪ G Gigabytes</li> <li>▪ T Terabytes</li> </ul> <p><b>Note:</b> The MEMORYSIZE parameter refers to memory allocated to this pool per node and not across the whole cluster. The default of 0% means that the pool has no memory allocated to it and must exclusively borrow from the GENERAL <b>pool</b> (page 652).</p>
MAXMEMORYSIZE ' <i>sizeUnits</i> '   NONE	<p>[Default unlimited] Maximum size the resource pool could grow by borrowing memory from the GENERAL pool. See <b>Built-in Pools</b> (page 652) for a discussion on how resource pools interact with the GENERAL pool.</p> <p>Units can be one of the following:</p> <ul style="list-style-type: none"> <li>▪ % percentage of total memory available to the Resource Manager. (In this case, size must be 0-100). This notation has special meaning for the GENERAL pool, described in Notes below.</li> <li>▪ K Kilobytes</li> <li>▪ M Megabytes</li> <li>▪ G Gigabytes</li> <li>▪ T Terabytes</li> </ul> <p>If MAXMEMORYSIZE NONE is specified, there is no upper limit.</p> <p><b>Notes:</b></p> <p>The MAXMEMORYSIZE parameter refers to the maximum memory borrowed by this pool per node and not across the whole cluster.</p> <p>The default of unlimited means that the pool can borrow as much memory from GENERAL pool as is available.</p> <p>When set as a percentage (%) value, GENERAL.MAXMEMORYSIZE governs the total amount of RAM that the Resource Manager can use for queries, regardless of whether the parameter is set to a percent or to a specific value (for example, '10G'). The default setting is 95%.</p> <p>The MAXMEMORYSIZE of the WOSDATA and SYSDATA pools cannot be changed as long as any of their memory is in use. For example, in order to change the MAXMEMORYSIZE of the WOSDATA pool, you need to disable any trickle loading jobs and wait until the WOS is empty before you can change the MAXMEMORYSIZE.</p>

PRIORITY	[Default 0] An integer that represents priority of queries in this pool, when they compete for resources in the <code>GENERAL</code> pool. Higher numbers denote higher priority. Administrator-created resource pools can have a priority of -100 to 100. The built-in resource pools <code>SYSQUERY</code> , <code>RECOVERY</code> , and <code>TM</code> can have a range of -110 to 110.
QUEUE_TIMEOUT	[Default 300 seconds] An integer, in seconds, that represents the maximum amount of time the request is allowed to wait for resources to become available before being rejected. If set to <code>NONE</code> , the request can be queued for an unlimited amount of time.
PLANNED_CONCURRENCY	[Default: $\text{Max}(4, \text{Min}(\text{total available memory}/2\text{GB}, \#\text{cores}))$ ] An integer that represents number of concurrent queries that are normally expected to be running against the resource pool. This is not a hard limit and is used when apportioning memory in the pool to various requests. <b>Note:</b> This is a cluster wide maximum and NOT a per-node limit.
SINGLE_INITIATOR	[Default false] A boolean that indicates whether all requests using this pool are issued against the same initiator node or whether multiple initiator nodes can be used; for instance in a round robin configuration. <b>Note:</b> Vertica recommends distributing requests evenly across all nodes and leaving this parameter unchanged.
MAX_CONCURRENCY	[Default unlimited] An integer that represents the maximum number of concurrent execution slots available to the resource pool. If <code>MAX_CONCURRENCY NONE</code> is specified, there is no limit. <b>Note:</b> This is a cluster wide maximum and NOT a per-node limit.

## Notes

- The resource pool must exist before you can issue the `ALTER RESOURCE POOL` (page 582) command.
- Resource pool names are subject to the same rules as Vertica *identifiers* (page 15). **Built-in pool** (page 652) names cannot be used for user-defined pools.
- New resource pools can be created or altered without shutting down the system. The only exception is that changes to `GENERAL.MAXMEMORYSIZE` take effect only on a node restart. When a new pool is created (or its size altered), `MEMORYSIZE` amount of memory is taken out of the `GENERAL` pool. If the `GENERAL` pool does not currently have sufficient memory to create the pool due to existing queries being processed, a request is made to the system to create a pool as soon as resources become available. The pool is in operation as soon as the specified amount of memory becomes available. You can monitor whether the `ALTER` has been completed in the `V_MONITOR.RESOURCE_POOL_STATUS` (page 814) system table.
- If the `GENERAL.MAXMEMORYSIZE` parameter is modified while a node is down, and that node is restarted, the restarted node sees the new setting whereas other nodes continue to see the old setting until they are restarted. Vertica recommends that you do not change this parameter unless absolutely necessary.

- Under normal operation, `MEMORYSIZE` is required to be less than `MAXMEMORYSIZE` and an error is returned during `CREATE/ALTER` operations if this size limit is violated. However, under some circumstances where the node specification changes by addition/removal of memory, or if the database is moved to a different cluster, this invariant could be violated. In this case, `MAXMEMORYSIZE` is reduced to `MEMORYSIZE`.
- If two pools have the same `PRIORITY`, their requests are allowed to borrow from the `GENERAL` pool in order of arrival.

See Guidelines for Setting Pool Parameters in the Administrator's Guide for details about setting these parameters.

### See Also

**`CREATE RESOURCE POOL`** (page 649)

**`CREATE USER`** (page 695)

**`DROP RESOURCE POOL`** (page 705)

**`RESOURCE_POOL_STATUS`** (page 877)

**`SET SESSION RESOURCE POOL`** (page 778)

**`SET SESSION MEMORYCAP`** (page 777)

Managing Workloads in the Administrator's Guide

## ALTER ROLE RENAME

Rename an existing role.

### Syntax

```
ALTER ROLE name RENAME [TO] new_name;
```

### Parameters

<i>name</i>	The current name of the role that you want to rename.
<i>new_name</i>	The new name for the role.

### Example

```
=> ALTER ROLE applicationadministrator RENAME TO appadmin;
ALTER ROLE
```

## ALTER SCHEMA

Renames one or more existing schemas.

### Syntax

```
ALTER SCHEMA schema-name [ , ... ] {
```

```
... RENAME TO new-schema-name [ , ... ] }
```

## Parameters

<i>schema-name</i>	Specifies the name of one or more schemas to rename.
RENAME TO	<p>Specifies one or more new schema names.</p> <p>The lists of schemas to rename and the new schema names are parsed from left to right and matched accordingly using one-to-one correspondence.</p> <p>When renaming schemas, be sure to follow these standards:</p> <ul style="list-style-type: none"><li>▪ The number of schemas to rename must match the number of new schema names supplied.</li><li>▪ The new schema names must not already exist.</li></ul> <p>The RENAME TO parameter is applied atomically. Either all the schemas are renamed or none of the schemas are renamed. If, for example, the number of schemas to rename does not match the number of new names supplied, none of the schemas are renamed.</p> <p><b>Note:</b> Renaming a schema that is referenced by a view will cause the view to fail unless another schema is created to replace it.</p>

## Notes

- Only the superuser or schema owner can use the ALTER SCHEMA command.
- Renaming schemas does not affect existing prejoin projections because prejoin projections refer to schemas by the schemas' unique numeric IDs (OIDs), and the OIDs for schemas are not changed by ALTER SCHEMA.

## Tip

Renaming schemas is useful for swapping schemas without actually moving data. To facilitate the swap, enter a non-existent, temporary placeholder schema. The following example uses the temporary schema *temps* to facilitate swapping schema S1 with schema S2. In this example, S1 is renamed to *temps*. Then S2 is renamed to S1. Finally, *temps* is renamed to S2.

```
ALTER SCHEMA S1, S2, temps
  RENAME TO temps, S1, S2;
```

## Examples

The following example renames schema S1 to S3 and schema S2 to S4:

```
ALTER SCHEMA S1, S2
  RENAME TO S3, S4;
```

## See Also

**CREATE SCHEMA** (page 657) and **DROP SCHEMA** (page 706)

## ALTER SEQUENCE

Changes the sequence attributes.

**Syntax1**

```
ALTER SEQUENCE [schema-name.] sequence-name
... [ INCREMENT [ BY ] increment ]
... [ MINVALUE minvalue | NO MINVALUE ]
... [ MAXVALUE maxvalue | NO MAXVALUE ]
... [ START [ WITH ] start ]
... [ RESTART [ [ WITH ] restart ] ]
... [ CACHE cache ]
... [ CYCLE | NO CYCLE ]
```

**Syntax 2**

```
ALTER SEQUENCE [schema-name.] name RENAME TO new-name
```

**Syntax 3**

```
ALTER SEQUENCE [schema-name.] name SET SCHEMA new-schema-name
```

**Parameters**

<i>sequence-name</i>	The name (optionally schema-qualified) of the sequence to be altered. The name must be unique among sequences, tables, projections, and views.
<i>increment</i>	Modifies the value, which is added to the current sequence value to create a new value. A positive value makes an ascending sequence, a negative one a descending sequence.
MINVALUE   NO MINVALUE	Modifies the minimum value a sequence can generate. If you change this value and the current value falls outside of the range, the current value is changed to the minimum value if increment is greater than zero or the maximum value if increment is smaller than 0.
MAXVALUE   NO MAXVALUE	Modifies the maximum value for the sequence. If you change this value and the current value falls outside of the range, the current value is changed to the minimum value if increment is greater than zero or the maximum value if increment is smaller than 0.
<i>start</i>	Allows the sequence to begin anywhere.
<i>restart</i>	Changes the current value of the sequence. The specified value is returned by the next call of NEXTVAL.
<i>cache</i>	Modifies how many sequence numbers are preallocated and stored in memory for faster access. The default is 250,000 with a minimum value of 1 (only one value can be generated at a time, for example, no cache).
CYCLE   NO CYCLE	Allows you to switch between CYCLE and NO CYCLE. The CYCLE option allows the sequence to wrap around when the max value or minvalue is reached by an ascending or descending sequence respectively. If the limit is reached, the next number generated is the minvalue or maxvalue, respectively. If NO CYCLE is specified, any calls to NEXTVAL after the

	sequence has reached its maximum/minimum value, return an error. The default is NO CYCLE.
RENAME TO <i>new-name</i>	Renames a sequence within the same schema. To move a sequence, see SET SCHEMA below.
SET SCHEMA <i>new-schema-name</i>	Moves a sequence between schemas.

### Notes

- You must own the sequence or be the superuser to use ALTER SEQUENCE.
- To change a sequence's schema, you must also have CREATE privilege on the new schema.
- Any parameters not specifically set in the ALTER SEQUENCE command retain their prior settings.

### Examples

The following example modifies an ascending sequence called sequential to start at 105:

```
ALTER SEQUENCE sequential RESTART WITH 105;
```

The following example moves a sequence from one schema to another:

```
ALTER SEQUENCE [public.]sequence SET SCHEMA vmart;
```

The following example renames a sequence in the Vmart schema:

```
ALTER SEQUENCE [vmart.]sequence RENAME TO serial;
```

### See Also

**CREATE SEQUENCE** (page 658)

**CURRVAL** (page 310)

**DROP SEQUENCE** (page 708)

**GRANT (Sequence)** (page 725)

**NEXTVAL** (page 308)

Using Sequences and Sequence Privileges in the Administrator's Guide

## ALTER TABLE

Modifies an existing table.

### Syntax1

```
ALTER TABLE [schema-name.] table-name {
... ADD COLUMN column-definition
... | ADD table-constraint (on page 593)
... | ALTER COLUMN column-name [ SET DEFAULT default-expression ]
... | [ DROP DEFAULT ]
... | DROP CONSTRAINT constraint-name [ RESTRICT | CASCADE ]
... | RENAME [ COLUMN ] column TO new-column
... | SET SCHEMA new-schema-name [ CASCADE | RESTRICT ] }
```

### Syntax2

```
ALTER TABLE [schema-name.] table-name [ , ... ]
... RENAME [TO] new-table-name [ , ... ]
```

### Parameters

<p><i>[schema-name.]table-name</i></p>	<p>Specifies the name of the table to be altered. When using more than one schema, specify the schema that contains the table.</p> <p>ALTER TABLE can be used in conjunction with SET SCHEMA to move only one table between schemas at a time.</p> <p>When using ALTER TABLE to rename one or more tables, you can specify a comma-delimited list of table names to rename.</p>
<p>ADD COLUMN <i>column-definition</i></p>	<p>Adds a new column defined by <i>column-definition</i> to a table and to all superprojections of the table. The column definition cannot contain constraints. Columns cannot be added to tables that have out-of-date superprojections with up-to-date buddies.</p> <p>When a new column is added:</p> <ul style="list-style-type: none"> <li>▪ A unique projection column name is generated in each superprojection.</li> <li>▪ The default value is inserted for existing rows. For example, if CURRENT_TIMESTAMP is the default expression, all rows have the current timestamp.</li> </ul> <p><b>Tip:</b> When adding a column, only expressions that can be folded into a constant can be specified as a default column expression. This means that all system information functions (except CURRENT_DATABASE()) cannot be specified. Other functions that cannot be used are CURRENT_USER, SESSION_USER, and certain formatting functions such as TO_DATE and TO_TIMESTAMP.</p> <p>Volatile functions cannot be specified through ADD COLUMN. Use ALTER COLUMN to specify volatile functions. (Volatile functions change with every invocation.)</p> <p><b>Note:</b> Columns added to a table that is referenced by a view do not appear in the result set of the view even if the view uses the wild card (*) to represent all columns in the table. Recreate the view to incorporate the</p>

	column.
ADD	Adds a <b>table-constraint</b> (on page 593) to a table that does not have any associated projections. <b>Note:</b> Adding a table constraint has no effect on views that reference the table.
ALTER COLUMN	Alters an existing column within the specified table to change or drop a default expression. <b>Tip:</b> You cannot specify a volatile function in its default column expression. To work around this, add the column and fill it with NULLS. Then, alter the column to specify the volatile function. For example: <pre>ALTER TABLE tbl ADD COLUMN newcol float; ALTER TABLE tbl ALTER COLUMN newcol SET DEFAULT random();</pre>
DROP CONSTRAINT	Drops the specified table-constraint from the table. <b>Note:</b> Dropping a table constraint has no effect on views that reference the table.
[ RESTRICT   CASCADE ]	Use the <b>CASCADE</b> keyword to drop a constraint upon which something else depends. For example, a <b>FOREIGN KEY</b> constraint depends on a <b>UNIQUE</b> or <b>PRIMARY KEY</b> constraint on the referenced columns.

RENAME [TO]	<p>RENAME can be used to rename one or more tables. In either case, the key word changes the name of the table or tables to the specified name or names.</p> <p>To rename two or more tables simultaneously, use a comma-delimited list. The lists of tables to rename and the new table names are parsed from left to right and matched accordingly using one-to-one correspondence.</p> <p>When renaming tables, be sure to follow these standards:</p> <ul style="list-style-type: none"> <li>▪ Do not specify the schema-name as part of the table specification after the RENAME TO clause. The schema-name is specified only after the ALTER TABLE clause because this statement applies to only one schema.</li> <li>▪ The following example renames tables T1 and T2 in the S1 schema to U1 and U2 respectively. =&gt; ALTER TABLE S1.T1, S1.T2 RENAME TO U1, U2;</li> <li>▪ The following example generates a syntax error: =&gt; ALTER TABLE S1.T1, S1.T2 RENAME TO S1.U1, S1.U2;</li> <li>▪ The number of tables to rename must match the number of new table names supplied.</li> <li>▪ The new table names must not already exist.</li> </ul> <p>The RENAME TO parameter is applied atomically. Either all the tables are renamed or none of the tables are renamed. If, for example, the number of tables to rename does not match the number of new names supplied, none of the tables are renamed.</p> <p><b>Note:</b> Renaming a table that is referenced by a view causes the view to fail unless another table is created to replace it.</p>
RENAME [ COLUMN ]	<p>Renames the specified column within the table.</p> <p><b>Note:</b> If a column that is referenced by a view is renamed, the column does not appear in the result set of the view even if the view uses the wild card (*) to represent all columns in the table. Recreate the view to incorporate the column's new name.</p>
SET SCHEMA	<p>Moves the table to the specified schema. By default, SET SCHEMA is set to CASCADE. This means that all the projections that are anchored on this table are automatically moved to the new schema regardless of the schema in which they reside. To move only projections that are anchored on this table and that reside in the same schema, use the RESTRICT key word.</p> <p>If the name of the table or any of the projections that you want to move already exists in the new schema, the statement rolls back and the tables and projections are not moved. In the new schema, rename the table or projections that conflict with the ones that you want to move and then rerun the statement.</p> <p><b>Notes:</b> Although this is likely to occur infrequently, Vertica supports moving system tables to system schemas if necessary. This might occur to support designs created through Database Designer.</p> <p>Temporary tables cannot be moved between schemas.</p> <p>SET SCHEMA supports moving only one table between schemas at a time.</p>

## Notes

- To use the `ALTER TABLE` statement, the user must either be a superuser or be the table owner and have `CREATE` privilege on the affected schema. If you use `SET SCHEMA`, you must also have `CREATE` privilege on the schema to which you want to move the table.
- With the exception of performing a table rename, one operation can be performed at a time in an `ALTER TABLE` command; for example, to add multiple columns, issue consecutive `ALTER TABLE ADD COLUMN` commands.
- You cannot add a column constraint using the `ALTER TABLE` command:  

```
=> ALTER TABLE t1 ADD COLUMN c INT NOT NULL;  
ROLLBACK: ALTER TABLE does not support ADD COLUMN with other clauses
```
- The following clauses cannot be used with any other clauses. They are exclusive:
  - `RENAME [TO]`
  - `RENAME COLUMN`
  - `SET SCHEMA`
  - `ADD COLUMN`
- The `ADD` constraints and `DROP` constraints clauses can be used together.
- Adding a column to a table does not affect the K-safety of the physical schema design.
- You cannot use `ALTER TABLE ... ADD COLUMN` on a temporary table.
- Vertica allows adding 1600 columns to a table.
- Renaming tables does not affect existing pre-join projections because pre-join projections refer to tables by the tables' unique numeric IDs (OIDs), and the OIDs for tables are not changed by `ALTER TABLE`.

## Tip

Renaming tables is useful for swapping tables within the same schema without actually moving data. It cannot be used to swap tables across schemas. To enable the swap, use a non-existent, temporary placeholder table. The following example uses the temporary table *temps* to facilitate swapping table *T1* with table *T2*. In this example, *T1* is renamed to *temps*. Then *T2* is renamed to *T1*. Finally, *temps* is renamed to *T2*.

```
=> ALTER TABLE T1, T2, temps RENAME TO temps, T1, T2;
```

## Examples

The following example drops the default expression specified for the `Discontinued_flag` column.

```
=> ALTER TABLE Retail.Product_Dimension  
ALTER COLUMN Discontinued_flag DROP DEFAULT;
```

The following example renames a column in the `Retail.Product_Dimension` table from `Product_description` to `Item_description`:

```
=> ALTER TABLE Retail.Product_Dimension  
RENAME COLUMN Product_description TO Item_description;
```

The following example moves table T1 from schema S1 to schema S2. SET SCHEMA defaults to CASCADE so all the projections that are anchored on table T1 are automatically moved to schema S2 regardless of the schema in which they reside

```
=> ALTER TABLE S1.T1 SET SCHEMA S2;
```

## table-constraint

Adds a join constraint to the metadata of a table. See Adding Constraints in the Administrator's Guide.

### Syntax

```
[ CONSTRAINT constraint_name ]
... { PRIMARY KEY ( column [ , ... ] )
... | FOREIGN KEY ( column [ , ... ] )
... REFERENCES table
... | UNIQUE ( column [ , ... ] )
```

### Parameters

CONSTRAINT <i>constraint_name</i>	Optionally assigns a name to the constraint. Vertica recommends that you name all constraints.
PRIMARY KEY ( <i>column</i> [ , ... ] )	Adds a referential integrity constraint defining one or more NOT NULL numeric columns as the primary key.
FOREIGN KEY ( <i>column</i> [ , ... ] )	Adds a referential integrity constraint defining one or more numeric columns as a foreign key.
REFERENCES <i>table</i>	Specifies the table to which the FOREIGN KEY constraint applies. If <i>column</i> is omitted, the default is the primary key of <i>table</i> .
UNIQUE ( <i>column</i> [ , ... ] )	Ensures that the data contained in a column or a group of columns is unique with respect to all the rows in the table.

### Notes

- A foreign key constraint can be specified solely by a reference to the table that contains the primary key. The columns in the referenced table do not need to be explicitly specified; for example:

```
CREATE TABLE fact(c1 INTEGER PRIMARY KEY);
CREATE TABLE dim (c1 INTEGER REFERENCES fact);
```

- Define PRIMARY KEY and FOREIGN KEY constraints in all tables that participate in inner joins. See Adding Constraints.
- Adding constraint to a table that is referenced in a view does not affect the view.

### Examples

```
CORRELATION (Product_Description) DETERMINES (Category_Description)
```

The Retail Sales Example Database described in the Getting Started Guide contains a table Product\_Dimension in which products have descriptions and categories. For example, the description "Seafood Product 1" exists only in the "Seafood" category. You can define several similar correlations between columns in the Product Dimension table.

## ALTER USER

Changes a database user account.

### Syntax

```
ALTER USER name
... [ ACCOUNT { LOCK | UNLOCK } ]
... [ DEFAULT ROLE {role [, ...] | NONE} ]
... [ IDENTIFIED BY 'password' [ REPLACE 'old-password' ] ]
... [ MEMORYCAP { 'memory-limit' | NONE } ]
... [ PASSWORD EXPIRE ]
... [ PROFILE { profile-name | DEFAULT } ]
... [ RESOURCE POOL pool-name ]
... [ RUNTIMECAP { 'time-limit' | NONE } ]
... [ TEMPSPACECAP { 'space-limit' | NONE } ]
```

### Parameters

<i>name</i>	Specifies the name of the user to alter; names that contain special characters must be double-quoted.
ACCOUNT LOCK   UNLOCK	Locks or unlocks the user's account. Users cannot log in if their account is locked. Accounts can be locked either explicitly by the superuser, or if the user has more failed login attempts than is allowed by their profile.
DEFAULT ROLE { <i>role</i> [, ...]   NONE}	One or more roles that should be active when the user's session starts. The user must have already been granted access to the roles (see <b>GRANT (Role)</b> (page 723)). The role or roles specified in this command replace any existing default roles. Use the NONE keyword to eliminate all default roles for the user.
IDENTIFIED BY ' <i>password</i> ' [ REPLACE ' <i>old_password</i> ' ]	Sets the user's password to <i>password</i> . Non-superusers can only their own passwords and must supply the REPLACE parameter along with their old password. The superuser can change any user's password without supplying the REPLACE parameter.
PASSWORD EXPIRE	Expires the user's password. Vertica will force the user to change passwords during his or her next login. <b>Note:</b> PASSWORD EXPIRE has no effect when using external password authentication methods such as LDAP or Kerberos.
PROFILE <i>profile-name</i>   DEFAULT	Sets the user's profile to <i>profile-name</i> . Using the value DEFAULT sets the user's profile to the default profile.
MEMORYCAP ' <i>memory-limit</i> '   NONE	Limits the amount of memory that the user's requests can use. This value is a number representing the amount of space, followed by a unit (for example,

	'10G'). The unit can be one of the following: <ul style="list-style-type: none"> <li>▪ % percentage of total memory available to the Resource Manager. (In this case value of the size size must be 0-100)</li> <li>▪ K Kilobytes</li> <li>▪ M Megabytes</li> <li>▪ G Gigabytes</li> <li>▪ T Terabytes</li> </ul> Setting this value to <code>NONE</code> means the user has no limits on memory use.
<code>RESOURCE POOL pool-name</code>	Sets the name of the default resource pool for the user.
<code>RUNTIMECAP 'time-limit'   NONE</code>	Sets the maximum amount of time any of the user's queries can execute. <code>time-limit</code> is an interval, such as '1 minute' or '100 seconds' (see <b>Interval Values</b> (page 30) for details). This value cannot exceed one year. Setting this value to <code>NONE</code> means there is no time limit on the user's queries.
<code>TEMPSPACECAP 'space-limit'   NONE</code>	Limits the amount of temporary file storage the user's requests can use. This parameter's value has the same format as the <code>MEMORYCAP</code> value.

## Notes

- Users can alter some of their own settings. They can set their default `RESOURCE POOL` to any pool on which they have been granted usage privileges. They can also change their password. They must supply their old password to do so.
- Only a superuser can alter another user.
- Attempting to switch users to using a resource pool on which they have not been granted access results in an error (even for the superuser).
- `ALTER USER` does not affect current sessions.

## See Also

**CREATE USER** (page 695)

Managing Workloads in the Administrator's Guide

## COMMENT ON COLUMN

Adds, revises, or removes a comment on a projection column. Each object can have a maximum of 1 comment (1 or 0). Comments are stored in the `V_CATALOG.COMMENTS` system table.

## Syntax

```
COMMENT ON COLUMN [ schema_name.]proj_name.column_name IS [ 'comment' | NULL ]
```

## Parameters

<code>[schema_name.]proj_name.column_name</code>	The name of the projection and column associated with the comment.
--	--

<i>comment</i>	Text of the comment you want to add. Enclose the text of the comment within single-quotes. If a comment already exists for this column, the comment you enter here overwrites the previous comment.  You can enclose a blank value within single quotes to remove an existing comment.
NULL	Allows you to remove an existing comment.

### Notes

- Comments can be added to projection columns only; they cannot be added to table columns.
- The superuser can view and add comments to all objects.
- A user must own an object to be able to add or edit comments for the object.
- A user must have view privileges on an object to view its comments.
- If you drop an object, all comments associated with the object are dropped as well.

### Example

The following example adds a comment for the `customer_name` column in the `customer_dimension` projection:

```
=> COMMENT ON COLUMN customer_dimension_vmart_node01.customer_name IS 'Last name only';
```

The following examples remove a comment from the `customer_name` column in the `customer_dimension` projection:

```
=> COMMENT ON COLUMN customer_dimension_vmart_node01.customer_name IS NULL;
=> COMMENT ON COLUMN customer_dimension_vmart_node01.customer_name IS '';
```

### See also

**V\_CATALOG.COMMENTS** (page 797)

## COMMENT ON CONSTRAINT

Adds, revises, or removes a comment on a constraint. Each object can have a maximum of 1 comment (1 or 0). Comments are stored in the `V_CATALOG.COMMENTS` system table.

### Syntax

```
COMMENT ON CONSTRAINT constraint_name ON [ schema. ] table_name IS
... [ 'comment' | NULL ];
```

### Parameters

<i>constraint_name</i>	The name of the constraint associated with the comment.
[ <i>schema.</i> ] <i>table_name</i>	The name of the table containing the constraint associated with the comment.
<i>comment</i>	Text of the comment you want to add. Enclose the text of the comment within single-quotes. If a comment already exists for

	<p>this constraint, the comment you enter here overwrites the previous comment.</p> <p>You can enclose a blank value within single quotes to remove an existing comment.</p>
NULL	Allows you to remove an existing comment.

### Notes

- The superuser can view and add comments to all objects.
- A user must own an object to be able to add or edit comments for the object.
- A user must have viewing privileges on an object to view its comments.
- If you drop an object, all comments associated with the object are dropped as well.

### Example

The following example adds a comment for the `constraint_x` constraint on the `promotion_dimension` table:

```
=> COMMENT ON CONSTRAINT constraint_x ON promotion_dimension IS 'Primary key';
```

The following examples remove a comment from the `constraint_x` constraint on the `promotion_dimension` table:

```
=> COMMENT ON CONSTRAINT constraint_x ON promotion_dimension IS NULL;
=> COMMENT ON CONSTRAINT constraint_x ON promotion_dimension IS '';
```

### See Also

**V\_CATALOG.COMMENTS** (page 797)

## COMMENT ON FUNCTION

Adds, revises, or removes a comment on a function. Each object can have a maximum of 1 comment (1 or 0). Comments are stored in the V\_CATALOG.COMMENTS system table.

### Syntax

```
COMMENT ON FUNCTION function_name function_arg IS [ 'comment' | NULL ];
```

### Parameters

<i>function_name</i>	The name of the function associated with the comment.
<i>function_arg</i>	The arguments used by the function.
<i>comment</i>	<p>Text of the comment you want to add. Enclose the text of the comment within single-quotes. If a comment already exists for this function, the comment you enter here overwrites the previous comment.</p> <p>You can enclose a blank value within single quotes to remove an existing comment.</p>
NULL	Allows you to remove an existing comment.

## Notes

- The superuser can view and add comments to all objects.
- A user must own an object to be able to add or edit comments for the object.
- A user must have viewing privileges on an object to view its comments.
- If you drop an object, all comments associated with the object are dropped as well.

## Example

The following example adds a comment for the `macros.zerowhennull (x INT)` function:

```
=> COMMENT ON FUNCTION macros.zerowhennull(x INT) IS 'Returns a 0 if not NULL';
```

The following examples remove a comment the `macros.zerowhennull (x INT)` function:

```
=> COMMENT ON FUNCTION macros.zerowhennull(x INT) IS NULL;  
=> COMMENT ON FUNCTION macros.zerowhennull(x INT) IS '';
```

## See Also

**V\_CATALOG.COMMENTS** (page 797)

# COMMENT ON LIBRARY

Adds, revises, or removes a comment on a library . Each object can have a maximum of 1 comment (1 or 0). Comments are stored in the V\_CATALOG.COMMENTS system table.

## Syntax

```
COMMENT ON LIBRARY [ schema. ] library_name IS [ 'comment' | NULL ]
```

## Parameters

<i>[schema.] library_name</i>	The name of the library associated with the comment.
<i>comment</i>	Text of the comment you want to add. Enclose the text of the comment within single-quotes. If a comment already exists for this library, the comment you enter here overwrites the previous comment.  You can enclose a blank value within single quotes to remove an existing comment.
NULL	Allows you to remove an existing comment.

## Notes

- The superuser can view and add comments to all objects.
- A user must own an object to be able to add or edit comments for the object.
- A user must have view privileges on an object to view its comments.
- If you drop an object, all comments associated with the object are dropped as well.

## Example

The following example adds a comment for the library `MyFunctions`:

```
=> COMMENT ON LIBRARY MyFunctions IS 'In development';
```

The following examples remove a comment the library `MyFunctions`:

```
=> COMMENT ON LIBRARY MyFunctions IS NULL;
=> COMMENT ON LIBRARY MyFunctions IS '';
```

## See Also

[\*\*V\\_CATALOG.COMMENTS\*\*](#) (page 797)

## COMMENT ON NODE

Adds, revises, or removes a comment on a node. Each object can have a maximum of 1 comment (1 or 0). Comments are stored in the `V_CATALOG.COMMENTS` system table.

## Syntax

```
COMMENT ON NODE node_name IS [ 'comment' | NULL ]
```

## Parameters

<i>node_name</i>	The name of the node associated with the comment.
<i>comment</i>	Text of the comment you want to add. Enclose the text of the comment within single-quotes. If a comment already exists for this node, the comment you enter here overwrites the previous comment.  You can enclose a blank value within single quotes to remove an existing comment.
NULL	Allows you to remove an existing comment.

## Notes

- Only the superuser can add or edit comments on nodes.
- The superuser can view and add comments to all objects.
- A user must own an object to be able to add or edit comments for the object.
- A user must have viewing privileges on an object to view its comments.
- If you drop an object, all comments associated with the object are dropped as well.

## Example

The following example adds a comment for the `initiator` node:

```
=> COMMENT ON NODE initiator IS 'Initiator node';
```

The following examples remove a comment the `initiator` node.

```
=> COMMENT ON NODE initiator IS NULL;
=> COMMENT ON NODE initiator IS '';
```

**See Also****V\_CATALOG.COMMENTS** (page 797)

## COMMENT ON PROJECTION

Adds, revises, or removes a comment on a projection. Each object can have a maximum of 1 comment (1 or 0). Comments are stored in the V\_CATALOG.COMMENTS system table.

**Syntax**

```
COMMENT ON PROJECTION [ schema. ] proj_name IS [ 'comment' | Null ]
```

**Parameters**

<i>[schema.] projection_name</i>	The name of the projection associated with the comment.
<i>comment</i>	Text of the comment you want to add. Enclose the text of the comment within single-quotes. If a comment already exists for this projection, the comment you enter here overwrites the previous comment. You can enclose a blank value within single quotes to remove an existing comment.
Null	Allows you to remove an existing comment.

**Notes**

- The superuser can view and add comments to all objects.
- A user must own an object to be able to add or edit comments for the object.
- A user must have view privileges on an object to view its comments.
- If you drop an object, all comments associated with the object are dropped as well.

**Example**

The following example adds a comment for the `customer_dimension_vmart_node01` projection:

```
=> COMMENT ON PROJECTION customer_dimension_vmart_node01 IS 'Test data';
```

The following examples remove a comment from the `customer_dimension_vmart_node01` projection

```
=> COMMENT ON PROJECTION customer_dimension_vmart_node01 IS NULL;
```

```
=> COMMENT ON PROJECTION customer_dimension_vmart_node01 IS '';
```

**See Also****V\_CATALOG.COMMENTS** (page 797)

## COMMENT ON SCHEMA

Adds, revises, or removes a comment on a schema. Each object can have a maximum of 1 comment (1 or 0). Comments are stored in the V\_CATALOG.COMMENTS system table.

### Syntax

```
COMMENT ON SCHEMA schema_name IS [ 'comment' | NULL ]
```

### Parameters

<i>schema_name</i>	The name of the schema associated with the comment.
<i>comment</i>	Text of the comment you want to add. Enclose the text of the comment within single-quotes. If a comment already exists for this schema, the comment you enter here overwrites the previous comment. You can enclose a blank value within single quotes to remove an existing comment.
NULL	Allows you to remove an existing comment.

### Notes

- The superuser can view and add comments to all objects.
- A user must own an object to be able to add or edit comments for the object.
- A user must have view privileges on an object to view its comments.
- If you drop an object, all comments associated with the object are dropped as well.

### Example

The following example adds a comment for the `public` schema:

```
=> COMMENT ON SCHEMA public IS 'All users can access this schema';
```

The following examples remove a comment the `public` schema.

```
=> COMMENT ON SCHEMA public IS NULL;
=> COMMENT ON SCHEMA public IS '';
```

### See Also

**V\_CATALOG.COMMENTS** (page 797)

## COMMENT ON SEQUENCE

Adds, revises, or removes a comment on a sequence. Each object can have a maximum of 1 comment (1 or 0). Comments are stored in the V\_CATALOG.COMMENTS system table.

### Syntax

```
COMMENT ON SEQUENCE sequence_name IS [ 'comment' | NULL ]
```

## Parameters

<i>sequence_name</i>	The name of the sequence associated with the comment.
<i>comment</i>	Text of the comment you want to add. Enclose the text of the comment within single-quotes. If a comment already exists for this sequence, the comment you enter here overwrites the previous comment. You can enclose a blank value within single quotes to remove an existing comment.
NULL	Allows you to remove an existing comment.

## Notes

- The superuser can view and add comments to all objects.
- A user must own an object to be able to add or edit comments for the object.
- A user must have view privileges on an object to view its comments.
- If you drop an object, all comments associated with the object are dropped as well.

## Example

The following example adds a comment for a sequence called `prom_seq`.

```
=> COMMENT ON SEQUENCE prom_seq IS 'Promotion codes';
```

The following examples remove a comment from the `prom_seq` sequence.

```
=> COMMENT ON SEQUENCE prom_seq IS NULL;
```

```
=> COMMENT ON SEQUENCE prom_seq IS '';
```

## See Also

**V\_CATALOG.COMMENTS** (page 797)

## COMMENT ON TABLE

Adds, revises, or removes a comment on a table. Each object can have a maximum of 1 comment (1 or 0). Comments are stored in the V\_CATALOG.COMMENTS system table.

## Syntax

```
COMMENT ON TABLE [ schema. ] table_name IS [ 'comment' | NULL ]
```

## Parameters

[ <i>schema.</i> ] <i>table_name</i>	The name of the table associated with the comment.
<i>comment</i>	Text of the comment you want to add. Enclose the text of the comment within single-quotes. If a comment already exists for this table, the comment you enter here overwrites the previous comment. You can enclose a blank value within single quotes to remove an

	existing comment.
Null	Allows you to remove a previously added comment.

### Notes

- The superuser can view and add comments to all objects.
- A user must own an object to be able to add or edit comments for the object.
- A user must have view privileges on an object to view its comments.
- If you drop an object, all comments associated with the object are dropped as well.

### Example

The following example adds a comment to table `promotion_dimension`:

```
=> COMMENT ON TABLE promotion_dimension IS '2011 Promotions';
```

The following examples remove a comment from table `promotion_dimension`:

```
=> COMMENT ON TABLE promotion_dimension IS NULL;
```

```
=> COMMENT ON TABLE promotion_dimension IS '';
```

### See Also

**V\_CATALOG.COMMENTS** (page 797)

## COMMENT ON VIEW

Adds, revises, or removes a comment on a view. Each object can have a maximum of 1 comment (1 or 0). Comments are stored in the V\_CATALOG.COMMENTS system table.

### Syntax

```
COMMENT ON VIEW [schema.]view_name IS [ 'comment' | NULL]
```

### Parameters

<i>[schema.]</i> <i>view_name</i>	The name of the view associated with the comment.
<i>comment</i>	Text of the comment you want to add. If a comment already exists for this view, the comment you enter here overwrites the previous comment.
NULL	Allows you to remove an existing comment.

### Notes

- The superuser can view and add comments to all objects.
- A user must own an object to be able to add or edit comments for the object.
- A user must have view privileges on an object to view its comments.
- If you drop an object, all comments associated with the object are dropped as well.

### Example

The following example adds a comment to a view called `curr_month_ship`:

```
=> COMMENT ON VIEW curr_month_ship IS 'Shipping data for the current month';
```

The following example removes a comment from the `curr_month_ship` view:

```
=> COMMENT ON VIEW curr_month_ship IS NULL;
```

**See Also**

***V\_CATALOG.COMMENTS*** (page 797)

## COMMIT

Ends the current transaction and makes all changes that occurred during the transaction permanent and visible to other users.

### Syntax

```
COMMIT [ WORK | TRANSACTION ]
```

### Parameters

WORK   TRANSACTION	Have no effect; they are optional keywords for readability.
--------------------	---

## CONNECT

Connects to another Vertica database to enable data import (using the ***COPY FROM VERTICA*** (page 626) statement) or export (using the ***EXPORT*** (page 718) statement).

**Note:** You can only connect to a database for import or export that uses trust (username-only) or password-based authentication. LDAP or SSL authentication is not supported.

### Syntax

```
CONNECT TO VERTICA database USER username PASSWORD 'password' ON 'host',port
```

### Parameters

<i>database</i>	The connection target database name
<i>username</i>	The username to use when connecting to the other database
<i>password</i>	A string containing the password to use to connect to the other database
<i>host</i>	A string containing the hostname of one of the nodes in the other database
<i>port</i>	The port number of the other database as an integer

### Notes

- Once established, the connection remains open for the current session. Use the ***DISCONNECT*** (page 700) statement to disconnect.
- You can only have one connection to a database at the same time. You can connect to multiple different databases in the same session.
- If the target database does not have a password, and you specify a password in the **CONNECT** statement, the connection succeeds silently. You are not alerted that you supplied an incorrect password.

### Example

```
=> CONNECT TO VERTICA ExampleDB USER dbadmin PASSWORD 'Password123' ON 'VerticaHost01',5433;  
CONNECT
```

### See Also

***COPY FROM VERTICA*** (page 626)

***DISCONNECT*** (page 700)

***EXPORT*** (page 718)

## COPY

Bulk loads data from one or more files or pipes on a cluster host into a Vertica database. (See **LCOPY** (page 731) to load from a data file on a client system using ODBC.)

`COPY` can load data in one of three formats:

- Text with delimiters (the default format)
- Native binary using the `NATIVE` keyword
- Native varchar using the `NATIVE VARCHAR` keyword

See Advanced Formats for Loading Data in the Administrator's Guide for details of using native binary or varchar.

`COPY` has many options that give you flexibility when importing your data. For example it can:

- Read data compressed using GZIP or BZIP as well as uncompressed data.
- Insert data into the WOS (memory) or directly into the ROS (disk).
- Set certain parameters (such as the delimiters and quote characters) for the entire copy operation, or for specific columns.
- Transform data before inserting it into the database.

**Note:** You must connect as the database superuser to copy from a file. Any user with `INSERT` privileges can copy data from the `STDIN` pipe.

### Syntax

```
COPY [schema-name.]table
... [ ( { column-as-expression | column }
..... [ FILLER datatype ]
..... [ FORMAT 'format' ]
..... [ ENCLOSED BY 'char' ]
..... [ ESCAPE AS 'char' | NO ESCAPE ]
..... [ NULL [ AS ] 'string' ]
..... [ DELIMITER [ AS ] 'char' ]
... [, ... ] ) ]
... [ COLUMN OPTION ( column
..... [ FORMAT 'format' ]
..... [ ENCLOSED BY 'char' ]
..... [ ESCAPE AS 'char' | NO ESCAPE ]
..... [ NULL [ AS ] 'string' ]
..... [ DELIMITER [ AS ] 'char' ]
... [, ... ] ) ]
FROM { STDIN
...[ BZIP | GZIP | UNCOMPRESSED ] | 'pathToData' [ ON nodename ]
...[ BZIP | GZIP | UNCOMPRESSED ] [, ... ] }
...[ NATIVE | NATIVE VARCHAR ]
...[ WITH ]
...[ DELIMITER [ AS ] 'char' ]
...[ TRAILING NULLCOLS ]
...[ NULL [ AS ] 'string' ]
```

```

...[ ESCAPE AS 'char' | NO ESCAPE ]
...[ ENCLOSED BY 'char' [ AND 'char' ] ]
...[ RECORD TERMINATOR 'string' ]
...[ SKIP integer ]
...[ REJECTMAX integer ]
...[ EXCEPTIONS 'path' [ ON nodename ] [, ...] ]
...[ REJECTED DATA 'path' [ ON nodename ] [, ...] ]
...[ ENFORCELENGTH ]
...[ ABORT ON ERROR ]
...[ AUTO | DIRECT | TRICKLE ]
...[ STREAM NAME 'streamName']
...[ NO COMMIT ]

```

## Parameters

<p><i>[schema-name.] table</i></p>	<p>Specifies the name of a schema table (not a projection). Vertica loads the data into all projections that include columns from the schema table.</p> <p>When using more than one schema, specify the schema that contains the table.</p>
<p><i>column-as-expression</i></p>	<p>Specifies the target column, for which you want to compute values, as an expression. This is used to transform data when it is loaded into the target database. Transforming data is useful for computing values to be inserted into a column in the target database from other columns in the source. (See Transforming Data During Loads in the Administrator's Guide.)</p> <p>Transformation requirements:</p> <ul style="list-style-type: none"> <li>▪ The <code>COPY</code> statement must contain at least one parsed column, which can be a filler column. (See Ignoring Columns and Fields in the Load File in the Administrator's Guide for more information about using fillers.)</li> <li>▪ For parsed columns, specify only raw data in the source.</li> <li>▪ The return data type of the expression must be coercible to that of the target column. Parameter (parsed columns) are also coerced to match the expression.</li> <li>▪ When there are computed columns, all parsed columns in the expression must be listed in the <code>COPY</code> statement.</li> </ul> <p>Transformation restrictions:</p> <ul style="list-style-type: none"> <li>▪ Computed columns cannot be used in <code>COPY</code> expressions.</li> <li>▪ Raw data cannot be specified in the source for computed columns.</li> <li>▪ <code>COPY</code> expressions may contain only constants.</li> <li>▪ <code>FORMAT</code> cannot be specified for a computed column.</li> </ul> <p>Transformation usage:</p> <ul style="list-style-type: none"> <li>▪ If nulls are specified in the raw data for parsed columns in the source, evaluation follows the same rules as for expressions within SQL statements.</li> <li>▪ Parsed and computed columns can be interspersed in the</li> </ul>

	<p><code>COPY</code> statement.</p> <ul style="list-style-type: none"> <li>▪ Multiple columns can be specified in a <code>COPY</code> expression.</li> <li>▪ Multiple <code>COPY</code> expressions can refer to the same parsed column.</li> <li>▪ A <code>COPY</code> expression can be as simple as a single column and can be as complex as a case expression with multiple columns.</li> <li>▪ <code>COPY</code> expressions can be specified for columns of all supported data types.</li> <li>▪ <code>COPY</code> expressions can use most Vertica-supported SQL functions, operators, constants, NULLs, and comments, as follows: <b>date/time</b> (page 184) functions, <b>formatting</b> (page 241) functions, <b>numeric</b> (page 279) functions, <b>string</b> (page 314) functions, <b>null-handling</b> (page 299) functions, and <b>system information</b> (page 367) functions.</li> <li>▪ <code>COPY</code> expressions cannot use SQL meta functions (<b>Vertica-specific</b> (page 374)), <b>analytic</b> (page 128) functions, and <b>aggregate</b> (page 110) functions.</li> </ul>
<i>column</i>	<p>Restricts the load to one or more specified columns in the table. If no columns are specified, all columns are loaded by default. Table columns that are not in the column list are given their default values. If no default value is defined for a column, <code>COPY</code> inserts NULL.</p> <p>There is no implicit casting during parsing, so mismatched data types cause the <code>COPY</code> operation to roll back and the row to be rejected. For parsed columns, specify only raw data in the source.</p> <p><b>Tip:</b> If you leave this parameter blank to load all the columns in the table, you can use the optional parameter <code>COLUMN OPTION</code> to specify parsing options for specific columns.</p> <p><b>Note:</b> The data file must contain the same number of columns as the <code>COPY</code> command's column list. For example, in a table <code>T1</code> with nine columns (<code>C1</code> through <code>C9</code>), the following command would load the three columns of data in each record to columns <code>C1</code>, <code>C6</code>, and <code>C9</code>, respectively:</p> <pre>=&gt; COPY T1 (C1, C6, C9);</pre>

FILLER	<p>Instructs Vertica not to load a column and the fields it contains into the destination table. This is useful for omitting columns that you do not want to transfer into a table.</p> <p>Transforms data from a source column and then loads the transformed data to a destination table without loading the original, untransformed source column (parsed column). (See Transforming Data During Loads in the Administrator's Guide.)</p> <p>Filler requirements:</p> <ul style="list-style-type: none"> <li>▪ The data type of the filler column must be specified.</li> <li>▪ The name of the filler column must be unique across the source file and target table.</li> <li>▪ The filler column must be a parsed column, not a computed column.</li> </ul> <p>Filler restrictions:</p> <ul style="list-style-type: none"> <li>▪ The source columns in a <code>COPY</code> statement cannot consist of only filler columns.</li> <li>▪ Target table columns cannot be specified as filler whether they appear in the column list or not.</li> </ul> <p>Filler usage:</p> <ul style="list-style-type: none"> <li>▪ Expressions can contain filler columns.</li> <li>▪ There is no restriction on the number of filler columns that can be used in a <code>COPY</code> statement — other than at least one column must not be a filler column.</li> <li>▪ A data file can consist of only filler columns. This means that all data in a data file can be loaded into filler columns and then transformed and loaded into table columns.</li> <li>▪ All parser parameters can be specified for filler columns.</li> <li>▪ All statement level parser parameters apply to filler columns.</li> </ul>
FORMAT	<p>Is specified for <b><i>date/time</i></b> (page 72), and <b><i>binary</i></b> (page 65) data types.</p> <p>Supported date/time formats are the same as those accepted by the <b><i>TO_DATE</i></b> (page 243) function. For example:</p> <pre>=&gt; TO_DATE('05 Dec 2000', 'DD Mon YYYY')</pre> <p>If you specify invalid format strings, the <code>COPY</code> operation returns an error. See the following links for supported formats:</p> <ul style="list-style-type: none"> <li>▪ <b><i>Template Patterns for Date/Time Formatting</i></b> (page 248)</li> <li>▪ <b><i>Template Pattern Modifiers for Date/Time Formatting</i></b> (page 249)</li> <li>▪ Loading Data into Binary Data Types</li> </ul> <p>See Loading Data into Binary Data Types to learn more about using the date/time and binary data types..</p> <p><b>Note:</b> the <code>FORMAT</code> keyword significantly improves performance for loading <code>DATE</code> data types.</p>

<i>pathToData</i>	<p>Specifies the absolute path of the file containing the data, which can be from multiple input sources.</p> <p>Path can optionally contain wildcards to match more than one file. The file or files must be accessible to the host on which the COPY statement runs.</p> <p>You can use variables to construct the pathname as described in Using Load Scripts.</p> <p>The supported patterns for wildcards are specified in the <i>Linux Manual Page GLOB(7), Globbing pathnames</i> <a href="http://man-wiki.net/index.php/7:glob">http://man-wiki.net/index.php/7:glob</a>.</p>
<i>nodename</i>	<p>Is optional. If omitted, operations default to the query's initiator node.</p> <p><b>Note:</b> Nodename cannot be specified with <code>STDIN</code> because <code>STDIN</code> is read on the initiator node only.</p>
STDIN	<p>Reads from the client a standard input instead of a file. <code>STDIN</code> takes one input source only and is read on the initiator node. To load multiple input sources, use <i>pathToData</i>.</p>
BZIP GZIP UNCOMPRESSED	<p>Input files can be of any format. If wildcards are used, then all qualifying input files must be of the same format. <code>UNCOMPRESSED</code> is the default.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>▪ When using concatenated BZIP or GZIP files, be sure that each source file is terminated with a record terminator before you concatenate them.</li> <li>▪ Concatenated BZIP and GZIP files are not supported for <code>NATIVE (binary)</code> and <code>NATIVE VARCHAR</code> formats.</li> </ul>
WITH, AS	<p>For readability and have no effect.</p>
NATIVE	<p>Specifies that the data is in a binary-format file. Loading data through a binary-format file is often faster than normal text mode, because it does not require the use and processing of delimiters. This saves the database the extra work of converting integers, dates, and timestamps from text to their native storage format. Binary format data files can be bigger than their text equivalents, however, you can reduce the space usage by compressing binary data using <code>GZIP</code> or <code>BZIP</code>.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>▪ Native binary format loading can be used when developing plug-ins to ETL applications, as well as by batch inserts issued from ODBC and JDBC.</li> <li>▪ Binary-format files must meet exacting specifications, as per Creating Native-Format Files to Load Data in the Administrator's Guide.</li> <li>▪ You cannot mix Binary and ASCII source files in the same COPY statement.</li> <li>▪ Concatenated BZIP and GZIP files are not supported for <code>NATIVE (binary)</code>.</li> </ul> <p>See Advanced Formats For Loading Data in the Administrator's</p>

	Guide.
NATIVE VARCHAR	<p>Uses a similar file format to <code>NATIVE</code> (binary), but all fields are represented as strings in <code>CHAR</code> or <code>VARCHAR</code>. Conversion to the actual table data type is done on the database server; thus, <code>NATIVE VARCHAR</code> does not provide the same efficiency as <code>NATIVE</code>; however, <code>NATIVE VARCHAR</code> provides the convenience of not having to use delimiters or escape special characters, such as quotes, which can make working with client applications easier.</p> <p><b>Note:</b> Concatenated BZIP and GZIP files are not supported for <code>NATIVE VARCHAR</code> formats.</p> <p>See Advanced Formats For Loading Data in the Administrator's Guide.</p>
COLUMN OPTION	<p>A parsing option that allows a subset of columns to be specified through the table column list. For example, you can specify that a particular column has its own delimiter, enclosed by, null as <code>'NULL'</code> expression, and so on. You don't have to specify all the column names in the <code>COPY</code> column list, which can be especially useful for large tables with lots of columns.</p> <p><b>Note:</b> You cannot specify the format of a column not accounted for in the table column list.</p>
DELIMITER	<p>Is the single ASCII character that separates columns within each record of a file. You can choose any ASCII value in the range <code>E'\001'</code> to <code>E'\177'</code> inclusive (any ASCII character except <code>NULL: E'\000'</code>). The default in Vertica is a vertical bar (<code> </code>).</p> <p><b>Note:</b> A comma (<code>,</code>) is the delimiter commonly used in CSV data files.</p> <p>If the delimiter character appears in string of data values, use the <code>ESCAPE AS</code> character (<code>\</code> by default) to indicate that it is a literal. See Loading Data into Character Data Types.</p> <p>To specify a non-printing character, use either the extended string syntax (<code>E'...'</code>) or, if <code>StandardConformingStrings</code> is enabled, a Unicode string literal (<code>U&amp;'...'</code>). For example, to specify tab as the delimiter, you could use either <code>E'\t'</code> or <code>U&amp;'\\0009'</code>.</p>
TRAILING NULLCOLS	<p>Specifies that if Vertica encounters a short record, the missing columns are inserted with <code>NULLS</code>.</p> <p><b>Note:</b> Vertica verifies that there is no <code>NOT NULL</code> constraint on a column before inserting a <code>NULL</code>. If a <code>NOT NULL</code> constraint exists on the column, Vertica returns an error and rolls back the statement.</p>

ESCAPE AS	<p>Sets the escape character that prevents the following character from being interpreted as one of the special characters defined by the COPY command (the record terminator, delimiter, enclosed by, and the escape characters). When any of these special characters are preceded by the escape character, the COPY command ignores their special meaning and copies them into the database literally. Whether or not the character following the escape character is one of these special characters, the escape character is always removed from the input. If you want the escape character value to be inserting into your database, you must escape it. For example, if you leave the escape character as the default backslash (\) character, you need to use two backslashes (\\) anyplace in your input where you want a backslash to appear in the loaded data. The alternative (and perhaps easier) method is use ESCAPE AS to change the escape character to a value that does not appear in your input data.</p> <p>The default value for the escape character is backslash (\). You can set the escape character to be any ASCII value in the range E'\001' to E'\177' inclusive (any ASCII character except NULL: E'\000').</p> <p><b>Note:</b> The data read in by the COPY command is not interpreted as <i>string literals</i> (page 19), and therefore does not follow the same escape rules as SQL statements (including the arguments to the COPY command). Only the characters defined by ESCAPE AS, DELIMITER, ENCLOSED BY, and RECORD TERMINATOR are treated as special values in the data read by the COPY command.</p>
NO ESCAPE	<p>Eliminates escape character handling. Use this option if you do not need any character escaping and you want to prevent characters in your data from being misinterpreted as escape sequences.</p>
ENCLOSED BY	<p>Sets the quote character and allows delimiter characters to be embedded in string values. You can choose any ASCII value in the range E'\001' to E'\177' inclusive (any ASCII character except NULL: E'\000'). By default, ENCLOSED BY has no value, meaning data is not enclosed by any sort of quote character.</p> <p>Given the following input (with the default   DELIMITER) :</p> <pre>"vertica   value"</pre> <p>The default is:</p> <ul style="list-style-type: none"> <li>▪ Column 1 contains "vertica</li> <li>▪ Column 1 contains value"</li> </ul> <p>Notice the double quotes (") before vertica and after value.</p> <p>When you enable ENCLOSED BY, you can specify an ASCII character to enclose data. All ASCII characters are allowed except for a space. Double quote is the most commonly used quotation character.</p> <p>The following indicates that data in the input to COPY is within double quotes:</p> <pre>ENCLOSED BY '''</pre>

	<p>Using the following sample input, columns are distributed as follows:</p> <pre>"1", "vertica,value", ",", ""</pre> <ul style="list-style-type: none"> <li>▪ Column 1 contains 1</li> <li>▪ Column 2 contains vertica,value</li> <li>▪ Column 3 contains ,</li> <li>▪ Column 4 contains ''</li> </ul> <p>You could also write the above example using any ASCII character of your choosing:</p> <pre>~1~, ~vertica,value~, ~,~, ~'~</pre> <p>You can use single quote as the quote character, but you must escape it by using either the extended string syntax, a Unicode literal string if <code>StandardConformingStrings</code> is enabled, or by using four single quotes:</p> <pre>ENCLOSED BY E'\'' ENCLOSED BY U&amp;'\0027' ENCLOSED BY ''''</pre> <p>Using any of the above means the following input is properly parsed:</p> <pre>'1', 'vertica,value', ',', ''</pre> <p>See <b><i>String Literals (Character)</i></b> (page 21) for an explanation of the string literal formats you can use to specify the <code>ENCLOSED BY</code> parameter.</p> <p>Use the <code>ESCAPE AS</code> character to embed the <code>ENCLOSED BY</code> delimiter within character string values. For example, using the default <code>ESCAPE AS</code> character (<code>\</code>) and double quote as the <code>ENCLOSED BY</code> character, the following input returns "vertica"</p> <pre>"\"vertica\""</pre>
<p>NULL</p>	<p>The string that represents a null value. It can contain any ASCII values in the range <code>E'\001'</code> to <code>E'\177'</code> inclusive (any ASCII character except <code>NULL: E'\000'</code>). The default is an empty string (<code>'</code>).</p> <p>When <code>NULL</code> is an empty string (<code>'</code>), use quotes to insert an empty string instead of a <code>NULL</code>. For example, using <code>NULL " ENCLOSED BY '''</code>,</p> <ul style="list-style-type: none"> <li>▪ <code>1  3</code> — Inserts a <code>NULL</code> in the second columns.</li> <li>▪ <code>1 "" 3</code> — Inserts an empty string instead of a <code>NULL</code> in the second columns.</li> </ul> <p>To input an empty or literal string, use quotes (<code>ENCLOSED BY</code>); for example:</p> <pre>NULL '' NULL 'literal'</pre> <p>The null string is case-insensitive and must be the only value between the delimiters. For example, if the null string is <code>NULL</code> and the delimiter is the vertical bar (<code> </code>):</p> <pre> NULL </pre> <p>indicates a null value.</p>

	<p>  NULL   does not indicate a null value.</p> <p>When you use the COPY command in a script, you must substitute a double-backslash for each null string that includes a backslash. For example, the scripts used to load the example databases contain:</p> <pre>COPY ... NULL E'\\n' ...</pre>																																				
RECORD TERMINATOR	<p>Specifies the literal character string that indicates the end of a data file record. If you do not specify a value, then Vertica attempts to determine the correct line ending, accepting either just a linefeed (E'\n') common on UNIX systems, or a carriage return and linefeed (E'\r\n') common on Windows platforms. If you specify a RECORD TERMINATOR, you must be sure the input file matches, otherwise you may get inconsistent data loads.</p> <p>To specify non-printing characters as the RECORD TERMINATOR, use either the extended string syntax or Unicode string literals. The following table lists some common record terminator characters. See <i>String Literals</i> (page 21) for an explanation of the literal string formats.</p> <table border="1"> <thead> <tr> <th>Extended String Syntax</th> <th>Unicode Literal String</th> <th>Description</th> <th>ASCII Decimal</th> </tr> </thead> <tbody> <tr> <td>E'\a'</td> <td>U&amp;'\0007'</td> <td>Bell</td> <td>7</td> </tr> <tr> <td>E'\b'</td> <td>U&amp;'\0008'</td> <td>Backspace</td> <td>8</td> </tr> <tr> <td>E'\t'</td> <td>U&amp;'\0009'</td> <td>Horizontal tab</td> <td>9</td> </tr> <tr> <td>E'\n'</td> <td>U&amp;'\000a'</td> <td>Linefeed</td> <td>10</td> </tr> <tr> <td>E'\v'</td> <td>U&amp;'\000b'</td> <td>Vertical tab</td> <td>11</td> </tr> <tr> <td>E'\f'</td> <td>U&amp;'\000c'</td> <td>Formfeed</td> <td>12</td> </tr> <tr> <td>E'\r'</td> <td>U&amp;'\000d'</td> <td>Carriage return</td> <td>13</td> </tr> <tr> <td>E'\''</td> <td>U&amp;'\005c'</td> <td>Backslash</td> <td>92</td> </tr> </tbody> </table> <p>Note: The record terminator cannot be the same as DELIMITER, NULL, ESCAPE or ENCLOSED BY.</p>	Extended String Syntax	Unicode Literal String	Description	ASCII Decimal	E'\a'	U&'\0007'	Bell	7	E'\b'	U&'\0008'	Backspace	8	E'\t'	U&'\0009'	Horizontal tab	9	E'\n'	U&'\000a'	Linefeed	10	E'\v'	U&'\000b'	Vertical tab	11	E'\f'	U&'\000c'	Formfeed	12	E'\r'	U&'\000d'	Carriage return	13	E'\''	U&'\005c'	Backslash	92
Extended String Syntax	Unicode Literal String	Description	ASCII Decimal																																		
E'\a'	U&'\0007'	Bell	7																																		
E'\b'	U&'\0008'	Backspace	8																																		
E'\t'	U&'\0009'	Horizontal tab	9																																		
E'\n'	U&'\000a'	Linefeed	10																																		
E'\v'	U&'\000b'	Vertical tab	11																																		
E'\f'	U&'\000c'	Formfeed	12																																		
E'\r'	U&'\000d'	Carriage return	13																																		
E'\''	U&'\005c'	Backslash	92																																		
SKIP	Skips the first <i>n</i> records in each file in a load, which is useful if you want to omit table header information.																																				
REJECTMAX	<p>Sets an upper limit on the number of logical records to be rejected before a load fails. A rejection is data that could not be parsed into the corresponding data type during a bulk load. (It does not refer to referential constraints.)</p> <p>The limit on the number of logical records to be rejected is one less than the value specified for REJECTMAX. When the number of rejected records becomes equal to the value specified for REJECTMAX, the load fails and the failed records are placed into the reject file. If not specified or if value is 0, REJECTMAX allows an unlimited number of rejections.</p>																																				

	<p><b>Note:</b> Vertica does not accumulate rejected records across files or nodes while the data is loading. If one file exceeds the maximum reject number, the entire load fails.</p>
EXCEPTIONS	<p>Specifies the filename or absolute path in which to write messages indicating the input line number and the reason for each rejected data record. The default path is:</p> <pre>&lt;Catalog dir&gt;/CopyErrorLogs/&lt;tablename&gt;-&lt;filename of source&gt;-copy-from-exceptions</pre> <p>&lt;Catalog dir&gt; represents the directory in which the database catalog files are stored, and &lt;tablename&gt;-&lt;filename of source&gt; are the names of the table and data file. If copying from STDIN, the &lt;filename of source&gt; is STDIN.</p> <p><b>Note:</b> Filename is required because of multiple input files. Also, long table names combined with long data file names can exceed the operating system's maximum length (typically 255 characters). To work around this limitation, specify a path for the exceptions file that is different from the default path; for example, <code>\tmp\&lt;shorter-file-name&gt;</code>.</p> <p>If exceptions files are not specified:</p> <ul style="list-style-type: none"> <li>▪ If there is one data source file (<i>pathToData</i> or <code>STDIN</code>), all information is stored as one file in the default directory.</li> <li>▪ If there are multiple data files, all information is stored as separate files, one for each data file in default directory.</li> </ul> <p>If exception files are specified:</p> <ul style="list-style-type: none"> <li>▪ If there is one data file, path is treated as a file with all information stored in this file. If path is not a file, then the system returns an error.</li> <li>▪ If there are multiple data files, path is treated as a directory with all information stored in separate files, one for each data file in <i>this</i> directory. If path is not a directory, then the system returns an error.</li> <li>▪ Exceptions files are not shipped to the initiator node.</li> <li>▪ Only one path per node is accepted. If more than one is provided, Vertica returns an error.</li> <li>▪ The format for the <code>EXCEPTIONS</code> file is: <ul style="list-style-type: none"> <li>▪ <code>COPY: Input record &lt;num&gt; in &lt;pathofinputfile&gt; has been rejected (&lt;reason&gt;). Please see &lt;pathtorejectfile&gt;, record &lt;recordnum&gt; for the rejected record.</code></li> </ul> </li> </ul>
REJECTED DATA	<p>Specifies the filename or absolute path in which to write rejected rows. This file can then be edited to resolve problems and reloaded. The default path is:</p>

	<p>&lt;Catalog dir&gt;/CopyErrorLogs/&lt;tablename&gt;-&lt;filename of source&gt;-copy-from-data</p> <p>&lt;Catalog dir&gt; represents the directory in which the database catalog files are stored, and &lt;tablename&gt;-&lt;filename of source&gt; are the names of the table and data file. If copying from STDIN, the &lt;filename of source&gt; is STDIN.</p> <p><b>Notes:</b> Filename is required because of multiple input files. Also, long table names combined with long data file names can exceed the operating system's maximum length (typically 255 characters). To work around this limitation, specify a path for the rejected data file that is different from the default path; for example, <code>\tmp\&lt;shorter-file-name&gt;</code>.</p> <p>If rejected data files are not specified:</p> <ul style="list-style-type: none"> <li>▪ If there is one data source file (<i>pathToData</i> or <code>STDIN</code>), all information is stored as one file in the default directory.</li> <li>▪ If there are multiple data files, all information is stored as separate files, one for each data file in default directory.</li> </ul> <p>If rejected data files are specified:</p> <ul style="list-style-type: none"> <li>▪ If there is one data file, path is treated as a file with all information stored in this file. If path is not a file, then the system returns an error.</li> <li>▪ If there are multiple data files, path is treated as a directory, with all information stored in separate files, one for each data file in <i>this</i> directory. If path is not a directory, then the system returns an error.</li> <li>▪ Rejected data files are not shipped to the initiator node.</li> <li>▪ Only one path per node is accepted. If more than one is provided, Vertica returns an error.</li> </ul>
ENFORCELENGTH	<p>Rejects rows that do not fit into the target table instead of truncating them. This can occur with column types of char, varchar, binary, and varbinary.</p> <p>For example, if 'abc' is loaded into <code>VARCHAR(2)</code> it is automatically truncated to 'ab' and loaded. Using <code>ENFORCELENGTH</code> causes the 'abc' to be rejected.</p> <p><b>Note:</b> <code>COPY</code> has a hard 65K character limit on the length of <code>NATIVE</code> and <code>NATIVE VARCHAR</code> data. If it encounters a value that is longer than this length, it always rejects the row, even if <code>ENFORCELENGTH</code> is disabled.</p>
ABORT ON ERROR	<p>Stops the <code>COPY</code> command if a row is rejected and rolls back the command. No data is loaded.</p>
AUTO   DIRECT   TRICKLE	<p>Specifies how data is loaded into the database.</p> <ul style="list-style-type: none"> <li>▪ <code>AUTO</code> (the default) loads data into the WOS (Write Optimized Store) until it is full, then it loads directly into ROS (Read Optimized Store) containers.</li> <li>▪ <code>DIRECT</code> bypasses the WOS and loads data into ROS containers. This option is best suited when you are loading large amounts of data (100MB or more) at a time. Using</li> </ul>

	<p><code>DIRECT</code> for many loads of smaller data sets results in many ROS containers, which have to be combined later.</p> <ul style="list-style-type: none"> <li>▪ <code>TRICKLE</code> loads data only into the WOS. If the WOS becomes full, an error occurs and the entire data load is rolled back. This option is more efficient than <code>AUTO</code> when loading data into partitioned tables. Use this option only when you have a finely-tuned load and moveout process so you can be sure there is room in the WOS for the data you are loading.</li> </ul>
STREAM NAME	<p>Is the optional identifier that names a stream, which could be useful for quickly identifying a particular load.</p> <p><code>STREAM NAME</code> appears in the <code>stream</code> column of the <code>LOAD_STREAMS</code> (page 856) table.</p> <p>By default, Vertica names streams by table and file name. For example, if you have two files (f1, f2) in Table A, stream names would appear as A-f1, A-f2, etc.</p> <p>Use the following statement to name a stream:</p> <pre>=&gt; COPY &lt;mytable&gt; FROM &lt;myfile&gt; DELIMITER ' ' DIRECT STREAM NAME 'My stream name';</pre>
NO COMMIT	<p>Prevents the <code>COPY</code> statement from automatically committing its transaction when it finishes copying data. You may want to use this option when sequentially performing several copy statements, so that all of the copied data is either committed or rolled back as a single transaction. Using a single transaction for multiple <code>COPY</code> statements also allows Vertica to more efficiently load the data since it can combine the larger amount of data from the <code>COPY</code> statements into fewer ROS containers.</p> <p><code>NO COMMIT</code> can be combined with any other existing <code>COPY</code> option, and all the usual transaction semantics apply.</p> <p>If there is a transaction in progress initiated by a statement other than <code>COPY</code> (for example, <code>INSERT</code>), <code>COPY... NO COMMIT</code> adds rows to the existing transaction instead of starting a new one. The previous statements are NOT committed.</p> <p><b>Tip:</b> Use the <code>NO COMMIT</code> keywords to incorporate detection of constraint violations into the load process. Vertica checks for violations when queries are run, not when data is loaded. To avoid constraint violations, load data without committing it and then perform a post-load check of your data using the <b><code>ANALYZE_CONSTRAINTS</code></b> (page 377) function. If the function finds constraint violations, you can easily roll back the load because you have not committed it.</p>

## COPY Formats

The following `COPY` options are available when loading all data formats (delimited text, `NATIVE` (binary) and `NATIVE VARCHAR`):

- COLUMN OPTION
- DIRECT
- ENFORCELENGTH
- EXCEPTIONS
- FILLER
- REJECTED DATA
- ABORT ON ERROR
- STREAM NAME
- SKIP
- REJECTMAX
- STORAGE
- STDIN
- BZIP|GZIP|UNCOMPRESSED

**NO COMMIT** The following option is available when loading data in delimited text or **NATIVE VARCHAR** format:

**FORMAT** The following options are only available when loading delimited text:

- NULL
- DELIMITER
- ENCLOSED BY
- ESCAPE AS
- TRAILING NULLCOLS
- RECORD TERMINATOR

## Notes

- The data read in by the COPY command is not interpreted as **string literals** (page 19), and therefore does not follow the same escape rules as SQL statements (including the arguments to the COPY command). Only the characters defined by ESCAPE AS, DELIMITER, ENCLOSED BY, and RECORD TERMINATOR are treated as special values in the data read by the COPY command.
- To prevent excessive resource usage, COPY is limited to loading 50 files per node at a time.
- The COPY command automatically commits itself and any current transaction unless **NO COMMIT** is specified and unless the tables are temp tables. Vertica recommends that you **COMMIT** (page 605) or **ROLLBACK** (page 744) the current transaction before you use COPY.
- You cannot use the same character in both the DELIMITER and NULL strings.
- String data in load files is considered to be all characters between the specified delimiters. Do not enclose character strings in quotes. In other words, quote characters are treated as ordinary data.
- Invalid input is defined as:
  - Missing columns (too few columns in an input line).
  - Extra columns (too many columns in an input line).

- Empty columns for INTEGER or DATE/TIME data types. COPY does not use the default data values defined by the **CREATE TABLE** (page 664) command.
- Incorrect representation of data type. For example, non-numeric data in an INTEGER column is invalid.
- Empty values (two consecutive delimiters) are accepted as valid input data for CHAR and VARCHAR data types. Empty columns are stored as an empty string ( ' ' ), which is not equivalent to a null string.
- When an empty line is encountered during load, it is neither inserted nor rejected. However, the record number is incremented. Bear this in mind when you evaluate lists of rejected records. If you return a list of rejected records and one empty row was encountered during load, the position of rejected records is bumped up one position.
- Canceling a COPY statement rolls back all rows loaded by that statement.
- If you are using JDBC, Vertica recommends that you use the following value for the RECORD TERMINATOR:  
`System.getProperty("line.separator")`
- Named pipes are supported. Naming conventions have the same rules as file names on the given file system. Permissions are open, write, and close.
- The following parameters can be specified on either a statement or on a per-column basis: DELIMITER, ENCLOSED BY, ESCAPE AS, and NULL. The same rules apply whether the parameter is specified at the statement or column level. Column-level parameters override statement-level parameters. If no column-level parameter is specified, the statement-level parameter is used. If neither a column-level nor statement-level parameter is specified, the default is used.

## Examples

### Basic Examples

The following examples specify FORMAT, DELIMITER, NULL and ENCLOSED BY strings.

```
=> COPY public.customer_dimension (customer_since FORMAT 'YYYY')
    DELIMITER ','
    NULL AS 'null'
    ENCLOSED BY ''
=> COPY store.store_dimension
    FROM :input_file
    DELIMITER '|'
    NULL ''
    RECORD TERMINATOR E'\f'
=> COPY a
    FROM stdin
    DELIMITER ','
    NULL E'\\N'
    DIRECT;
```

## Changing the Escape Character

If your input contains backslash characters (\) that you want to be read as data rather than as escape sequences, you can change the escape character to some other character that does not appear in your input (such as a control character):

```
=> COPY mytable FROM '/data/input.txt' ESCAPE AS E('\001');
```

## Loading Comma Separated Values

If you have a file containing comma-separated values that are terminated by line feeds, you can use the following command:

```
=> COPY mytable FROM STDIN DELIMITER ',' RECORD TERMINATOR E'\n';
```

## Specifying Quote Characters for a Single Column

The following example sets a single column to be enclosed by double quotes, rather than the entire row.

```
=> COPY Retail.Dim (Dno, Dname ENCLOSED BY '"', Dstore)
    FROM '/home/dbadmin/dim3.txt'
    EXCEPTIONS '/home/dbadmin/exp.txt'
    DELIMITER ',';
```

This example properly reads data such as:

```
123,"Smith, John",9832
```

## Loading Binary Data Through Delimiters

In the following example create a table that loads a different binary format for each column and insert the same value, the byte sequence {0x61,0x62,0x63,0x64,0x65}.

```
=> CREATE TABLE t(
    oct VARBINARY(5),
    hex VARBINARY(5),
    bitstring VARBINARY(5) );
```

Create the projection:

```
=> CREATE PROJECTION t_p(oct, hex, bitstring) AS SELECT * FROM t;
```

Issue the COPY command. Note that the copy is from STDIN, not a file.

```
=> COPY t (oct FORMAT 'octal',
    hex FORMAT 'hex',
    bitstring FORMAT 'bitstring')
    FROM STDIN DELIMITER ',';
```

Enter the data to be copied, and end it with a backslash and a period on a line by itself:

```
>> 141142143144145,0x6162636465,0110000101100010011000110110010001100101
>> \.
```

Now query table t to see the inputs:

```
=> SELECT * FROM t;
oct    | hex    | bitstring
-----+-----+-----
```

```
abcde | abcde | abcde
(1 row)
```

For more information, see Loading Data into Binary Data Types.

### Using Compressed Data and Named Pipes

The following command creates the named pipe, *pipe1*:

```
\! mkfifo pipe1
\set dir `pwd`/
\set file ''':dir'pipe1'''
```

The following sequence copies an uncompressed file from the named pipe:

```
\! cat pf1.dat > pipe1 &
COPY fact FROM :file delimiter '|';
SELECT * FROM fact;
COMMIT;
```

The following statement copies a GZIP file from named pipe and uncompresses it:

```
\! gzip pf1.dat
\! cat pf1.dat.gz > pipe1 &
COPY fact FROM :file ON site01 GZIP delimiter '|';
SELECT * FROM fact;
COMMIT;
\!gunzip pf1.dat.gz
```

The following COPY command copies a BZIP file from named pipe and then uncompresses it:

```
\!bzip2 pf1.dat
\! cat pf1.dat.bz2 > pipe1 &
COPY fact FROM :file ON site01 BZIP delimiter '|';
SELECT * FROM fact;
COMMIT;
bunzip2 pf1.dat.bz2
```

### User-specified Exceptions and Rejected Data

```
\set dir `pwd`/data/
\set remote_dir /scratch_b/qa/tmp_ms/
```

Reject/Exception files NOT specified. The inputs are multiple files, and exceptions and rejection files go to the default directory on each node:

```
\set file1 ''':dir'C1_fact.dat''
\set file2 ''':dir'C2_fact.dat''
\set file3 ''':remote_dir'C3_fact.dat''
\set file4 ''':remote_dir'C4_fact.dat''
COPY fact FROM :file1 ON site01,
               :file2 ON site01,
               :file3 ON site02,
               :file4 ON site02
DELIMITER '|';
```

Reject/Exception files SPECIFIED. Input is a single file on the initiator, and the exceptions and rejected data are file names instead of directories:

```
\set except_s1 ''':dir'exceptions'''
\set reject_s1 ''':dir'rejections'''
```

```
COPY fact FROM :file1 ON site01
DELIMITER '|'
REJECTED DATA :reject_s1 ON site01
EXCEPTIONS :except_s1 ON site01;
```

**Reject/Exception files SPECIFIED. A single file is on remote node:**

```
\set except_s2 ''':remote_dir'exceptions'''
\set reject_s2 ''':remote_dir'rejections'''
```

```
COPY fact FROM :file1 ON site02
DELIMITER '|'
REJECTED DATA :reject_s2 ON site02
EXCEPTIONS :except_s2 ON site02;
```

**Reject/Exception files SPECIFIED. Multiple data files on multiple nodes, with rejected data and exceptions referring to the directory on which the files reside:**

```
\set except_s1 ''':dir''''
\set reject_s1 ''':dir''''
\set except_s2 ''':remote_dir''''
\set reject_s2 ''':remote_dir''''
COPY fact FROM :file1 ON site01,
               :file2 ON site01,
               :file3 ON site02,
               :file4 ON site02
DELIMITER '|'
REJECTED DATA :reject_s1 ON site01, :reject_s2 ON site02
EXCEPTIONS :except_s1 ON site01, :except_s2 ON site02;
```

### Loading NULL values

You can specify NULL values by entering fields in a data file without content. For example, given the default delimiter (|) and default NULL (empty string), the following inputs:

```
| | 1
| 2 | 3
4 | | 5
6 | |
```

are inserted into the table as follows:

```
(null, null, 1)
(null, 2, 3)
(4, null, 5)
(6, null, null)
```

If NULL is set as a literal ('null'), the following inputs:

```
null | null | 1
null | 2 | 3
4 | null | 5
6 | null | null
```

are inserted into the table as follows:

```
(null, null, 1)
(null, 2, 3)
(4, null, 5)
(6, null, null)
```

### Using Trailing NULL Columns

The following illustrates how trailing null columns handles a short record:

```
=> CREATE TABLE z (
    a INT,
    b INT,
    c INT );
```

Insert some values:

```
=> INSERT INTO z VALUES (1, 2, 3);
```

Query table z to see the inputs:

```
=> SELECT * FROM z;
 a | b | c
---+---+---
 1 | 2 | 3
(1 row)
```

Now insert data using the STDIN keyword:

```
=> COPY z FROM STDIN TRAILING NULLCOLS;
>> 4 | 5 | 6
>> 7 | 8
>> \.
```

```
=> SELECT * FROM z;
 a | b | c
---+---+---
 1 | 2 | 3
 4 | 5 | 6
 7 | 8 |
(3 rows)
```

The following example shows what happens when you try to use a trailing null column on a column that contains a NOT NULL constraint:

```
=> CREATE TABLE n (
    a INT,
    b INT NOT NULL,
    c INT );
=> INSERT INTO n VALUES (1, 2, 3);
=> SELECT * FROM n;
 a | b | c
---+---+---
 1 | 2 | 3
(1 row)
=> COPY n FROM STDIN trailing nullcols abort on error;
>> 4 | 5 | 6
>> 7 | 8
>> 9
```

```
>> \.
ERROR: COPY: Input record 3 has been rejected (Cannot set trailing column to
NULL as column 2 (b) is NOT NULL)
=> SELECT * FROM n;
 a | b | c
---+---+---
 1 | 2 | 3
(1 row)
```

## Transforming Data

The following example derives and loads values for the year, month, and day columns in the target database based on the timestamp column in the source database. It also loads the parsed column, timestamp, from the source database to the target database.

```
=> CREATE TABLE t (
    year VARCHAR(10),
    month VARCHAR(10),
    day VARCHAR(10),
    k TIMESTAMP );
=> CREATE PROJECTION tp (
    year,
    month,
    day,
    k )
AS SELECT * FROM t;
=> COPY t (year AS TO_CHAR(k, 'YYYY'),
    month AS TO_CHAR(k, 'Month'),
    day AS TO_CHAR(k, 'DD'),
    k FORMAT 'YYYY-MM-DD') FROM STDIN NO COMMIT;
>> 2009-06-17
>> 1979-06-30
>> 2007-11-26
>> \.
=> SELECT * FROM t;
```

year	month	day	k
2009	June	17	2009-06-17 00:00:00
1979	June	30	1979-06-30 00:00:00
2007	November	26	2007-11-26 00:00:00

(3 rows)

## Ignoring Columns and Fields in the Load File

The following example derives and loads the value for the `TIMESTAMP` column in the target database from the year, month, and day columns in the source input. The year, month, and day columns are not loaded because the `FILLER` keyword skips them.

```
=> CREATE TABLE t (k TIMESTAMP);
=> CREATE PROJECTION tp (k) AS SELECT * FROM t;
=> COPY t (year FILLER VARCHAR(10),
    month FILLER VARCHAR(10),
    day FILLER VARCHAR(10),
    k AS TO_DATE(YEAR || MONTH || DAY, 'YYYYMMDD') )
FROM STDIN NO COMMIT;
```

```
>> 2009|06|17
>> 1979|06|30
>> 2007|11|26
>> \.
=> SELECT * FROM t;
      k
-----
2009-06-17 00:00:00
1979-06-30 00:00:00
2007-11-26 00:00:00
(3 rows)
```

## Specifying Parsing Options for a Column

First create a simple table.

```
=> CREATE TABLE t(
    pk INT,
    col1 VARCHAR(10),
    col2 VARCHAR(10),
    col3 VARCHAR(10),
    col4 TIMESTAMP);
```

Now use the `COLUMN OPTION` parameter to change the `col1` default delimiter to a tilde (~).

```
=> COPY t COLUMN OPTION(col1 DELIMITER '~') FROM STDIN NO COMMIT;
>> 1|ee~gg|yy|1999-12-12
>> \.

=> SELECT * FROM t;
 pk | col1 | col2 | col3 |          col4
-----+-----+-----+-----+-----
  1 | ee   | gg   | yy   | 1999-12-12 00:00:00
(1 row)
```

## See Also

***LCOPY*** (page 731)

***SQL Data Types*** (page 64)

***ANALYZE CONSTRAINTS*** (page 377)

Loading and Modifying Data in the Administrator's Guide

## COPY FROM VERTICA

Copies data from another Vertica database. You must have already established a connection to the other Vertica database using the ***CONNECT*** (page 605) statement. This statement works similarly to the ***COPY*** (page 607) statement, but has a more limited set of parameters.

### Syntax

```
COPY [target_schema.]target_table
... [(target_column_name[,target_column_name2,...])]
... FROM VERTICA database. [source_schema.]source_table
```

```

... [(source_column_name[, source_column_name2,...])]
... [AUTO | DIRECT | TRICKLE]
... [STREAM NAME 'stream name']
... [NO COMMIT]

```

## Parameters

[target_schema.]target_table	The table to store the copied data. This table must be in your local database, and must already exist.
(target_column_name[, target_column_name2,...])	A list of columns in the target table to store the copied data.
database	The name of the database that is the source of the copied data. You must have already created a connection to this database in the current session.
[source_schema.]source_table	The table in the source database that is the source of the copied data.
(source_column_name[, source_column_name2,...])	A list of columns in the source table to be copied. If this list is supplied, only these columns are copied from the source table.
AUTO   DIRECT   TRICKLE	Specifies how data is loaded into the database. <ul style="list-style-type: none"> <li>▪ <b>AUTO</b> (the default) loads data into the WOS (Write Optimized Store) until it is full, then it loads directly into ROS (Read Optimized Store) containers.</li> <li>▪ <b>DIRECT</b> bypasses the WOS and loads data into ROS containers. This option is best suited when you are loading large amounts of data (100MB or more) at a time. Using <b>DIRECT</b> for many loads of smaller data sets results in many ROS containers, which have to be combined later.</li> <li>▪ <b>TRICKLE</b> loads data only into the WOS. If the WOS becomes full, an error occurs and the entire data load is rolled back. This option is more efficient than <b>AUTO</b> when loading data into partitioned tables. Use this option only when you have a finely-tuned load and moveout process so you can be sure there is room in the WOS for the data you are loading.</li> </ul>
NO COMMIT	Prevents the COPY statement from automatically committing its transaction when it finishes copying data. You may want to use this option when sequentially performing several copy statements, so that all of the copied data is either committed or rolled back as a single transaction. Using a single transaction for multiple COPY statements also allows Vertica to more efficiently load the data since it can combine the larger amount of data from

	<p>the COPY statements into fewer ROS containers.</p> <p>NO COMMIT can be combined with any other existing COPY option, and all the usual transaction semantics apply.</p> <p>If there is a transaction in progress initiated by a statement other than COPY (for example, INSERT), COPY... NO COMMIT adds rows to the existing transaction instead of starting a new one. The previous statements are NOT committed.</p> <p><b>Tip:</b> Use the NO COMMIT keywords to incorporate detection of constraint violations into the load process. Vertica checks for violations when queries are run, not when data is loaded. To avoid constraint violations, load data without committing it and then perform a post-load check of your data using the <b>ANALYZE CONSTRAINTS</b> (page 377) function. If the function finds constraint violations, you can easily roll back the load because you have not committed it.</p>
--	---

## Notes

- The user account that was used to make the connection to the source database must have SELECT privileges on the source table, and the user running the COPY FROM VERTICA statement must have INSERT privileges for the destination table.
- If you do not supply a list of source and destination columns, COPY FROM VERTICA attempts to match columns in the source table with corresponding columns in the destination table. See the following section for details.

## Source and Destination Column Mapping

The COPY FROM VERTICA statement needs to map columns in the source table to columns in the destination table. You can optionally supply lists of either source columns to be copied, columns in the destination table where data should be stored, or both. These lists let you choose a subset of the columns in the source table to be copied to the destination table. Since neither of these lists are required, there are a few different behaviors depending on which list (if any) are present, as explained in the following table.

<b>No Source Column List</b>	<b>Source Column List Supplied</b>

<b>No Destination Column List</b>	All columns in the source table are matched to columns in the destination table. The number of columns in the two tables do not have to match, but the destination table must have at least as many columns as the source.	Only the content from the columns in the source list are exported. Columns in the destination table are matched to columns in the source list. The destination table must have at least the number of columns in the source list. It can have more columns.
<b>Destination Column List Supplied</b>	The columns in the destination column list are matched to columns in the source. The number of columns in the destination list must match the number of columns in the source table.	Columns are matched in the to column lists. The lists must have the same number of columns.

### Example

This example demonstrates connecting to another database, copying the contents of an entire table from the source database to an identically-defined table in the current database directly into ROS, and then closing the connection.

```
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> COPY customer_dimension FROM VERTICA vmart.customer_dimension DIRECT;
  Rows Loaded
-----
          500000
(1 row)
=> DISCONNECT vmart;
DISCONNECT
```

This example demonstrates copying several columns from a table in the source database into a table in the local database.

```
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> COPY people (name, gender, age) FROM VERTICA
-> vmart.customer_dimension (customer_name, customer_gender,
-> customer_age);
  Rows Loaded
-----
          500000
(1 row)
=> DISCONNECT vmart;
DISCONNECT
```

## CREATE FUNCTION (SQL Macros)

Lets you store SQL expressions as functions in Vertica for use in queries. Called SQL Macros, these functions are useful for executing complex queries or combining Vertica built-in functions. You simply call the function name you assigned.

### Syntax

```
CREATE [ OR REPLACE ] FUNCTION
... [ schema-name. ] function-name ( [ argname argtype [, ...] ] )
```

```

... RETURN rettype
... AS
... BEGIN
..... RETURN expression;
... END;

```

## Parameters

<code>[<i>schema-name</i>.]<i>function-name</i></code>	Specifies a name for the function (SQL Macro) to create. When using more than one schema, specify the schema that contains the function.
<code><i>argname</i></code>	Specifies the name of the argument.
<code><i>argtype</i></code>	Specifies the data type for argument that is passed to the function. Argument types must match Vertica type names. See <b>SQL Data Types</b> (page 64).
<code><i>rettype</i></code>	Specifies the data type to be returned by the function.
<code>RETURN <i>expression</i>;</code>	Specifies the SQL Macro (function body), which must be in the form of 'RETURN <i>expression</i> .' <i>expression</i> can contain built-in functions, operators, and argument names specified in the CREATE FUNCTION statement. A semicolon at the end of the expression is required. <b>Note:</b> Only one RETURN <i>expression</i> is allowed in the CREATE FUNCTION definition. FROM, WHERE, GROUP BY, ORDER BY, LIMIT, aggregation, analytics and meta function are not allowed.

## Notes

- A SQL Macro can be used anywhere in a query where an ordinary SQL expression can be used, except in the table partition clause or the projection segmentation clause.
- SQL Macros are flattened in all cases, including DDL. See Flattening FROM Clause Subqueries and Views in the Programmer's Guide.
- You can **create views** (page 697) on the queries that use SQL Macros and then query the views. When you create a view, a SQL Macro replaces a call to the user-defined function with the function body in a view definition. Therefore, when the body of the user-defined function is replaced, the view should also be replaced.
- If you want to change the body of a SQL Macro, use the CREATE OR REPLACE syntax. The command replaces the function with the new definition. If you change only the argument name or argument type, the system maintains both versions under the same function name. See **Examples** section below.
- If multiple SQL Macros with same name and argument type are in the search path, the first match is used when the function is called.
- The strictness and volatility (stable, immutable, or volatile) of a SQL Macro are automatically inferred from the function's definition. Vertica then performs constant folding optimization, when possible, and determines the correctness of usage, such as where an immutable function is expected but a volatile function is provided.

- You can return a list of all SQL Macro functions by querying the system table `V_CATALOG.USER_FUNCTIONS` (page 822) and executing the vsql meta-command `\df`. Users see only the functions on which they have `EXECUTE` privileges.

### Permissions

- To create a SQL Macro, a user must have `CREATE` privileges on the schema.
- To use a SQL Macro the user must have `USAGE` privileges on the schema and `EXECUTE` privileges on the defined function. See ***GRANT (Function)*** (page 721) and ***REVOKE (Function)*** (page 735).
- Only the superuser or the SQL Macro owner can ***drop*** (page 700) or ***alter*** (page 578) a SQL Macro.

### Example

This following statement creates a SQL function called `zeroifnull` that accepts an `INTEGER` argument and returns an `INTEGER` result.

```
=> CREATE FUNCTION zeroifnull(x INT) RETURN INT
    AS BEGIN
        RETURN (CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END);
    END;
```

You can use the new SQL function (`zeroifnull`) anywhere you use an ordinary SQL expression. For example, create a simple table:

```
=> CREATE TABLE tabwnulls(col1 INT);
=> INSERT INTO tabwnulls VALUES(1);
=> INSERT INTO tabwnulls VALUES(NULL);
=> INSERT INTO tabwnulls VALUES(0);
=> SELECT * FROM tabwnulls;
```

```
a
-----
1
0
(3 rows)
```

Use the `zeroifnull` function in a `SELECT` statement, where the function calls `col1` from table `tabwnulls`:

```
=> SELECT zeroifnull(col1) FROM tabwnulls;
zeroifnull
```

```
-----
1
0
0
(3 rows)
```

Use the `zeroifnull` function in the `GROUP BY` clause:

```
=> SELECT COUNT(*) FROM tabwnulls GROUP BY zeroifnull(col1);
count
-----
2
1
(2 rows)
```

If you want to change a SQL Macro's body, use the `CREATE OR REPLACE` syntax. The following command modifies the `CASE` expression:

```
=> CREATE OR REPLACE FUNCTION zeroifnull(x INT) RETURN INT
  AS BEGIN
    RETURN (CASE WHEN (x IS NULL) THEN 0 ELSE x END);
  END;
```

To see how this information is stored in the Vertica catalog, see [Viewing Information About SQL Macros](#) in the <SQL\_PROGRAMMERS\_GUIDE>.

### See Also

**ALTER FUNCTION** (page 578)

**DROP FUNCTION** (page 700)

**GRANT (Function)** (page 721)

**REVOKE (Function)** (page 735)

**V\_CATALOG.USER\_FUNCTIONS** (page 822)

Using SQL Macros in the Programmer's Guide

## CREATE FUNCTION (UDF)

Adds a User Defined Scalar Function (UDSF) stored in a shared Linux library to the catalog. You must have already loaded this library using the **CREATE LIBRARY** (page 634) statement. When you call the SQL function, Vertica passes the parameters to the code in the library to process it.

### Syntax

```
CREATE [ OR REPLACE ] FUNCTION [ schema-name.]function-name
... AS LANGUAGE 'language' NAME 'factory' LIBRARY library_name
... [ IMMUTABLE | STABLE | VOLATILE ]
... [ CALLED ON NULL INPUT | RETURN NULL ON NULL INPUT | STRICT ];
```

### Parameters

[ OR REPLACE ]	If you do not supply this parameter, the CREATE FUNCTION statement fails if an existing function matches the name and parameters of the function you are trying to define. If you do supply this parameter, the new function definition overwrites the old.
[ <i>schema-name.</i> ] <i>function-name</i>	The name of the function to create. If the function name is schema-qualified, the function is created in the specified schema. This name does not need to match the name of the factory, but it's much less confusing if they are the same or similar.
LANGUAGE ' <i>language</i> '	The programming language used to develop the function. Currently only 'C++' is supported.
NAME ' <i>factory</i> '	The name of the factory class in the shared library that

	generates the object to handle the function's processing.
<code>LIBRARY <i>library_name</i></code>	The name of the shared library that contains the C++ object to perform the processing for this function. This library must have been previously loaded using the <b>CREATE LIBRARY</b> (page 634) statement.
<code>[ IMMUTABLE   STABLE   VOLATILE ]</code>	<p>Sets the volatility of the function:</p> <ul style="list-style-type: none"> <li>▪ IMMUTABLE means that repeated calls to the function with the same input always returns the same output.</li> <li>▪ STABLE means that repeated calls to the function with the same input <i>within the same statement</i> returns the same output. For example, a function that returns the current user name would be stable since the user cannot change within a statement, but could change between statements.</li> <li>▪ VOLATILE means that repeated calls to the function with the same input can result in different output.</li> </ul> <p>If not supplied, Vertica assumes that the function is VOLATILE, and needs to be called for each invocation.</p> <p>Setting the volatility of the function helps Vertica optimize expressions. For example, if a function is IMMUTABLE, its results can be cached.</p> <p><b>Caution:</b> specifying IMMUTABLE or STABLE when the function is actually VOLATILE can result in incorrect or inconsistent answers.</p>
<code>[ CALLED ON NULL INPUT   RETURN NULL ON NULL INPUT   STRICT ]</code>	<p>Sets the null behavior of the function:</p> <ul style="list-style-type: none"> <li>• CALLED ON NULL INPUT means that even if all parameters passed to the function are NULL&lt; the function can return a non-null value.</li> <li>• RETURN NULL ON NULL INPUT and STRICT mean that the function returns NULL if any of its input is NULL.</li> </ul> <p>If not specified, Vertica assumes that the function is CALLED ON NULL INPUT and must be called even if its input values are NULL.</p> <p>Setting the NULL behavior of a function allows Vertica to optimize expressions. For example, if the function is set to STRICT, it does not need to be called if one of the input parameters is NULL.</p>

## Notes

- The parameters and return value for the function are automatically determined by the CREATE FUNCTION statement, based on data supplied by the factory class.
- Multiple functions can share the same name if they have different parameters. When you call a multiply-defined function, Vertica selects the UDF function whose input parameters match the parameters in the function call to perform the processing. This behavior is similar to having multiple signatures for a method or function in other programming languages.

- You can return a list of all SQL Macro functions and UDFs by querying the system table `V_CATALOG.USER_FUNCTIONS` (page 822) or executing the vsql meta-command `\df`. Users see only the functions on which they have `EXECUTE` privileges.

### Permissions

- Only the superuser can create or drop a function that uses a UDF library.
- To use a UDF, the user must have `USAGE` privileges on the schema and `EXECUTE` privileges on the defined function. See ***GRANT (Function)*** (page 721) and ***REVOKE (Function)*** (page 735).

### Example

The following example demonstrates loading a library named `scalarfunctions.so`, then defining a function named `Add2ints` that is mapped to the `Add2intsInfo` factory class in the library:

```
=> CREATE LIBRARY ScalarFunctions AS '/home/dbadmin/scalarfunctions.so';
CREATE LIBRARY
=> CREATE FUNCTION Add2Ints AS LANGUAGE 'C++' NAME 'Add2IntsInfo' LIBRARY
ScalarFunctions;
CREATE FUNCTION
=> \x
Expanded display is on.
=> SELECT * FROM USER_FUNCTIONS;
-[ RECORD 1 ]-----+-----
schema_name          | public
function_name        | Add2Ints
function_return_type | int
function_argument_type | Integer, Integer
function_definition  | Func 'Add2intsInfo' in Lib 'public.ScalarFunctions'
volatility            | volatile
is_strict             | f

=> \x
Expanded display is off.
=> -- Try a simple call to the function
=> SELECT Add2Ints(23,19);
   Add2Ints
-----
          42
(1 row)
```

`Add2Ints` function add two numbers together. Adding always returns the same output for a specific input, so its results are immutable. The following example shows recreating the function with the volatility set to `IMMUTABLE`.

```
=> CREATE FUNCTION Add2Ints AS LANGUAGE 'C++' NAME 'Add2IntsInfo' LIBRARY
ScalarFunctions IMMUTABLE;
ROLLBACK: Function with same name and number of parameters already exists:
Add2ints
=> -- Oops. Need to replace the old function.
=> CREATE OR REPLACE FUNCTION Add2ints AS LANGUAGE 'C++' NAME 'Add2IntsInfo'
LIBRARY ScalarFunctions IMMUTABLE;
CREATE FUNCTION
```

**See Also****CREATE LIBRARY** (page 634)**DROP FUNCTION** (page 700)**GRANT (Function)** (page 721)**REVOKE (Function)** (page 735)**V\_CATALOG.USER\_FUNCTIONS** (page 822)

Using User Defined Functions in the Programmer's Guide

## CREATE LIBRARY

Loads a shared library containing user defined functions (UDFs). You supply the absolute path to a Linux shared library (.so) file that contains the functions you want to access. See Using User Defined Functions in the Programmer's Guide for details.

**Warning:** User defined libraries are directly loaded by Vertica and run within the database process. Poorly-coded UDFs can cause instability or even database crashes. You should thoroughly test any UDF in a testing environment before deploying it.

**Syntax**

```
CREATE LIBRARY [schema].library_name AS 'library_path'
```

**Parameters**

<code>[schema].library_name</code>	A name to assign to this library. This is the name you use in a CREATE FUNCTION statement to enable user defined functions stored in the library. Note that this name is arbitrary. It does not need to reflect the name of the library file, although it would be less confusing if did.
<code>'library_path'</code>	The absolute path and filename of the library to load located on the initiator node.

**Notes**

- Only the superuser can create a library.
- As part of the loading process, Vertica distributes the library file to other nodes in the database. Any nodes that are down or that are added to the cluster later automatically receive a copy of the library file when they join the cluster.

- The CREATE LIBRARY statement performs some basic checks on the library file to ensure it is compatible with Vertica. The statement fails if it detects that the library was not correctly compiled or it finds other basic incompatibilities. However, there are many issues in shared libraries that CREATE LIBRARY cannot detect. Loading the library is no guarantee that it functions correctly.
- Libraries are added to the database catalog, and therefore persist across database restarts.

## Examples

To load a library in the home directory of the dbadmin account with the name MyFunctions:

```
=> CREATE LIBRARY MyFunctions AS 'home/dbadmin/my_functions.so';
```

To load a library located in the directory where you started vsql:

```
=> \set libfile '\``pwd`'/MyOtherFunctions.so\`;
=> CREATE LIBRARY MyOtherFunctions AS :libfile;
```

## CREATE PROCEDURE

Adds an external procedure to Vertica. See Implementing External Procedures in the Programmer's Guide for more information about external procedures.

### Syntax

```
CREATE PROCEDURE [schema-name.]procedure-name (
... [ argname ] argtype [, ...] )
... AS 'exec-name'
... LANGUAGE language-name
... USER 'OS-user'
```

### Parameters

[ <i>schema-name</i> .]	[Optional] Specifies the name of a schema.
<i>procedure-name</i>	Specifies a name for the external procedure. If the procedure-name is schema-qualified, the procedure is created in the specified schema.
<i>argname</i>	Provides a cue to procedure callers.
<i>argtype</i>	Specifies the data type for argument(s) that are passed to the procedure. Argument types must match Vertica type names. See <b>SQL Data Types</b> (page 64).
AS	Specifies the executable program in the procedures directory.
LANGUAGE	Specifies the language used, This parameter must be set to EXTERNAL.
USER	Specifies the user executed as. The user is the owner of the file. The user cannot be root. <b>Note:</b> The external program must allow execute privileges for this user.

## Notes

- A procedure file must be owned by the database administrator (OS account) or by a user in the same group as the administrator. (The procedure file owner cannot be root.) The procedure file must also have the set UID attribute enabled, and allow read and execute permission for the group.
- Only the database superuser can create procedures.
- By default, only the database superuser can execute procedures. However, the superuser can grant the right to execute procedures to other users. See **GRANT (Procedure)** (page 721).

## Example

This example illustrates how to create a procedure named *helloplanet* for the *helloplanet.sh* external procedure file. This file accepts one varchar argument.

### Sample file:

```
#!/bin/bash
echo "hello planet argument: $1" >> /tmp/myprocedure.log
exit 0
```

Issue the following SQL to create the procedure:

```
CREATE PROCEDURE helloplanet(arg1 varchar) as 'helloplanet.sh' language
'external' USER 'release';
```

## See Also

**DROP PROCEDURE** (page 702)

Installing External Procedure Executables in the Programmer's Guide

## CREATE PROFILE

Creates a profile that controls password requirements for users. Only the database superuser can create or alter a profile.

### Syntax

```
CREATE PROFILE name LIMIT
... [PASSWORD_LIFE_TIME {life-limit | DEFAULT | UNLIMITED}]
... [PASSWORD_GRACE_TIME {grace-period | DEFAULT | UNLIMITED}]
... [FAILED_LOGIN_ATTEMPTS {login-limit | DEFAULT | UNLIMITED}]
... [PASSWORD_LOCK_TIME {lock-period | DEFAULT | UNLIMITED}]
... [PASSWORD_REUSE_MAX {reuse-limit | DEFAULT | UNLIMITED}]
... [PASSWORD_REUSE_TIME {reuse-period | DEFAULT | UNLIMITED}]
... [PASSWORD_MAX_LENGTH {max-length | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LENGTH {min-length | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LETTERS {min-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_UPPERCASE_LETTERS {min-cap-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LOWERCASE_LETTERS {min-lower-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_DIGITS {min-digits | DEFAULT | UNLIMITED}]
```

**Note:** For all parameters, the special DEFAULT value means that the parameter's value is inherited from the DEFAULT profile. Any changes to the parameter in the DEFAULT profile is reflected by all of the profiles that inherit that parameter. Any parameter not specified in the CREATE PROFILE command is set to DEFAULT.

ParametersParameter Name	Description	Meaning of UNLIMITED value
<i>name</i>	The name of the profile to create	N/A
PASSWORD_LIFE_TIME <i>life-limit</i>	Integer number of days a password remains valid. After the time elapses, the user must change the password (or will be warned that their password has expired if PASSWORD_GRACE_TIME is set to a value other than zero or UNLIMITED).	Passwords never expire.
PASSWORD_GRACE_TIME <i>grace-period</i>	Integer number of days the users are allowed to login (while being issued a warning message) after their passwords are older than the PASSWORD_LIFE_TIME. After this period expires, users are forced to change their passwords on login if they have not done so after their password expired.	No grace period (the same as zero)
FAILED_LOGIN_ATTEMPTS <i>login-limit</i>	The number of consecutive failed login attempts that result in a user's account being locked.	Accounts are never locked, no matter how many failed login attempts are made.
PASSWORD_LOCK_TIME <i>lock-period</i>	Integer value setting the number of days an account is locked after the user's account was locked by having too many failed login attempts. After the PASSWORD_LOCK_TIME has expired, the account is automatically unlocked.	Accounts locked because of too many failed login attempts are never automatically unlocked. They must be manually unlocked by the database superuser.
PASSWORD_REUSE_MAX <i>reuse-limit</i>	The number of password changes that need to occur before the current password can be reused.	Users are not required to change passwords a certain number of times before reusing an old password.
PASSWORD_REUSE_TIME <i>reuse-period</i>	The integer number of days that must pass after a password has been set before the before it can be reused.	Password reuse is not limited by time.
PASSWORD_MAX_LENGTH <i>max-length</i>	The maximum number of	Passwords are limited to 100

	characters allowed in a password. Value must be in the range of 8 to 100.	characters.
PASSWORD_MIN_LENGTH <i>min-length</i>	The minimum number of characters required in a password. Valid range is 0 to <i>max-length</i> .	Equal to <i>max-length</i> .
PASSWORD_MIN_LETTERS <i>min-of-letters</i>	Minimum number of letters (a-z and A-Z) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).
PASSWORD_MIN_UPPERCASE_LETTERS <i>min-cap-letters</i>	Minimum number of capital letters (A-Z) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).
PASSWORD_MIN_LOWERCASE_LETTERS <i>min-lower-letters</i>	Minimum number of lowercase letters (a-z) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).
PASSWORD_MIN_DIGITS <i>min-digits</i>	Minimum number of digits (0-9) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).
PASSWORD_MIN_SYMBOLS <i>min-symbols</i>	Minimum number of symbols (any printable non-letter and non-digit character, such as \$, #, @, and so on) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).

**Note:** Only the profile settings for how many failed login attempts trigger account locking and how long accounts are locked have an effect on external password authentication methods such as LDAP or Kerberos. All password complexity, reuse, and lifetime settings only have an effect on passwords managed by Vertica.

## CREATE PROJECTION

Creates metadata for a projection in the Vertica catalog.

### Syntax

```
CREATE PROJECTION [ IF NOT EXISTS ]
... [ schema-name. ] projection-name (
... projection-column [ ENCODING encoding-type (on page 644) ]
... [ ACCESSRANK integer ] [ , ... ]
... [ GROUPED( projection-column-reference [ , ... ] ) ] )
AS SELECT table-column [ , ... ] FROM table-reference [ , ... ]
... [ WHERE join-predicate (on page 58) [ AND join-predicate ] ...
... [ ORDER BY table-column [ , ... ] ]
... [ hash-segmentation-clause (on page 646) | range-segmentation-clause (on page
647)
... | UNSEGMENTED { NODE node | ALL NODES } ]
... [ KSAFE [ k-num ] ]
```

### Parameters

[ IF NOT EXISTS ]	If supplied, CREATE PROJECTION does not generate an error if a projection with the same name already exists. If not supplied and CREATE PROJECTION finds a projection named <i>projection-name</i> , it reports an error and performs a rollback.
[ <i>schema-name</i> . ]	[Optional] Specifies the name of a schema.
<i>projection-name</i>	Specifies the name of the projection to be created. If the <i>projection-name</i> is schema-qualified, the projection is created in the specified schema. Otherwise, the projection is created in the same schema as the anchor table.
<i>projection-column</i>	Specifies the name of a column in the projection. The data type is inferred from the corresponding column in the schema table (based on ordinal position).  If projection columns are not explicitly named, they are inferred from the column names for the table specified in the SELECT statement. The following example automatically uses store and transaction as the projection column names for sales_p:  => CREATE TABLE sales(store INTEGER, transaction INTEGER); => CREATE PROJECTION sales_p AS SELECT * FROM sales KSAFE 1;  Note that you cannot specify specific encodings on projection columns using this method.  Different <i>projection-column</i> names can be used to distinguish multiple columns of the same name from different tables so that no aliases are needed.
ENCODING <i>encoding-type</i>	[Optional] Specifies the <b>type of encoding</b> (see " <b>encoding-type</b> " on page 644) to use on the column. By default, the <i>encoding-type</i> is auto.  <b>Caution:</b> Using the NONE keyword for strings could negatively affect the behavior of string columns.
ACCESSRANK <i>integer</i>	[Optional] Overrides the default access rank for a column. This

	is useful if you want to increase or decrease the speed at which a column is accessed. See <a href="#">Creating and Configuring Storage Locations and Prioritizing Column Access Speed</a> in the <a href="#">Administrator's Guide</a> .
GROUPED	<p>Groups two or more columns into a single disk file. This minimizes file I/O for work loads that:</p> <ul style="list-style-type: none"> <li>▪ Read a large percentage of the columns in a table.</li> <li>▪ Perform single row look-ups.</li> <li>▪ Query against many small columns.</li> <li>▪ Frequently update data in these columns.</li> </ul> <p>If you have data that is always accessed together and it is not used in predicates, you can increase query performance by grouping these columns. Once grouped, queries can no longer independently retrieve from disk all records for an individual column independent of the other columns within the group.</p> <p><b>Note:</b> RLE compression is reduced when a RLE column is grouped with one or more non-RLE columns.</p> <p>When grouping columns you can:</p> <ul style="list-style-type: none"> <li>▪ Group some of the columns: (a, GROUPED(b, c), d)</li> <li>▪ Group all of the columns: (GROUPED(a, b, c, d))</li> <li>▪ Create multiple groupings in the same projection: (GROUPED(a, b), GROUPED(c, d))</li> </ul> <p><b>Note:</b> Vertica performs dynamic column grouping. For example, to provide better read and write efficiency for small loads, Vertica ignores any projection-defined column grouping (or lack thereof) and groups all columns together by default.</p>
SELECT <i>table-column</i>	Specifies a list of schema table columns corresponding (in ordinal position) to the projection columns.
<i>table-reference</i>	Specifies a list of schema tables containing the columns to include in the projection in the form: <i>table-name</i> [ AS ] <i>alias</i> [ ( <i>column-alias</i> [ , ... ] ) ] [ , ... ] ]
WHERE <i>join-predicate</i>	Specifies foreign-key = primary-key equijoins between the fact table and dimension tables. No other predicates are allowed.
ORDER BY <i>table-column</i>	[Optional] Specifies which columns to sort. Because all projection columns are sorted in ascending order in physical storage, CREATE PROJECTION does not allow you to specify ascending or descending. <b>Note:</b> If you do not specify the sort order, Vertica uses the order in which columns are specified in the column list as the sort order for the projection.
<i>hash-segmentation-clause</i>	[Optional] Allows you to segment a projection based on a built-in hash function that provides even distribution of data across nodes, resulting in optimal query execution. See <b><i>hash-segmentation-clause</i></b> (on page 646).
<i>range-segmentation-clause</i>	[Optional] Allows you to segment a projection based on a known range of values stored in a specific column chosen to provide even

	distribution of data across a set of nodes, resulting in optimal query execution. See <b>range-segmentation-clause</b> (on page 647).
UNSEGMENTED { NODE <i>node</i>   ALL NODES }	[Optional] Allows you to specify that the projection be unsegmented, as follows: <ul style="list-style-type: none"> <li>▪ NODE <i>node</i>—Creates the unsegmented projection on the specified node only. Dimension table projections must be UNSEGMENTED.</li> <li>▪ ALL NODES—Creates a separate unsegmented projection on each node (automatic replication). To perform distributed query execution, Vertica requires an exact, unsegmented copy of each dimension table superprojection on each node.</li> </ul>
KSAFE [ <i>k-num</i> ]	Specifies the K-safety level of the projection. This integer determines how many replicated or segmented buddy projections are created. The value must be greater than or equal to the current K-safety level of the database and less than the total number of nodes. If KSAFE or its value are not specified, the projection might not be K-safe. This example creates a superprojection for a database with a K-safety of one (1): KSAFE 1 <b>Note:</b> KSAFE cannot be used with range segmentation.

### Unsegmented Projection Naming

CREATE PROJECTION ... UNSEGMENTED takes a snapshot of the nodes defined at execution time to generate a node list in a predictable order. Thus, replicated projections have the name:

*projection-name\_node-name*

For example, if the nodes are named NODE01, NODE02, and NODE03 then the following command creates projections named ABC\_NODE01, ABC\_NODE02, and ABC\_NODE03:

```
=> CREATE PROJECTION ABC ... UNSEGMENTED ALL NODES;
```

This naming convention could affect functions that provide information about projections, for example, **GET\_PROJECTIONS** (page 429) or **GET\_PROJECTION\_STATUS** (page 428), where you must provide the name ABC\_NODE01 instead of just ABC. To view a list of the nodes in a database, use the View Database command in the Administration Tools.

### Notes

- Constraints on columns are checked at projection creation time. If a join predicate includes a column with a FOREIGN\_KEY constraint and without a NOT\_NULL constraint, the CREATE PROJECTION statement fails.
- If there is a naming conflict with existing catalog objects (projections), the CREATE PROJECTION statement fails.
- Vertica recommends that you use multiple projection syntax for K-safe clusters.
- CREATE PROJECTION does not load data into physical storage. If the tables over which the projection is defined already contain data you must issue **START\_REFRESH** (page 470) to bring the projection up-to-date. This process could take a long time, depending on how much data is in the tables. Once a projection is up-to-date, it is updated as part of INSERT, UPDATE, DELETE or COPY statements.

- If no segmentation is specified, the default is UNSEGMENTED on the node where the CREATE PROJECTION was run.
- A projection is not refreshed until after a buddy projection is created. After the CREATE PROJECTION is run, if you run SELECT START\_REFRESH() the following message displays:

```
Starting refresh background process
```

However, the refresh does not begin until after a buddy projection is created. You can monitor the refresh operation by examining the `vertica.log` file or view the final status of the projection refresh by using `SELECT get_projections('table-name;')`. For example:

```
=> SELECT get_projections('customer_dimension');
           get_projections
```

```
-----
Current system K is 1.
```

```
# of Nodes: 4.
```

```
Table public.customer_dimension has 4 projections.
```

```
Projection Name: [Segmented] [Seg Cols] [# of Buddies] [Buddy
  Projections] [Safe] [UptoDate]
```

```
-----
public.customer_dimension_node0004 [Segmented: No] [Seg Cols: ] [K: 3]
  [public.customer_dimension_node0003,
  public.customer_dimension_node0002,
  public.customer_dimension_node0001] [Safe: Yes] [UptoDate:
  Yes][Stats: Yes]
```

```
public.customer_dimension_node0003 [Segmented: No] [Seg Cols: ] [K: 3]
  [public.customer_dimension_node0004,
  public.customer_dimension_node0002,
  public.customer_dimension_node0001] [Safe: Yes] [UptoDate:
  Yes][Stats: Yes]
```

```
public.customer_dimension_node0002 [Segmented: No] [Seg Cols: ] [K: 3]
  [public.customer_dimension_node0004,
  public.customer_dimension_node0003,
  public.customer_dimension_node0001] [Safe: Yes] [UptoDate:
  Yes][Stats: Yes]
```

```
public.customer_dimension_node0001 [Segmented: No] [Seg Cols: ] [K: 3]
  [public.customer_dimension_node0004,
  public.customer_dimension_node0003,
  public.customer_dimension_node0002] [Safe: Yes] [UptoDate:
  Yes][Stats: Yes]
```

```
(1 row)
```

Vertica internal operations (mergeout, refresh, and recovery) maintain partition separation except in certain cases:

- Recovery of a projection when the buddy projection from which the partition is recovering is identically sorted. If the projection is undergoing a full rebuild, it is recovered one ROS container at a time. The projection ends up with a storage layout identical to its buddy and is, therefore, properly segmented.

**Note:** In the case of a partial rebuild, all recovered data goes into a single ROS container and must be partitioned manually.

- Manual tuple mover operations often output a single storage container, combining any existing partitions; for example, after executing any of the `PURGE ()` operations.

### Example

The following example groups the highly correlated columns `bid` and `ask`. However, the `stock` column is stored separately.

```
=> CREATE TABLE trades (stock CHAR(5), bid INT, ask INT);
=> CREATE PROJECTION tradeproj (stock ENCODING RLE, GROUPED(bid ENCODING DELTAVAL,
ask))
    AS (SELECT * FROM trades) KSAFE 1;
```

## encoding-type

Vertica supports the following encoding and compression types:

### ENCODING AUTO (default)

For `CHAR/VARCHAR`, `BOOLEAN`, `BINARY/VARBINARY`, and `FLOAT` columns, Lempel-Ziv-Oberhumer-based (LZO) compression is used. For `INTEGER`, `DATE/TIME/TIMESTAMP`, and `INTERVAL` types, the compression scheme is based on the delta between consecutive column values.

Encoding Auto is ideal for sorted, many-valued columns such as primary keys. It is also suitable for general purpose applications for which no other encoding or compression scheme is applicable. Therefore, it serves as the default if no encoding/compression is specified.

The CPU requirements for this type are relatively small. In the worst case, data might expand by eight percent (8%) for LZO and twenty percent (20%) for integer data.

### ENCODING DELTAVAL

For `INTEGER` and `DATE/TIME/TIMESTAMP/INTERVAL` columns, data is recorded as a difference from the smallest value in the data block. This encoding has no effect on other data types.

Encoding Deltaval is best used for many-valued, unsorted integer or integer-based columns. The CPU requirements for this type are small, and the data never expands.

### ENCODING RLE

Run Length Encoding (RLE) replaces sequences (runs) of identical values with a single pair that contains the value and number of occurrences. Therefore, it is best used for low cardinality columns that are present in the `ORDER BY` clause of a projection.

The Vertica execution engine processes RLE encoding run-by-run and the Vertica optimizer gives it preference. Use it only when the run length is large, such as when low-cardinality columns are sorted.

The storage for RLE and AUTO encoding of `CHAR/VARCHAR` and `BINARY/VARBINARY` is always the same.

## **ENCODING BLOCK\_DICT**

For each block of storage, Vertica compiles distinct column values into a dictionary and then stores the dictionary and a list of indexes to represent the data block.

BLOCK\_DICT is ideal for few-valued, unsorted columns in which saving space is more important than encoding speed. Certain kinds of data, such as stock prices, are typically few-valued within a localized area once the data is sorted, such as by stock symbol and timestamp, and are good candidates for BLOCK\_DICT. Long CHAR/VARCHAR columns are not good candidates for BLOCK\_DICT encoding.

CHAR and VARCHAR columns that contain 0x00 or 0xFF characters should not be encoded with BLOCK\_DICT. Also, BINARY/VARBINARY columns do not support BLOCK\_DICT encoding.

The encoding CPU for BLOCK\_DICT is significantly higher than for default encoding schemes. The maximum data expansion is eight percent (8%).

## **ENCODING BLOCKDICT\_COMP**

This encoding type is similar to BLOCK\_DICT except that dictionary indexes are entropy coded. This encoding type requires significantly more CPU time to encode and decode and has a poorer worst-case performance. However, use of this type can lead to space savings if the distribution of values is extremely skewed.

## **ENCODING DELTARANGE\_COMP**

This compression scheme is primarily used for floating point data, and it stores each value as a delta from the previous one.

This scheme is ideal for many-valued FLOAT columns that are either sorted or confined to a range. Do not use this scheme for unsorted columns that contain NULL values, as the storage cost for representing a NULL value is high. This scheme has a high cost for both compression and decompression.

To determine if DELTARANGE\_COMP is suitable for a particular set of data, compare it to other schemes. Be sure to use the same sort order as the projection, and select sample data that will be stored consecutively in the database.

## **ENCODING COMMONDELTA\_COMP**

This compression scheme builds a dictionary of all the deltas in the block and then stores indexes into the delta dictionary using entropy coding.

This scheme is ideal for sorted FLOAT and INTEGER-based (DATE/TIME/TIMESTAMP/INTERVAL) data columns with predictable sequences and only the occasional sequence breaks, such as timestamps recorded at periodic intervals or primary keys. For example, the following sequence compresses well: 300, 600, 900, 1200, 1500, 600, 1200, 1800, 2400. The following sequence does not compress well: 1, 3, 6, 10, 15, 21, 28, 36, 45, 55.

If the delta distribution is excellent, columns can be sorted in less than one bit per row. However, this scheme is very CPU intensive. If you use this scheme on data with arbitrary deltas, it can lead to significant data expansion.

## ENCODING NONE

Do not specify this value. It is obsolete and exists only for backwards compatibility. The result of ENCODING NONE is the same as ENCODING AUTO except when applied to CHAR and VARCHAR columns. Using ENCODING NONE on these columns increases space usage, increases processing time, and leads to problems if 0x00 or 0xFF characters are present in the data.

## hash-segmentation-clause

Allows you to segment a projection based on a built-in hash function that provides even distribution of data across some or all of the nodes in a cluster, resulting in optimal query execution.

**Note:** Hash segmentation is the preferred method of segmentation. The Database Designer uses hash segmentation by default.

### Syntax

```
SEGMENTED BY expression
  [ ALL NODES [ OFFSET offset ] | NODES node [ ,... ] ]
```

### Parameters

SEGMENTED BY <i>expression</i>	Can be a general SQL expression, but there is no reason to use anything other than the built-in <b>HASH</b> (page 287) or <b>MODULARHASH</b> (page 290) functions with table columns as arguments.  Choose columns that have a large number of unique data values and acceptable skew in their data distribution. Primary key columns that meet the criteria could be an excellent choice for hash segmentation.
ALL NODES	Automatically distributes the data evenly across all nodes at the time the CREATE PROJECTION statement is run. The ordering of the nodes is fixed.
OFFSET <i>offset</i>	Is an integer that specifies the node within the ordered sequence on which to start the segmentation distribution, relative to 0. See example below.
NODES <i>node</i> [ ,... ]	Specifies a subset of the nodes in the cluster over which to distribute the data. You can use a specific node only once in any projection. For a list of the nodes in a database, use the View Database command in the Administration Tools.

### Notes

- Omitting an OFFSET clause is equivalent to OFFSET 0.
- CREATE PROJECTION accepts the deprecated syntax SITES *node* for compatibility with previous releases.
- Table column names must be used in the expression, not the new projection column names
- If you want to use a different SEGMENTED BY expression, the following restrictions apply:
  - All leaf expressions must be either constants or **column-references** (see "**Column References**" on page 46) to a column in the SELECT list of the CREATE PROJECTION command.

- Aggregate functions are not allowed.
- The expression must return the same value over the life of the database.
- The expression must return non-negative INTEGER values in the range  $0 \leq x < 2^{63}$  (two to the sixty-third power or  $2^{63}$ ), and values are generally distributed uniformly over that range.
- If *expression* produces a value outside the expected range (a negative value for example), no error occurs, and the row is added to the first segment of the projection.

## Examples

```
=> CREATE PROJECTION ... SEGMENTED BY HASH(C1,C2) ALL NODES;
=> CREATE PROJECTION ... SEGMENTED BY HASH(C1,C2) ALL NODES OFFSET 1;
```

The example produces two hash-segmented buddy projections that form part of a K-Safe design. The projections can use different sort orders.

```
=> CREATE PROJECTION fact_ts_2 (
    f_price,
    f_cid,
    f_tid,
    f_cost,
    f_date) AS (
    SELECT price, cid, tid, cost, dwwdate FROM fact)
SEGMENTED BY ModularHash(dwwdate) ALL NODES OFFSET 2;
```

## See Also

**HASH** (page 287) and **MODULARHASH** (page 290)

## range-segmentation-clause

Allows you to segment a projection based on a known range of values stored in a specific column chosen to provide even distribution of data across a set of nodes, resulting in optimal query execution.

**Note:** Vertica recommends that you use hash segmentation, instead of range segmentation.

## Syntax

```
SEGMENTED BY expression
    NODE node VALUES LESS THAN value
    ...
    NODE node VALUES LESS THAN MAXVALUE
```

## Parameters (Range Segmentation)

SEGMENTED BY <i>expression</i>	<p>Is a single <b>column reference</b> (see "<b>Column References</b>" on page 46) to a column in the SELECT list of the CREATE PROJECTION statement. Choose a column that has:</p> <ul style="list-style-type: none"> <li>▪ INTEGER or FLOAT data type</li> <li>▪ A known range of data values</li> <li>▪ An even distribution of data values</li> <li>▪ A large number of unique data values</li> </ul> <p>Avoid columns that:</p>
--------------------------------	--

	<ul style="list-style-type: none"> <li>▪ Are foreign keys</li> <li>▪ Are used in query predicates</li> <li>▪ Have a date/time data type</li> <li>▪ Have correlations with other columns due to functional dependencies.</li> </ul> <p><b>Note:</b> Segmenting on DATE/TIME data types is valid but guaranteed to produce temporal skew in the data distribution and is not recommended. If you choose this option, do not use TIME or TIMETZ because their range is only 24 hours.</p>
NODE <i>node</i>	Is a symbolic name for a node. You can use a specific node only once in any projection. For a list of the nodes in a database, use <code>SELECT * FROM NODE_RESOURCES</code> .
VALUES LESS THAN <i>value</i>	Specifies that this segment can contain a range of data values <i>less than</i> the specified <i>value</i> , except that segments cannot overlap. In other words, the minimum value of the range is determined by the <i>value</i> of the previous segment (if any).
MAXVALUE	Specifies a sub-range with no upper limit. In other words, it represents a value greater than the maximum value that can exist in the data. The maximum value depends on the data type of the segmentation column.

## Notes

- The `SEGMENTED BY expression` syntax allows a general SQL expression but there is no reason to use anything other than a single **column reference** (see "**Column References**" on page 46) for range segmentation. If you want to use a different expression, the following restrictions apply:
  - All leaf expressions must be either constants or **column-references** (see "**Column References**" on page 46) to a column in the SELECT list of the CREATE PROJECTION command
  - Aggregate functions are not allowed
  - The expression must return the same value over the life of the database.
- During INSERT or COPY to a segmented projection, if *expression* produces a value outside the expected range (a negative value for example), no error occurs, and the row is added to a segment of the projection.
- CREATE PROJECTION with range segmentation accepts the deprecated syntax `SITE node` for compatibility with previous releases.
- CREATE PROJECTION with range segmentation allows the `SEGMENTED BY` expression to be a single column-reference to a column in the *projection-column* list for compatibility with previous releases. This syntax is considered to be a deprecated feature and causes a warning message. See DEPRECATED syntax in the Troubleshooting Guide.

## See Also

**NODE\_RESOURCES** (page 861)

## CREATE RESOURCE POOL

Creates a resource pool.

### Syntax

```
CREATE RESOURCE POOL pool-name MEMORYSIZE 'sizeUnits'
... [ MAXMEMORYSIZE 'sizeUnits' | NONE ]
... [ PRIORITY n ]
... [ QUEUE_TIMEOUT n | NONE ]
... [ PLANNED_CONCURRENCY n ]
... [ SINGLE_INITIATOR bool ]
... [ MAX_CONCURRENCY n | NONE ]
```

### Parameters

<i>pool-name</i>	Specifies the name of the resource pool to create.
MEMORYSIZE ' <i>sizeUnits</i> '	<p>[Default 0%] Amount of memory allocated to the resource pool. See also MAXMEMORYSIZE parameter.</p> <p>Units can be one of the following:</p> <ul style="list-style-type: none"> <li>▪ % percentage of total memory available to the Resource Manager. (In this case, size must be 0-100).</li> <li>▪ K Kilobytes</li> <li>▪ M Megabytes</li> <li>▪ G Gigabytes</li> <li>▪ T Terabytes</li> </ul> <p><b>Note:</b> This parameter refers to memory allocated to this pool per node and not across the whole cluster.</p> <p>The default of 0% means that the pool has no memory allocated to it and must exclusively borrow from the GENERAL pool.</p>
MAXMEMORYSIZE ' <i>sizeUnits</i> '   NONE	<p>[Default unlimited] Maximum size the resource pool could grow by borrowing memory from the GENERAL pool. See <b>Built-in Pools</b> (page 652) for a discussion on how resource pools interact with the GENERAL pool.</p> <p>Units can be one of the following:</p> <ul style="list-style-type: none"> <li>▪ % percentage of total memory available to the Resource Manager. (In this case, size must be 0-100). This notation has special meaning for the GENERAL pool, described in Notes below.</li> <li>▪ K Kilobytes</li> <li>▪ M Megabytes</li> <li>▪ G Gigabytes</li> <li>▪ T Terabytes</li> </ul> <p>If MAXMEMORYSIZE NONE is specified, there is no upper limit.</p> <p><b>Notes:</b></p> <p>The MAXMEMORYSIZE parameter refers to the maximum memory borrowed by this pool per node and not across the</p>

	<p>whole cluster.</p> <p>The default of unlimited means that the pool can borrow as much memory from <code>GENERAL</code> pool as is available.</p> <p>When set as a percentage (%) value, <code>GENERAL.MAXMEMORYSIZE</code> governs the total amount of RAM that the Resource Manager can use for queries, regardless of whether the parameter is set to a percent or to a specific value (for example, '10G'). The default setting is 95%.</p>
PRIORITY	<p>[Default 0] An integer that represents priority of queries in this pool, when they compete for resources in the <code>GENERAL</code> pool. Higher numbers denote higher priority.</p>
QUEUETIMEOUT	<p>[Default 300 seconds] An integer, in seconds, that represents the maximum amount of time the request is allowed to wait for resources to become available before being rejected. If set to <code>NONE</code>, the request can be queued for an unlimited amount of time.</p>
PLANNEDCONCURRENCY	<p>[Default: Max ( 4, Min ( total available memory / 2GB, #cores ) ) ]</p> <p>An integer that represents number of concurrent queries that are normally expected to be running against the resource pool. This is not a hard limit and is used when apportioning memory in the pool to various requests.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>▪ This is a cluster-wide maximum and not a per-node limit.</li> <li>▪ If you created or upgraded your database in 4.0 or 4.1, the <code>PLANNEDCONCURRENCY</code> setting on the <code>GENERAL</code> pool defaults to a too-small value for machines with large numbers of cores. To adjust to a more appropriate value: <ul style="list-style-type: none"> <li>▪ =&gt; <code>ALTER RESOURCE POOL general PLANNEDCONCURRENCY</code></li> <li>▪ <code>&lt;#cores&gt;;</code></li> <li>▪ You only need to set this parameter if you created a database before 4.1, patchset 1.</li> </ul> </li> </ul>
SINGLEINITIATOR	<p>[Default false] A boolean that indicates whether all requests using this pool are issued against the same initiator node or whether multiple initiator nodes can be used; for instance in a round-robin configuration.</p> <p><b>Note:</b> Vertica recommends that you distribute requests evenly across all nodes and leave this parameter unchanged.</p>
MAXCONCURRENCY	<p>[Default unlimited] An integer that represents the maximum number of concurrent execution slots available to the resource pool. If <code>MAXCONCURRENCY NONE</code> is specified, there is no limit.</p> <p><b>Note:</b> This is a cluster-wide maximum and not a per-node limit.</p>

## Notes

- Resource pool names are subject to the same rules as Vertica *identifiers* (page 15). **Built-in pool** (page 652) names cannot be used for user-defined pools.
- New resource pools can be created or altered without shutting down the system.
- When a new pool is created (or its size altered), `MEMORYSIZE` amount of memory is taken out of the `GENERAL pool` (page 652). If the `GENERAL pool` does not currently have sufficient memory to create the pool due to existing queries being processed, a request is made to the system to create a pool as soon as resources become available. The pool is created immediately and memory is moved to the pool as it becomes available. Such memory movement has higher priority than any query.

The pool is in operation as soon as the specified amount of memory becomes available. You can monitor whether the `ALTER` has been completed in the `V_MONITOR.RESOURCE_POOL_STATUS` (page 814) system table.

- Under normal operation, `MEMORYSIZE` is required to be less than `MAXMEMORYSIZE` and an error is returned during `CREATE/ALTER` operations if this size limit is violated. However, under some circumstances where the node specification changes by addition/removal of memory, or if the database is moved to a different cluster, this invariant could be violated. In this case, `MAXMEMORYSIZE` is increased to `MEMORYSIZE`.
- If two pools have the same `PRIORITY`, their requests are allowed to borrow from the `GENERAL pool` in order of arrival.

See Guidelines for Setting Pool Parameters in the Administrator's Guide for details about setting these parameters.

## Example

The following command creates a resource pool with `MEMORYSIZE` of 110MB to ensure that the CEO query has adequate memory reserved for it:

```
=> CREATE RESOURCE POOL ceo_pool MEMORYSIZE '110M' PRIORITY 10;
\pset expanded
Expanded display is on.
SELECT * FROM resource_pools WHERE name = 'ceo_pool';
-[ RECORD 1 ]-----+-----
name                | ceo_pool
is_internal          | f
memorysize           | 110M
maxmemorysize       |
priority             | 10
queuetimeout         | 300
plannedconcurrency  | 4
maxconcurrency       |
singleinitiator      | f
```

Assuming the CEO report user already exists, associate this user with the above resource pool using `ALTER USER` statement.

```
=> ALTER USER ceo_user RESOURCE POOL ceo_pool;
```

Issue the following command to confirm that the `ceo_user` is associated with the `ceo_pool`:

```
=> SELECT * FROM users WHERE user_name = 'ceo_user';
```

```

-[ RECORD 1 ]--+-----
user_id      | 45035996273713548
user_name    | ceo_user
is_super_user | f
resource_pool | ceo_pool
memory_cap_kb | unlimited
    
```

**See Also**

**ALTER RESOURCE POOL** (page 582)

**CREATE USER** (page 695)

**DROP RESOURCE POOL** (page 705)

**SET SESSION RESOURCE POOL** (page 778)

**SET SESSION MEMORYCAP** (page 777)

Managing Workloads in the Administrator's Guide

**Built-in Pools**

Vertica is preconfigured to have several built-in pools for various system tasks. The built-in pools can be reconfigured to suit your usage. The following sections describe the purpose of built-in pools and the default settings.

**Built-in Pool      Settings**

GENERAL	<p>A special, catch-all pool used to answer requests that have no specific resource pool associated with them. Any memory left over after memory has been allocated to all other pools is automatically allocated to the <code>GENERAL</code> pool. The <code>MEMORYSIZE</code> parameter of the <code>GENERAL</code> pool is undefined (variable), however, the <code>GENERAL</code> pool must be at least 1GB in size and cannot be smaller than 25% of the memory in the system.</p> <p>The <code>MAXMEMORYSIZE</code> parameter of the <code>GENERAL</code> pool has special meaning; when set as a % value it represents the percent of total physical RAM on the machine that the Resource Manager can use for queries. By default it is set to 95%. The <code>GENERAL.MAXMEMORYSIZE</code> governs the total amount of RAM that the Resource Manager can use for queries, regardless of whether it is set to a percent or to a specific value (for example, '10GB')</p> <p>Any user-defined pool can “borrow” memory from the <code>GENERAL</code> pool to satisfy requests that need extra memory until the <code>MAXMEMORYSIZE</code> parameter of that pool is reached. If the pool is configured to have <code>MEMORYSIZE</code> equal to <code>MAXMEMORYSIZE</code> it cannot borrow any memory from the <code>GENERAL</code> pool and is said to be a standalone resource pool. When multiple pools request memory from the <code>GENERAL</code> pool, they are granted access to general pool memory according to their priority setting. In this manner, the <code>GENERAL</code> pool provides some elasticity to account for point-in-time deviations from normal usage of individual resource pools.</p>
SYSQUERY	Is used to answer queries against <i>system monitoring and catalog tables</i> (page 794).
SYSDATA	Is used to reserve memory for results of queries against <i>system monitoring and</i>

	<i>catalog tables</i> (page 794).
WOSDATA	Is used by the Write Optimized Store (WOS). Loads to the WOS automatically spill to the ROS once they exceed a certain amount of WOS usage; the <code>PLANNEDCONCURRENCY</code> parameter of the WOS is used to determine this spill threshold. For instance, if <code>PLANNEDCONCURRENCY</code> of the <code>WOSDATA</code> pool is set to 4, then a load spills to the ROS once it has occupied one quarter of the WOS. See Scenario: Tuning for Continuous Load and Query in the Administrator's Guide.
TM	Is memory used by the Tuple Mover(TM). The <code>MAXCONCURRENCY</code> parameter can be used to allow more than one concurrent TM operation to occur at the same time. See Scenario: Tuning Tuple Mover Pool Settings in the Administrator's Guide.
RECOVERY	Is used by queries issued when recovering another node of the database. The <code>MAXCONCURRENCY</code> parameter is used to determine how many concurrent recovery threads to use. The <code>PLANNEDCONCURRENCY</code> parameter (by default set to twice the <code>MAXCONCURRENCY</code> ) can be used to tune how to apportion memory to the recovery queries. See Scenario: Tuning for Recovery in the Administrator's Guide.
REFRESH	Is used by queries issued by the <b>PROJECTION_REFRESHES</b> (page 863) operations. Refresh currently does not use multiple concurrent threads; thus, changes to the <code>MAXCONCURRENCY</code> values have no effect. See Scenario: Tuning for Refresh in the Administrator's Guide.
DBD	Is used by the Database Designer to control resource usage by its internal processing. This pool is configured to have a <code>QUEUETIMEOUT</code> of 0 seconds. This is done because the Database Designer is a resource intensive process, and in a system under severe memory pressure, the Database Designer excuses itself from running (instead of being queued and adding to the existing contention) and notifies the user to run it when the system is quieter. Vertica recommends that you do not reconfigure this pool.

## Upgrade from Earlier Versions of Vertica

For a database being upgraded from earlier versions of, Vertica automatically translates most existing parameter values into the new resource pool settings.

The `PLANNEDCONCURRENCY` and `MAXCONCURRENCY` parameters of the resource pools must be manually tuned per Guidelines for Setting Pool Parameters in the Administrator's Guide.

## Built-in Pool Configuration

The following tables show the default values of the configuration settings for the built-in pools for a new database and for a database upgraded from prior versions of Vertica.

**Note:** Some of the parameter values of built-in pools have special restrictions, which are noted in the tables.

**GENERAL**

Setting	Value
MEMORYSIZE	N/A (cannot be set)
MAXMEMORYSIZE	<p>Default: 95% of Total RAM on the node.</p> <p>3.5 upgrade: Set as a % value based on <code>MaxResourceUsagePct</code> parameter.</p> <p>Restrictions:</p> <ul style="list-style-type: none"> <li>▪ Cannot be less than 1GB.</li> <li>▪ Cannot be less than 25% of RAM.</li> <li>▪ Setting to 100% generates warnings that swapping could result.</li> </ul>
PRIORITY	0
QUEUE_TIMEOUT	<p>300</p> <p>3.5 Upgrade: Resource Timeout parameter</p>
PLANNED_CONCURRENCY	<p><math>\text{Min}(2 * \# \text{cores}, \text{TotalLoads})</math> if <code>TotalLoads</code> is set, otherwise <math>\text{Min}(2 * \# \text{cores}, \# \text{nodes} * \text{LoadsPerNode})</math></p> <p>3.5 Upgrade: Max (default above, <code>LPN+2</code>) where <code>LPN</code> is the <code>LoadsPerNode</code> parameter if set or <code>TotalLoads/# nodes</code> if set.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>▪ <code>QueriesPerNode</code> or <code>TotalQueries</code> are not used in the default calculation. See <i>Best Practices For Workload Management</i> in the <i>Administrator's Guide</i> for guidance on how to tune.</li> <li>▪ The <code>PLANNED_CONCURRENCY</code> setting on the <code>GENERAL</code> pool defaults to a too-small value for machines with large numbers of cores. To adjust to a more appropriate value:</li> <li>▪ <code>=&gt; ALTER RESOURCE POOL general PLANNED_CONCURRENCY &lt;#cores&gt;;</code></li> </ul> <p>See <i>Guidelines for Setting Pool Parameters</i> in the <i>Administrator's Guide</i></p>
MAX_CONCURRENCY	<p>Unlimited</p> <p>Restrictions: Setting to 0 generates warnings that no system queries may be able to run in the system.</p>
SINGLE_INITIATOR	False

**SYSQUERY**

Setting	Value
MEMORYSIZE	<p>64M</p> <p>3.5 Upgrade: <code>SysWOSSizeMB</code></p> <p>Restrictions: Setting to &lt;20M generates warnings because it could prevent system queries from running and make problem diagnosis difficult.</p>
MAXMEMORYSIZE	Unlimited

PRIORITY	110
QUEUETIMEOUT	300 3.5 Upgrade: Resource Timeout parameter
PLANNEDCONCURRENCY	See GENERAL
MAXCONCURRENCY	Unlimited Restrictions: Setting to 0 generates warnings that no system queries may be able to run in the system.
SINGLEINITIATOR	False

## SYSDATA

Setting	Value
MEMORYSIZE	100m 3.5 Upgrade: <code>SystemSizeMB</code>
MAXMEMORYSIZE	10% Restriction: Setting To <4m generates warnings that no system queries may be able to run in the system.
PRIORITY	N/A (cannot be set)
QUEUETIMEOUT	N/A (cannot be set)
PLANNEDCONCURRENCY	N/A (cannot be set)
MAXCONCURRENCY	N/A (cannot be set)
SINGLEINITIATOR	N/A (cannot be set)

## WOSDATA

Setting	Value
MEMORYSIZE	0% 3.5 Upgrade: If <code>ReserveWOSMemory</code> is set, then <code>MaxWOSMemPct</code> , else 0%
MAXMEMORYSIZE	25% or 2GB, whichever is less. (25% for databases created before 4.1 Patchset 1.) 3.5 Upgrade: <code>MaxWOSMemPct</code>
PRIORITY	N/A (cannot be set)
QUEUETIMEOUT	N/A (cannot be set)
PLANNEDCONCURRENCY	2*#nodes 3.5 Upgrade: <code>TotalLoads</code> , if set or <code>LoadsPerNode*#nodes</code>
MAXCONCURRENCY	N/A (cannot be set)
SINGLEINITIATOR	N/A (cannot be set)

## TM

Setting	Value
MEMORYSIZE	100M
MAXMEMORYSIZE	Unlimited
PRIORITY	105
QUEUETIMEOUT	300 3.5 Upgrade: Resource Timeout parameter
PLANNEDCONCURRENCY	1
MAXCONCURRENCY	2 Restrictions: Cannot set to 0 or NONE(unlimited)
SINGLEINITIATOR	True

## REFRESH

Setting	Value
MEMORYSIZE	0%
MAXMEMORYSIZE	Unlimited
PRIORITY	-10
QUEUETIMEOUT	300 3.5 Upgrade: Resource Timeout parameter
PLANNEDCONCURRENCY	4
MAXCONCURRENCY	Unlimited Restrictions: cannot set to 0
SINGLEINITIATOR	True

## RECOVERY

Setting	Value
MEMORYSIZE	0%
MAXMEMORYSIZE	Unlimited Restrictions: Cannot set to < 25%.
PRIORITY	107
QUEUETIMEOUT	300 3.5 Upgrade: Resource Timeout parameter

PLANNEDCONCURRENCY	Twice MAXCONCURRENCY
MAXCONCURRENCY	(# of cores / 2) + 1 3.5 Upgrade: RecoverThreadCount Restrictions: Cannot set to 0 or NONE (unlimited)
SINGLEINITIATOR	True

**DBD**

Setting	Value
MEMORYSIZE	0%
MAXMEMORYSIZE	Unlimited
PRIORITY	0
QUEUETIMEOUT	0
PLANNEDCONCURRENCY	See GENERAL
MAXCONCURRENCY	Unlimited
SINGLEINITIATOR	False

**CREATE ROLE**

Creates a new role. The new role is empty. You need to add permissions to it using GRANT statements (**GRANT (Table)** 727, **GRANT (Schema)** 725, **GRANT (View)** 728, **GRANT (Database)** 720, **GRANT (Procedure)** 721, **GRANT (Sequence)** 725, **GRANT (Resource Pool)** 722, **GRANT (Function)** 721).

**Syntax**

```
CREATE ROLE role;
```

**Parameters**

<i>role</i>	The name for the new role.
-------------	----------------------------

**Note:** Only the database superuser can create a new role.

**CREATE SCHEMA**

Defines a new schema.

**Syntax**

```
CREATE SCHEMA schemaname [ AUTHORIZATION user-name ]
```

## Parameters

<i>schemaname</i>	Specifies the name of the schema to create. The schema name must be distinct from all other schemas within the database. If the schema name is not provided, the user name is used as the schema name.
AUTHORIZATION <i>user-name</i>	Assigns ownership of the schema to a user. If a user name is not provided, the user who creates the schema is assigned ownership. Only a Superuser is allowed to create a schema that is owned by a different user.

## Notes

- To create a schema, the user must either be a superuser or have CREATE privilege for the database. See **GRANT (Database)** (page 720).
- Optionally, CREATE SCHEMA could include the following sub-statements to create tables within the schema:
  - **CREATE TABLE** (page 664)
  - GRANT
- With the following exceptions, these sub-statements are treated as if they have been entered as individual commands after the CREATE SCHEMA statement has completed:
  - If the AUTHORIZATION statement is used, all tables are owned by the specified user.
  - The CREATE SCHEMA statement and all its associated sub-statements are completed as one transaction. If any of the statements fail, the entire CREATE SCHEMA statement is rolled back.

## Examples

The following example creates a schema named `s1` with no objects.

```
=> CREATE SCHEMA s1;
```

The following series of commands create a schema named `s1` with a table named `t1` and grants Fred and Aniket access to all existing tables and ALL privileges on table `t1`:

```
=> CREATE SCHEMA s1;  
=> CREATE TABLE t1 (c INT);  
=> GRANT USAGE ON SCHEMA s1 TO Fred, Aniket;  
=> GRANT ALL ON TABLE t1 TO Fred, Aniket;
```

## See Also

**ALTER SCHEMA** (page 585)

**SET SEARCH\_PATH** (page 774)

**DROP SCHEMA** (page 706)

## CREATE SEQUENCE

Defines a new sequence number generator.

## Syntax

```
CREATE SEQUENCE [schema-name.] sequence_name
... [ INCREMENT [ BY ] increment ]
... [ MINVALUE minvalue | NO MINVALUE ]
... [ MAXVALUE maxvalue | NO MAXVALUE ]
... [ START [ WITH ] start ]
... [ CACHE cache ]
... [ CYCLE | NO CYCLE ]
```

## Parameters

<i>sequence_name</i>	The name (optionally schema-qualified) of the sequence to be created. The name must be unique among sequences, tables, projections, and views.
<i>increment</i>	Specifies which value is added to the current sequence value to create a new value. A positive value makes an ascending sequence; a negative value makes a descending sequence. The default value is 1.
<i>minvalue</i>   NO MINVALUE	Determines the minimum value a sequence can generate. If this clause is not supplied or NO MINVALUE is specified, then defaults are used. The defaults are 1 and $-2^{63}-1$ for ascending and descending sequences, respectively.
<i>maxvalue</i>   NO MAXVALUE	Determines the maximum value for the sequence. If this clause is not supplied or NO MAXVALUE is specified, then default values are used. The defaults are $2^{63}-1$ and -1 for ascending and descending sequences, respectively.
<i>start</i>	Allows the sequence to begin anywhere. The default starting value is <i>minvalue</i> for ascending sequences and <i>maxvalue</i> for descending sequences.
<i>cache</i>	Specifies how many sequence numbers are pre-allocated and stored in memory for faster access. The default is 250,000 with a minimum value is 1 (only one value can be generated at a time, for example, no cache). <b>Notes:</b> <ul style="list-style-type: none"> <li>▪ When the CACHE clause is specified, each session has its own cache on each Vertica node.</li> <li>▪ Sequences that specify a small cache size could cause a performance degradation.</li> </ul>
CYCLE   NO CYCLE	Allows the sequence to wrap around when the <i>maxvalue</i> or <i>minvalue</i> is reached by an ascending or descending sequence, respectively. If the limit is reached, the next number generated is the <i>minvalue</i> or <i>maxvalue</i> , respectively. If NO CYCLE is specified, any calls to NEXTVAL (page 308) after the sequence has reached its maximum/minimum value return an error. NO CYCLE is the default.

## Notes

- Consider using sequences or auto-incrementing columns for primary key columns, which guarantees uniqueness and avoids the constraint enforcement problem and associated overhead. For more information see Using Sequences in the Administrator's Guide.

- If a schema name is given, the sequence is created in the specified schema. Otherwise it is created in the current schema. The sequence name must be distinct from the name of any other sequence, table, index, or view in the same schema.
- You must have CREATE privileges on the schema in which you want to create a sequence.
- After a sequence is created, use the functions **NEXTVAL** (page 308) and **CURRVAL** (page 310) to operate on the sequence. A cache is created when NEXTVAL is called.
- You cannot use NEXTVAL or CURRVAL to act on a sequence in a SELECT statement:
  - in a WHERE clause
  - in a GROUP BY or ORDER BY clause
  - in a DISTINCT clause
  - along with a UNION
  - in a subquery
- Additionally, you cannot use NEXTVAL or CURRVAL to act on a sequence in:
  - a subquery of UPDATE or DELETE
  - a view
- You can work around some of these restrictions by using subqueries. For example, to use sequences with a DISTINCT clause:  
=> 

```
SELECT t.col1, shift_allocation_seq.nextval
FROM (
    SELECT DISTINCT col1 FROM av_temp1) t;
```
- If you want to generate, for example, only even numbers, specify INCREMENT BY 2 and use a *start* value of 2.
- Use **DROP SEQUENCE** (page 708) to remove a sequence; however, you cannot drop a sequence upon which other objects depends.
- DROP SEQUENCE ... CASCADE is not supported. Sequences used in a default expression of a column cannot be dropped until all references to the sequence are removed from the default expression.

### Cache Operation

- In each session, every node maintains its own cache of the sequence state; thus, Vertica internally takes a Global Catalog Lock(X) to obtain a cache of values from a sequence.
- Regardless of the number of calls to NEXTVAL and CURRVAL, sequences are incremented one time per row. This means multiple calls to NEXTVAL within the same row return the same value. If joins are used, a sequence is incremented one time for the final composite row.
- It is possible for one session to allocate a cache and use it slowly while another statement requests and loads many values. Therefore, the values returned from NEXTVAL in one statement could be distant from the values returned in another statement.
- If a disconnect occurs, any remaining values that have not been returned through NEXTVAL are lost.
- If a statement fails after NEXTVAL is called and the cache is incremented, the new cache value is not rolled back.
- When the cache runs out of sequence numbers, it automatically obtains more from the Global Catalog.

- Because large cache sizes can create gaps in the sequence, you might decide to specify a smaller cache. Note that smaller cache sizes can result in a performance degradation.

## Examples

The following example creates an ascending sequence called `sequential`, starting at 101:

```
=> CREATE SEQUENCE my_seq START 101;
```

After a sequence is created, use the sequence functions **NEXTVAL** (page 308) and **CURRVAL** (page 310) to operate on the sequence. These functions provide simple, multiuser-safe methods for obtaining successive sequence values from sequence objects.

### Note:

CURRVAL returns a sequence's most recent value, so if you run CURRVAL before NEXTVAL, the system returns an error:

```
ERROR: Sequence my_seq has not been accessed in the session
```

NEXTVAL must be called at least one time in a session to provide a value for CURRVAL. A cache is created when NEXTVAL is called.

The following command generates the first number for this sequence:

```
=> SELECT NEXTVAL('my_seq');
nextval
-----
      101
(1 row)
```

The following command returns the current value of this sequence. Since no other operations have been performed on the newly-created sequence, the function returns the expected value of 101:

```
=> SELECT CURRVAL('my_seq');
currval
-----
      101
(1 row)
```

The following command increments the value for this sequence by one (1):

```
=> SELECT NEXTVAL('my_seq');
nextval
-----
      102
(1 row)
```

Calling the CURRVAL again function returns only the current value:

```
=> SELECT CURRVAL('my_seq');
currval
-----
      102
(1 row)
```

The following example shows how to use the `my_sequence` sequence in an `INSERT` statement.

```
=> CREATE TABLE customer (  
    lname VARCHAR(25),  
    fname VARCHAR(25),  
    membership_card INTEGER,  
    ID INTEGER  
);  
=> INSERT INTO customer VALUES ('Hawkins', 'John', 072753, NEXTVAL('my_seq'));
```

Now query the table you just created. Notice that the `ID` column has been incremented 1 value to 103:

```
=> SELECT * FROM customer;  
  lname | fname | membership_card | ID  
-----+-----+-----+-----  
 Hawkins | John  |           72753 | 103  
(1 row)
```

The following example shows how to use a sequence as the default value for an `INSERT` command:

```
=> CREATE TABLE customer2(  
    ID INTEGER DEFAULT NEXTVAL('my_seq'),  
    lname VARCHAR(25),  
    fname VARCHAR(25),  
    membership_card INTEGER  
);  
=> INSERT INTO customer2 VALUES (default, 'Carr', 'Mary', 87432);
```

Now query the table you just created. The `ID` column has been incremented by (1) again to 104:

```
=> SELECT * FROM customer2;  
  ID | lname | fname | membership_card  
----+-----+-----+-----  
 104 | Carr  | Mary  |           87432  
(1 row)
```

The following example shows how to use `NEXTVAL` in a `SELECT` statement:

```
=> SELECT NEXTVAL('my_seq'), lname FROM customer2;  
NEXTVAL | lname  
-----+-----  
    105 | Carr  
(1 row)
```

As you can see, each time `NEXTVAL` is called, the value increments by 1.

The following example shows how to use `CURRVAL` in a `SELECT` statement:

```
=> SELECT CURRVAL('my_seq'), lname FROM customer2;  
CURRVAL | lname  
-----+-----  
    105 | Carr  
(1 row)
```

The value doesn't change above because the `CURRVAL` function returns only the current value.

**See Also****ALTER SEQUENCE** (page 586)CREATE TABLE **column-constraint** (page 674)**CURRVAL** (page 310)**DROP SEQUENCE** (page 708)**GRANT (Sequence)** (page 725)**NEXTVAL** (page 308)

Using Sequences and Sequence Privileges in the Administrator's Guide

## CREATE TABLE

Creates a table in the logical schema.

**Note:** A default superprojection is automatically created for the table. See "Superprojection Creation" within this topic for details about how it is implemented.

### Syntax

```
CREATE TABLE [ IF NOT EXISTS ] [ schema-name. ] table-name
... { ( column-definition (see "column-definition (table)" on page 670) [ , ...
] )
... | [ column-name-list (see "column-name-list (table)" on page 671) ] AS [ [ AT
EPOCH LATEST ]
... | [ AT TIME 'timestamp' ] ] query }
... [ ORDER BY table-column [ , ... ] ]
... [ ENCODED BY column-definition [ , ... ] ]
... [ hash-segmentation-clause (see "hash-segmentation-clause (table)" on page
679)
... | range-segmentation-clause (see "range-segmentation-clause (table)" on page
680)
... | UNSEGMENTED { NODE node | ALL NODES } ]
... [ KSAFE [ k_num ] ]
... [ PARTITION BY partition-clause ]
```

### Parameters

[ IF NOT EXISTS ]	If supplied, CREATE TABLE does not report an error if a table named <i>table-name</i> already exists.
[ <i>schema-name.</i> ] <i>table-name</i>	<i>table-name</i> specifies the name of the table to be created. <i>Schema-name</i> specifies the schema where the table is created. If <i>schema-name</i> is omitted, the table is created in the first schema listed in the current <b><i>search_path</i></b> . (page 774)
<i>column-definition</i>	Defines one or more columns. See <b><i>column-definition</i></b> (see " <b><i>column-definition (table)</i></b> " on page 670).
ORDER BY <i>table-column</i>	[Optional] Specifies the sort order for the superprojection that is automatically created for the table. If you do not specify the sort order, Vertica uses the order in which columns are specified in the column definition as the sort order for the projection. For example: ORDER BY col2, col1, col5 Note: Data is in ascending order only.
<i>column-name-list</i>	Renames columns when creating a table from a query (CREATE TABLE AS SELECT). See <b><i>column-name-list</i></b> (page 671).

<i>AS query</i>	<p>Creates a new table from the results of a query and fills it with data from the query. For example:</p> <pre>CREATE TABLE promo AS SELECT ... ;</pre> <p>Column renaming is supported as part of the process:</p> <pre>CREATE TABLE promo (name, address, ...) AS SELECT customer_name, customer_address ... ;</pre> <p>The query table-column must be followed by the FROM clause to identify the table from which to copy the columns. See the example at the bottom of this topic as well as the <b>SELECT</b> (page 747) statement.</p> <p>If the query output has expressions other than simple columns (for example, constants, functions, etc) then either an alias must be specified for that expression, or all columns must be listed in the column name list.</p>
AT EPOCH LATEST   AT TIME 'timestamp'	Used with AS query to query historical data. You can specify AT EPOCH LATEST to include data from the latest committed DML transaction or specify a specific epoch based on its time stamp.
ENCODED BY <i>column-definition</i>	<p>[CREATE TABLE AS query Only]</p> <p>This parameter is useful to specify the column encoding and/ or the access rank for specific columns in the query when a column-definition is not used to rename columns for the table to be created. See <b>column-definition</b> (see "<b>column-definition (temp table)</b>" on page 687) for examples.</p> <p>If you rename table columns when creating a table from a query, you can supply the encoding type and access rank in the column name list instead.</p>
<i>hash-segmentation-clause</i>	[Optional] Allows you to segment the superprojection based on a built-in hash function that provides even distribution of data across nodes, resulting in optimal query execution. See <b>hash-segmentation-clause</b> (see " <b>hash-segmentation-clause (table)</b> " on page 679).
<i>range-segmentation-clause</i>	[Optional] Allows you to segment the superprojection based on a known range of values stored in a specific column chosen to provide even distribution of data across a set of nodes, resulting in optimal query execution. See <b>range-segmentation-clause</b> (see " <b>range-segmentation-clause (table)</b> " on page 680).
UNSEGMENTED { NODE <i>node</i>   ALL NODES }	<p>[Optional] Allows you to specify that the projection be unsegmented, as follows:</p> <ul style="list-style-type: none"> <li>▪ NODE <i>node</i>—Creates the unsegmented projection on the specified node only. Dimension table projections must be UNSEGMENTED.</li> <li>▪ ALL NODES—Creates a separate unsegmented projection on each node (automatic replication). To perform distributed query execution, Vertica requires an exact, unsegmented copy of each dimension table superprojection on each node.</li> </ul>
KSAFE [ <i>k</i> ]	[Optional] Specifies the K-safety level of the automatic projection created for the table. The integer K determines how many

	<p>unsegmented or segmented buddy projections are created. The value must be greater than or equal to the current K-safety level of the database and less than the total number of nodes. If KSAFE or its value are not specified, the superprojection is created at the current system K-safety level.</p> <p>For example:</p> <pre>K-SAFE 1</pre> <p>Note: When a hash-segmentation-clause is used with KSAFE, Vertica automatically creates <code>k_num+1</code> buddy projections to meet the K-safety requirement.</p>
PARTITION BY <i>partition-clause</i>	<p>[Not supported for queries (CREATE TABLE AS SELECT)]</p> <ul style="list-style-type: none"> <li>▪ All leaf expressions must be either constants or columns of the table.</li> <li>▪ All other expressions must be functions and operators; aggregate functions and queries are not permitted in the expression.</li> <li>▪ The partition-clause must calculate an idempotent value from its arguments and must be not null.</li> <li>▪ SQL functions used in the partitioning expression must be immutable.</li> </ul>

### Automatic Projection Creation

To get your database up and running quickly, Vertica automatically creates a default projection for each table created through the **CREATE TABLE** (page 664) and **CREATE TEMPORARY TABLE** (page 682) statements. The timing of when the projection is created depends on how you use the CREATE TABLE statement:

- If you create a table without providing the projection-related clauses, a superprojection is automatically created for the table when an INSERT, COPY, or LCOPY command is issued to load data into the table for the first time. The projection is created in the same schema as the table. Once Vertica has created the projection, it loads the data.
- If you use CREATE TABLE AS SELECT to create a table from the results of a query, the table is created first and a projection is created immediately after, using some of the properties of the underlying SELECT query.
- (Advanced users only) If you use any of the following parameters, the default projection is created immediately upon table creation using the specified properties:
  - **column-definition** (page 670) (ENCODING encoding-type and ACCESSRANK integer)
  - ORDER BY table-column
  - **hash-segmentation-clause** (page 679)
  - **range-segmentation-clause** (page 680)
  - UNSEGMENTED { NODE *node* | ALL NODES }
  - KSAFE

**Note:** Before you define a superprojection in the above manner, read Creating Custom Designs in the Administrator's Guide.

## Characteristics of Default Automatic Projections

A default projection has the following characteristics:

- It is a superprojection.
- It uses the default *encoding-type* (page 644) AUTO.
- If the table has one or more primary keys defined, the projection is sorted by these columns. Otherwise, the projection is sorted in the same order as defined in the table column-definition list.
- If the K-safety for the database is zero (K-Safety=0), the projection is unsegmented on the initiator node. If K-Safety is greater than zero (K-Safety>0), the superprojection is replicated (unsegmented) on all nodes. See Segmentation in the Concepts Guide.
- If the projection was created through the CREATE TABLE AS SELECT statement, the projection uses the sort order, segmentation, and encoding specified for the columns in the query table.

Default automatic projections let you get your database up and running quickly; however, they might not necessarily provide the best performance. Vertica recommends that you start with these projections and then use the Database Designer to optimize your database. The Database Designer creates projections that optimize your database based on the characteristics of the data and, optionally, the queries you use.

## Partition Clauses

Creating a table with the partition clause causes all projections anchored on that table to be partitioned according to the partitioning clause. For each partitioned projection, logically, there are as many partitions as the number of unique values returned by the partitioned expression applied over the rows of the projection.

**Note:** Due to the impact on the number of ROS containers, explicit and implicit upper limits are imposed on the number of partitions a projection can have; these limits, however, are detected during the course of operation, such as during COPY.

Creating a partitioned table does not necessarily force all data feeding into a table's projection to be segregated immediately. Logically, the partition clause is applied after the segmented by clause.

Partitioning specifies how data is organized at individual nodes in a cluster and after projection data is segmented; only then is the data partitioned at each node based on the criteria in the partitioning clause.

SQL functions used in the partitioning expression must be immutable, which means they return the exact same value regardless of when it is invoked and independently of session or environment settings, such as LOCALE. For example, the TO\_CHAR function is dependent on locale settings and cannot be used. RANDOM produces different values on each invocation and cannot be used.

Data loaded with the COPY command is automatically partitioned according to the table's PARTITION BY clause.

For more information, see "Restrictions on Partitioning Expressions" in Defining Partitions in the Administrator's Guide

## Notes

- Canceling a CREATE TABLE statement can cause unpredictable results. Vertica recommends that you allow the statement to finish, then use **DROP TABLE** (page 709).

## Examples

The following example creates a table named Product\_Dimension in the Retail schema. It also creates a default superprojection when data is loaded:

```
=> CREATE TABLE Retail.Product_Dimension (  
    Product_Key                integer NOT NULL,  
    Product_Description         varchar(128),  
    SKU_Number                  char(32) NOT NULL,  
    Category_Description        char(32),  
    Department_Description      char(32) NOT NULL,  
    Package_Type_Description    char(32),  
    Package_Size                char(32),  
    Fat_Content                 integer,  
    Diet_Type                   char(32),  
    Weight                      integer,  
    Weight_Units_of_Measure     char(32),  
    Shelf_Width                 integer,  
    Shelf_Height                integer,  
    Shelf_Depth                 integer  
);
```

The following example creates a table named Employee\_Dimension and its associated superprojection in the Public schema. Instead of using the sort order from the column definition, the superprojection uses the sort order specified by the ORDER BY clause. The superprojection is created at the same time as the table because the superprojection is actively defined as part of the CREATE TABLE statement.

```
=> CREATE TABLE Public.Employee_Dimension (  
    Employee_key                integer PRIMARY KEY NOT NULL,  
    Employee_gender              varchar(8) ENCODING RLE,  
    Employee_title               varchar(8),  
    Employee_first_name          varchar(64),  
    Employee_middle_initial      varchar(8),  
    Employee_last_name           varchar(64), )  
ORDER BY Employee_gender, Employee_last_name, Employee_first_name;
```

The following example creates a table called time and partitions the data by year. It also creates a default superprojection when data is loaded:

```
=> CREATE TABLE time( ..., date_col date NOT NULL, ...)  
=> PARTITION BY extract('year' FROM date_col);
```

The following example creates a table named location and partitions the data by state. It also creates a default superprojection when data is loaded:

```
=> CREATE TABLE location(..., state VARCHAR NOT NULL, ...)  
=> PARTITION BY state;
```

The following table uses SELECT AS to create a table called promo and load data from columns in the customer\_dimension table in which the customer's annual\_income is greater than 1,000,000. The data is ordered by state and annual income.

```
=> CREATE TABLE promo
  AS SELECT
    customer_name,
    customer_address,
    customer_city,
    customer_state,
    annual_income
  FROM customer_dimension
  WHERE annual_income>1000000
  ORDER BY customer_state, annual_income;
```

The following table uses **SELECT AS** to create a table called **promo** and load data from the latest committed DML transaction (**AT EPOCH LATEST**).

```
=> CREATE TABLE promo
  AS AT EPOCH LATEST SELECT
    customer_name,
    customer_address,
    customer_city,
    customer_state,
    annual_income
  FROM customer_dimension;
```

The following example demonstrates using the **IF NOT EXISTS** argument to avoid errors if a table already exists. This argument is very useful for SQL scripts where you want to create a table if it does not already exist, and reuse the existing table if it does exist.

```
=> CREATE TABLE t (a INT); -- Create a simple table
CREATE TABLE
=> CREATE TABLE t (a INT, b VARCHAR(256)); -- Generates an error
ROLLBACK: object "t" already exists
=> CREATE TABLE IF NOT EXISTS t (a INT, b VARCHAR(256)); -- No error
NOTICE: object "t" already exists; nothing was done
CREATE TABLE
```

### See Also

Physical Schema in the Concepts Guide

**COPY** (page 607)

**CREATE TEMPORARY TABLE** (page 682)

**DROP PARTITION** (page 408)

**DROP PROJECTION** (page 704)

**DUMP PARTITION KEYS** (page 414)

**DUMP PROJECTION PARTITION KEYS** (page 415)

**DUMP TABLE PARTITION KEYS** (page 416)

**PARTITION PROJECTION** (page 440)

**PARTITION TABLE** (page 441)

**SELECT** (page 747)

## Partitioning Tables and Auto Partitioning in the Administrator's Guide

**column-definition (table)**

A column definition specifies the name, data type, and constraints to be applied to a column.

**Syntax**

```
column-name data-type {
... [ column-constraint (on page 674) [ ... ] | [ table-constraint (on page 678)
] [ ,... ]
... [ ENCODING encoding-type ]
... [ ACCESSRANK integer ] }
```

**Parameters**

<i>column-name</i>	Specifies the name of a column to be created or added.
<i>data-type</i>	Specifies one of the following data types: <b>BINARY</b> (page 65) <b>BOOLEAN</b> (page 69) <b>CHARACTER</b> (page 70) <b>DATE/TIME</b> (page 72) <b>NUMERIC</b> (page 95)
<i>column-constraint</i>	Specifies a <b>column constraint</b> (see " <b>column-constraint</b> " on page 674) to apply to the column.
ENCODING <i>encoding -type</i>	[Optional] Specifies the <b>type of encoding</b> (see " <b>encoding-type</b> " on page 644) to use on the column. By default, the encoding-type is auto. <b>Caution:</b> Using the NONE keyword for strings could negatively affect the behavior of string columns.
ACCESSRANK <i>integer</i>	[Optional] Overrides the default access rank for a column. This is useful if you want to increase or decrease the speed at which a column is accessed. See <a href="#">Creating and Configuring Storage Locations and Prioritizing Column Access Speed in the Administrator's Guide</a> .

**Example**

The following example creates a table named Employee\_Dimension and its associated superprojection in the Public schema. Note that encoding-type RLE is specified for the Employee\_gender column definition:

```
=> CREATE TABLE Public.Employee_Dimension (
    Employee_key                integer PRIMARY KEY NOT NULL,
    Employee_gender              varchar(8) ENCODING RLE,
    Employee_title               varchar(8),
    Employee_first_name          varchar(64),
    Employee_middle_initial      varchar(8),
    Employee_last_name           varchar(64),
);
```

## column-name-list (table)

Is used to rename columns when creating a table from a query (CREATE TABLE AS SELECT). It can also be used to specify the **encoding type** (see "**encoding-type**" on page 644) and access rank of the column.

### Syntax

```
column-name-list
... [ ENCODING encoding-type ]
... [ ACCESSRANK integer ] [ , ... ]
... [ GROUPED ( projection-column-reference [,...] ) ]
```

### Parameters

<i>column-name</i>	Specifies the new name for the column.
ENCODING <i>encoding-type</i>	Specifies the type of encoding to use on the column. By default, the encoding-type is auto. See <b>encoding type</b> (see " <b>encoding-type</b> " on page 644) for a complete list. <b>Caution:</b> Using the NONE keyword for strings could negatively affect the behavior of string columns.
ACCESSRANK <i>integer</i>	Overrides the default access rank for a column. This is useful if you want to increase or decrease the speed at which a column is accessed. See Creating and Configuring Storage Locations and Prioritizing Column Access Speed in the Administrator's Guide.

GROUPED	<p>Groups two or more columns into a single disk file. This minimizes file I/O for work loads that:</p> <ul style="list-style-type: none"> <li>▪ Read a large percentage of the columns in a table.</li> <li>▪ Perform single row look-ups.</li> <li>▪ Query against many small columns.</li> <li>▪ Frequently update data in these columns.</li> </ul> <p>If you have data that is always accessed together and it is not used in predicates, you can increase query performance by grouping these columns. Once grouped, queries can no longer independently retrieve from disk all records for an individual column independent of the other columns within the group.</p> <p><b>Note:</b> RLE compression is reduced when a RLE column is grouped with one or more non-RLE columns.</p> <p>When grouping columns you can:</p> <ul style="list-style-type: none"> <li>▪ Group some of the columns: <ul style="list-style-type: none"> <li>▪ (a, GROUPED(b, c), d)</li> </ul> </li> <li>▪ Group all of the columns: <ul style="list-style-type: none"> <li>▪ (GROUPED(a, b, c, d))</li> </ul> </li> <li>▪ Create multiple groupings in the same projection: <ul style="list-style-type: none"> <li>▪ (GROUPED(a, b), GROUPED(c, d))</li> </ul> </li> </ul> <p><b>Note:</b> Vertica performs dynamic column grouping. For example, to provide better read and write efficiency for small loads, Vertica ignores any projection-defined column grouping (or lack thereof) and groups all columns together by default.</p>
---------	---

### Notes if you are using a query:

Neither the data type nor column constraint can be specified for a column in the column-name-list. These are derived by the columns in the query table identified in the FROM clause. If the query output has expressions other than simple columns (for example, constants, functions, etc) then either an alias must be specified for that expression, or all columns must be listed in the column name list.

You can supply the encoding type and access rank in either the column-name-list or the column list in the query, but not both.

The following statements are both allowed:

```
=> CREATE TABLE promo (state ENCODING RLE ACCESSRANK 1, zip ENCODING RLE, ...)
  AS SELECT * FROM customer_dimension
  ORDER BY customer_state, ... ;
```

```
=> CREATE TABLE promo
  AS SELECT * FROM customer_dimension
  ORDER BY customer_state
```

```

    ENCODED BY customer_state ENCODING RLE ACCESSRANK 1, customer_zip ENCODING RLE
...;

```

The following statement is not allowed because encoding is specified in both column-name-list and ENCODED BY clause:

```

=> CREATE TABLE promo (state ENCODING RLE ACCESSRANK 1, zip ENCODING RLE, ...)
    AS SELECT * FROM customer_dimension
    ORDER BY customer_state
    ENCODED BY customer_state ENCODING RLE ACCESSRANK 1, customer_zip ENCODING RLE
...;

```

### Example

The following example creates a table named `employee_dimension` and its associated superprojection in the public schema. Note that encoding-type RLE is specified for the `employee_gender` column definition:

```

=> CREATE TABLE public.employee_dimension (
    employee_key          INTEGER PRIMARY KEY NOT NULL,
    employee_gender       VARCHAR(8) ENCODING RLE,
    employee_title        VARCHAR(8),
    employee_first_name   VARCHAR(64),
    employee_middle_initial VARCHAR(8),
    employee_last_name    VARCHAR(64)
);

```

Using the Vmart schema, the following example creates a table named `promo` from a query that selects data from columns in the `customer_dimension` table. RLE encoding is specified for the state column in the column name list.

```

=> CREATE TABLE promo (
    name,
    address,
    city,
    state ENCODING RLE, income )
    AS SELECT customer_name,
    customer_address,
    customer_city,
    customer_state,
    annual_income
FROM customer_dimension
WHERE annual_income > 1000000
ORDER BY customer_state, annual_income;

```

## column-constraint

Adds a referential integrity constraint to the metadata of a column. See Adding Constraints in the Administrator's Guide.

### Syntax

```
[ CONSTRAINT constraint-name ] {
...[ NOT ] NULL
...| PRIMARY KEY
...| REFERENCES table-name
...| UNIQUE
...[ DEFAULT default ]
...[ AUTO_INCREMENT ]
...[ IDENTITY [ ( seed , increment , cache ) ] ] }
```

### Parameters

CONSTRAINT <i>constraint-name</i>	Optionally assigns a name to the constraint. Vertica recommends that you name all constraints.
NULL	[Default] Specifies that the column is allowed to contain null values.
NOT NULL	Specifies that the column must receive a value during INSERT and UPDATE operations. If no DEFAULT value is specified and no value is provided, the INSERT or UPDATE statement returns an error because no default value exists.
PRIMARY KEY	Adds a referential integrity constraint defining the column as the primary key.
REFERENCES	Adds a referential integrity constraint defining the column as a foreign key. If column is omitted, the default is the primary key of table.
<i>table-name</i>	Specifies the table to which the REFERENCES constraint applies.
<i>column-name</i>	Specifies the column to which the REFERENCES constraint applies. If column is omitted, the default is the primary key of table-name.
UNIQUE	Ensures that the data contained in a column or a group of columns is unique with respect to all the rows in the table.
DEFAULT <i>default</i>	<p>Specifies a default data value for a column if the column is used in an INSERT operation and no value is specified for the column. If there is no value specified for the column and no default, the default is NULL.</p> <p>Default value usage:</p> <ul style="list-style-type: none"> <li>▪ A default value can be set for a column of any data type.</li> <li>▪ The default value can be any variable-free expression, as long as it matches the data type of the column.</li> <li>▪ Variable-free expressions can contain constants, SQL functions, null-handling functions, system information functions, string functions, numeric functions, formatting functions, nested functions, and all Vertica-supported operators</li> </ul> <p>Default value restrictions:</p>

	<ul style="list-style-type: none"> <li>▪ Expressions can contain only constant arguments.</li> <li>▪ Subqueries and cross-references to other columns in the table are not permitted in the expression.</li> <li>▪ The return value of a default expression cannot be NULL.</li> <li>▪ The return data type of the default expression after evaluation either matches that of the column for which it is defined, or an implicit cast between the two data types is possible. For example, a character value cannot be cast to a numeric data type implicitly, but a number data type can be cast to character data type implicitly.</li> <li>▪ Default expressions, when evaluated, conform to the bounds for the column.</li> <li>▪ Volatile functions are not supported when adding columns to existing tables. See <b>ALTER TABLE</b> (page 589).</li> </ul> <p><b>Note:</b> Vertica attempts to check the validity of default expressions, but some errors might not be caught until run time.</p>
AUTO_INCREMENT	<p>Creates a column within the specified table that consists of values generated by the database. These values cannot be modified. The initial value of this column is 1, and it is incremented by 1 each time a row is added.</p> <p><b>Note:</b> Vertica supports only one AUTO_INCREMENT or IDENTITY column per table.</p>
IDENTITY	<p>Creates an identity column within the specified table that consists of values generated by the database. These values cannot be modified. Identity columns can also be used as primary keys.</p> <p><b>Notes</b></p> <ul style="list-style-type: none"> <li>▪ AUTO_INCREMENT and IDENTITY are identical except that IDENTITY takes extra arguments, and IDENTITY arguments are optional.</li> <li>▪ Vertica supports only one IDENTITY or one AUTO_INCREMENT column per table.</li> </ul>
<i>seed</i>	<p>When used with IDENTITY, seed specifies the value for the first row loaded into the table. Default is 1.</p>
<i>increment</i>	<p>When used with IDENTITY, specifies the value that is added to identity value of the previous row. Default is 1.</p>
<i>cache</i>	<p>When used with IDENTITY, specifies the number of unique numbers to be preallocated and stored in memory for faster access. Default is 250,000 with a minimum value of 1.</p> <p><b>Note:</b> The cache value can contain positive integers only.</p>

### Notes

- IDENTITY arguments are optional; however you cannot specify increment without a seed. Thus, if you supply only one argument, the system assigns a seed value. Two values are seed and increment, and three values are seed, increment, and cache. The following are all valid examples:

```
=> CREATE TABLE t1(x IDENTITY(1,1,9), y INT);
=> CREATE TABLE t1(x IDENTITY(1,1), y INT);
=> CREATE TABLE t1(x IDENTITY(1), y INT);
```

- A FOREIGN KEY constraint can be specified solely by a REFERENCE to the table that contains the PRIMARY KEY. The columns in the referenced table do not need to be explicitly specified; for example:  
=> CREATE TABLE fact (c1 INTEGER PRIMARY KEY NOT NULL);  
=> CREATE TABLE dim (c1 INTEGER REFERENCES fact NOT NULL);
- Columns that are given a PRIMARY constraint must also be set NOT NULL. Vertica automatically sets these columns to be NOT NULL if you do not do so explicitly.
- Vertica supports variable-free expressions in the column DEFAULT clause. See **COPY** (page 607) [*Column as Expression*].
- If you are using a CREATE TABLE AS SELECT statement, the column-constraint parameter does not apply. Column constraints are set by the columns in the query table identified in the FROM clause.
- An auto-increment or identity value is never rolled back even if a transaction that tries to insert a value into a table is not committed.

### Example

The following command creates the `store_dimension` table and sets the default column value for `Store_state` to MA:

```
=> CREATE TABLE store_dimension (store_state CHAR (2) DEFAULT MA);
```

The following command creates the `public.employee_dimension` table and sets the default column value for `hire_date` to `current_date()`:

```
=> CREATE TABLE public.employee_dimension (hire_date DATE DEFAULT  
current_date());
```

The following example uses the IDENTITY column-constraint to create a table with an ID column that has an initial value of 1. It is incremented by 1 every time a row is inserted.

```
=> CREATE TABLE Premium_Customer(  
    ID IDENTITY(1,1),  
    lname VARCHAR(25),  
    fname VARCHAR(25),  
    store_membership_card INTEGER  
);  
=> INSERT INTO Premium_Customer (lname, fname, store_membership_card )  
VALUES ('Gupta', 'Saleem', 475987);
```

Confirm the row you added and see the ID value:

```
=> SELECT * FROM Premium_Customer;  
ID | lname | fname | store_membership_card  
---+-----+-----+-----  
 1 | Gupta | Saleem | 475987  
(1 row)
```

Now add another row:

```
=> INSERT INTO Premium_Customer (lname, fname, store_membership_card)  
VALUES ('Lee', 'Chen', 598742);
```

Calling the `LAST_INSERT_ID` function returns value 2 because you previously inserted a new customer (Chen Lee), and this value is incremented each time a row is inserted:

```
=> SELECT LAST_INSERT_ID();
last_insert_id
-----
                2
```

(1 row)

View all the ID values in the Premium\_Customer table:

```
=> SELECT * FROM Premium_Customer;
ID | lname | fname | store_membership_card
-----+-----+-----+-----
  1 | Gupta | Saleem |                475987
  2 | Lee   | Chen   |                598742
```

(2 rows)

The following example uses the AUTO\_INCREMENT column-constraint to create a table with an ID column that automatically increments every time a row is inserted.

```
=> CREATE TABLE Premium_Customer(
    ID AUTO_INCREMENT,
    lname VARCHAR(25),
    fname VARCHAR(25),
    store_membership_card INTEGER
);
=> INSERT INTO Premium_Customer (lname, fname, store_membership_card )
VALUES ('Gupta', 'Saleem', 475987);
```

Confirm the row you added and see the ID value:

```
=> SELECT * FROM Premium_Customer;
ID | lname | fname | store_membership_card
-----+-----+-----+-----
  1 | Gupta | Saleem |                475987
```

(1 row)

Now add two rows:

```
=> INSERT INTO Premium_Customer (lname, fname, store_membership_card)
VALUES ('Lee', 'Chen', 598742);
=> INSERT INTO Premium_Customer (lname, fname, store_membership_card)
VALUES ('Brown', 'John', 642159);
```

```
=> SELECT * FROM Premium_Customer;
ID | lname | fname | store_membership_card
-----+-----+-----+-----
  1 | Gupta | Saleem |                475987
  2 | Lee   | Chen   |                598742
  3 | Brown | John   |                642159
```

(3 rows)

This time the LAST\_INSERT\_ID returns a value of 3:

```
=> SELECT LAST_INSERT_ID();
LAST_INSERT_ID
-----
                3
```

(1 row)

For additional examples, see **CREATE SEQUENCE** (page 658).

## table-constraint

Adds a join constraint to the metadata of a table. See Adding Constraints in the Administrator's Guide.

### Syntax

```
[ CONSTRAINT constraint_name ]
... [ NOT ] NULL
... { PRIMARY KEY ( column [ , ... ] )
... | FOREIGN KEY ( column [ , ... ] ) REFERENCES table
... | UNIQUE ( column [ , ... ] )
```

### Parameters

CONSTRAINT	Optionally assigns a name to the constraint. Vertica recommends that you name all constraints.
NULL	[Default] Specifies that the column is allowed to contain null values.
NOT NULL	Specifies that the column must receive a value during INSERT and UPDATE operations. If no DEFAULT value is specified and no value is provided, the INSERT or UPDATE statement returns an error because no default value exists.
PRIMARY KEY	Adds a referential integrity constraint defining one or more NOT NULL numeric columns as the primary key.
FOREIGN KEY	Adds a referential integrity constraint defining one or more numeric columns as a foreign key.
REFERENCES	Specifies the table to which the FOREIGN KEY constraint applies. If <i>column</i> is omitted, the default is the primary key of <i>table</i> .
UNIQUE	Ensures that the data contained in a column or a group of columns is unique with respect to all the rows in the table.

### Notes

- A foreign key constraint can be specified solely by a reference to the table that contains the primary key. The columns in the referenced table do not need to be explicitly specified; for example:

```
CREATE TABLE fact(c1 INTEGER PRIMARY KEY);
CREATE TABLE dim (c1 INTEGER REFERENCES fact);
```

- Define PRIMARY KEY and FOREIGN KEY constraints in all tables that participate in inner joins. See Adding Join Constraints.
- Adding constraints to a table that is referenced in a view does not affect the view.

### Examples

The Retail Sales Example Database described in the Getting Started Guide contains a table Product\_Dimension in which products have descriptions and categories. For example, the description "Seafood Product 1" exists only in the "Seafood" category. You can define several similar correlations between columns in the Product Dimension table.

## hash-segmentation-clause (table)

Hash segmentation allows you to segment a projection based on a built-in hash function that provides even distribution of data across some or all of the nodes in a cluster, resulting in optimal query execution.

**Note:** Hash segmentation is the preferred method of segmentation. The Database Designer uses hash segmentation by default.

### Syntax

```
SEGMENTED BY expression
  [ ALL NODES | NODES node [ ,... ] ]
```

### Parameters

SEGMENTED BY <i>expression</i>	Can be a general SQL expression, but there is no reason to use anything other than the built-in <b>HASH</b> (page 287) or <b>MODULARHASH</b> (page 290) functions with table columns as arguments.  Choose columns that have a large number of unique data values and acceptable skew in their data distribution. Primary key columns that meet the criteria could be an excellent choice for hash segmentation.
ALL NODES	Automatically distributes the data evenly across all nodes at the time the projection is created. The ordering of the nodes is fixed.
NODES <i>node</i> [ ,... ]	Specifies a subset of the nodes in the cluster over which to distribute the data. You can use a specific node only once in any projection. For a list of the nodes in a database, use the View Database command in the Administration Tools.

### Notes

- Table column names must be used in the expression, not the projection column names.
- If you want to use a different SEGMENTED BY expression, the following restrictions apply:
  - All leaf expressions must be either constants or **column-references** (see "**Column References**" on page 46) to a column in the SELECT list of the CREATE PROJECTION command
  - Aggregate functions are not allowed
  - The expression must return the same value over the life of the database.
  - The expression must return non-negative INTEGER values in the range  $0 \leq x < 2^{63}$  (two to the sixty-third power or  $2^{63}$ ), and values are generally distributed uniformly over that range.
  - If *expression* produces a value outside the expected range (a negative value for example), no error occurs, and the row is added to the first segment of the projection.
- When a hash-segmentation-clause is used with KSAFE [k\_num], Vertica automatically creates k\_num+1 buddy projections to meet the K-safety requirement.
- The hash-segmentation-clause within the CREATE TABLE statement does not support the OFFSET keyword, which is available in the CREATE PROJECTION command. The OFFSET is set to zero (0).

## Example

This example segments the default superprojection and its buddies for the `Public.Employee_Dimension` table using HASH segmentation across all nodes based on the `Employee_key` column:

```
=> CREATE TABLE Public.Employee_Dimension (
    Employee_key          integer PRIMARY KEY NOT NULL,
    Employee_gender       varchar(8) ENCODING RLE,
    Employee_title        varchar(8),
    Employee_first_name   varchar(64),
    Employee_middle_initial varchar(8),
    Employee_last_name    varchar(64),
)
SEGMENTED BY HASH(Employee_key) ALL NODES;
```

## See Also

**HASH** (page 287) and **MODULARHASH** (page 290)

## range-segmentation-clause (table)

Range segmentation allows you to segment a projection based on a known range of values stored in a specific column chosen to provide even distribution of data across a set of nodes, resulting in optimal query execution.

**Note:** Vertica recommends that you use hash segmentation, instead of range segmentation.

## Syntax

```
SEGMENTED BY expression
    NODE node VALUES LESS THAN value
    ...
    NODE node VALUES LESS THAN MAXVALUE
```

## Parameters (Range Segmentation)

SEGMENTED BY <i>expression</i>	<p>Is a single <b>column reference</b> (see "<b>Column References</b>" on page 46) to a column in the column definition of the CREATE TABLE statement. Choose a column that has:</p> <ul style="list-style-type: none"> <li>▪ INTEGER or FLOAT data type</li> <li>▪ A known range of data values</li> <li>▪ An even distribution of data values</li> <li>▪ A large number of unique data values</li> </ul> <p>Avoid columns that:</p> <ul style="list-style-type: none"> <li>▪ Are foreign keys</li> <li>▪ Are used in query predicates</li> <li>▪ Have a date/time data type</li> <li>▪ Have correlations with other columns due to functional dependencies.</li> </ul> <p><b>Note:</b> Segmenting on DATE/TIME data types is valid but guaranteed to produce temporal skew in the data distribution and is not recommended. If</p>
--------------------------------	--

	you choose this option, do not use TIME or TIMETZ because their range is only 24 hours.
NODE <i>node</i>	Is a symbolic name for a node. You can use a specific node only once in any projection. For a list of the nodes in a database, use <code>SELECT * FROM NODE_RESOURCES</code> .
VALUES LESS THAN <i>value</i>	Specifies that this segment can contain a range of data values <i>less than</i> the specified <i>value</i> , except that segments cannot overlap. In other words, the minimum value of the range is determined by the <i>value</i> of the previous segment (if any).
MAXVALUE	Specifies a sub-range with no upper limit. In other words, it represents a value greater than the maximum value that can exist in the data. The maximum value depends on the data type of the segmentation column.

### Notes

- The `SEGMENTED BY expression` syntax allows a general SQL expression but there is no reason to use anything other than a single **column reference** (see "**Column References**" on page 46) for range segmentation. If you want to use a different expression, the following restrictions apply:
  - All leaf expressions must be either constants or column-references to a column in the SELECT list of the CREATE PROJECTION command
  - Aggregate functions are not allowed
  - The expression must return the same value over the life of the database.
- During INSERT or COPY to a segmented projection, if *expression* produces a value outside the expected range (a negative value for example), no error occurs, and the row is added to a segment of the projection.

### See Also

**NODE\_RESOURCES** (page 861)

## CREATE TEMPORARY TABLE

Creates a temporary table.

**Note:** A default superprojection is automatically created for the temporary table. See "Superprojection Creation" within this topic for details about how it is implemented.

### Syntax

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ]
... TABLE [schema-name].table-name {
... ( column-definition (see "column-definition (temp table)" on page 687) [ , ...
] )
... | [ column-name-list (see "column-name-list (temp table)" on page 688) ] }
... [ ON COMMIT { DELETE | PRESERVE } ROWS ]
... [ AS [ AT EPOCH LATEST ] | [ AT TIME 'timestamp' ] query ]
... [ [ ORDER BY table-column [ , ... ] ]
....[ ENCODED BY column-definition [ , ... ]
....[ hash-segmentation-clause (see "hash-segmentation-clause (temp table)" on
page 690) | range-segmentation-clause (see "range-segmentation-clause (temp
table)" on page 691)
....| UNSEGMENTED { NODE node | ALL NODES } ]
....[ KSAFE [ k-num ] ]
....| [ NO PROJECTION ] ]
```

### Parameters

GLOBAL	[Optional] Specifies that the table definition is visible to all sessions. Temporary table data is visible only to the session that inserts the data into the table. Temporary tables in Vertica default to global.
LOCAL	[Optional] Specifies that the table definition is visible only to the session in which it is created. Temporary tables in Vertica default to global.
TEMPORARY   TEMP	Specifies that the table is a temporary table.
[schema-name].table-name	Specifies the name of the temporary table to be created. For a global temporary table, the user can specify the schema where the table is to be created. If schema-name is omitted, the table is created in the first schema listed in current <b>search_path</b> (page 774). Schema-name is not supported for local temporary tables because they are always created in a special schema.
column-definition	Defines one or more columns. See <b>column-definition</b> (see " <b>column-definition (temp table)</b> " on page 687).
column-name-list	Renames columns when creating a temporary table from a query (CREATE TEMPORARY TABLE AS SELECT). See <b>column-name-list</b> (see " <b>column-name-list (temp table)</b> " on page 688).
ON COMMIT { PRESERVE   DELETE } ROWS	[Optional] Specifies whether data is transaction- or session-scoped: <ul style="list-style-type: none"> <li>DELETE marks the temporary table for transaction-scoped</li> </ul>

	<p>data. Vertica truncates the table (delete all its rows) after each commit. DELETE ROWS is the default.</p> <ul style="list-style-type: none"> <li>▪ PRESERVE marks the temporary table for session-scoped data, which is preserved beyond the lifetime of a single transaction. Vertica truncates the table (delete all its rows) when you terminate a session.</li> </ul>
AT EPOCH LATEST   AT TIME 'timestamp'	Used with AS query to query historical data. You can specify AT EPOCH LATEST to include data from the latest committed DML transaction or specify a specific epoch based on its time stamp.
AS query	<p>[Optional.] Creates a new table from the results of a query and fills it with data from the query as long as ON COMMIT PRESERVE ROWS is specified:</p> <pre>CREATE GLOBAL TEMP TABLE temp_table1 ON COMMIT PRESERVE ROWS AS SELECT ...;</pre> <p>If ON COMMIT DELETE ROWS is specified, the temporary table is created, but data is not inserted from the query:</p> <pre>CREATE GLOBAL TEMP TABLE temp_table1 ON COMMIT DELETE ROWS AS SELECT ...;</pre> <p>Column renaming is supported as part of the process:</p> <pre>CREATE TEMP TABLE temp-table1 (name, address, ...) AS SELECT customer_name, customer_address ... ;</pre>
ORDER BY table-column	<p>[Optional] Specifies the sort order for the superprojection that is automatically created for the table. If you do not specify the sort order, Vertica uses the order in which columns are specified in the column definition as the sort order for the projection. For example:</p> <pre>ORDER BY col2, col1, col5</pre> <p>Note: Data is in ascending order only.</p>
ENCODED BY column-definition	<p>[CREATE TEMPORARY TABLE AS query Only]</p> <p>This parameter is useful to specify the column encoding and/ or the access rank for specific columns in the query when a column-definition is not used to rename columns for the table to be created. See <b>column-definition</b> (see "<b>column-definition (temp table)</b>" on page 687) for examples.</p> <p>If you rename table columns when creating a table from a query, you can supply the encoding type and access rank in the column name list instead.</p>
hash-segmentation-clause	[Optional] Allows you to segment the superprojection based on a built-in hash function that provides even distribution of data across nodes, resulting in optimal query execution. See <b>hash-segmentation-clause</b> (see " <b>hash-segmentation-clause (temp table)</b> " on page 690).
range-segmentation-clause	[Optional] Allows you to segment the superprojection based on a known range of values stored in a specific column chosen to provide even distribution of data across a set of nodes, resulting in optimal query execution. See <b>range-segmentation-clause</b> (see " <b>range-segmentation-clause (temp table)</b> " on page 691).
UNSEGMENTED { NODE node   ALL NODES }	<p>[Optional] Allows you to specify that the projection be unsegmented, as follows:</p> <ul style="list-style-type: none"> <li>▪ NODE node—Creates the unsegmented projection on the</li> </ul>

	<p>specified node only. Dimension table projections must be UNSEGMENTED.</p> <ul style="list-style-type: none"> <li>▪ <b>ALL NODES</b>—Creates a separate unsegmented projection on each node (automatic replication). To perform distributed query execution, Vertica requires an exact, unsegmented copy of each dimension table superprojection on each node.</li> </ul>
KSAFE [ <i>k-num</i> ]	<p>[Optional] Specifies the K-safety level of the automatic projection created for the table. The integer K determines how many unsegmented or segmented buddy projections are created. The value must be greater than or equal to the current K-safety level of the database and less than the total number of nodes. If KSAFE or its value are not specified, the superprojection is created at the current system K-safety level.</p> <p>For example: <code>K-SAFE 1</code></p> <p><b>Note:</b> When a hash-segmentation-clause is used with <code>KSAFE</code>, Vertica automatically creates <code>k_num+1</code> buddy projections to meet the K-safety requirement.</p>
NO PROJECTION	<p>[Optional] Prevents the automatic creation of a default superprojection for the temporary table until data is loaded.</p> <p><code>NO PROJECTION</code> cannot be used with queries (<code>CREATE TEMPORARY TABLE AS SELECT</code>), <code>ORDER BY</code>, <code>ENCODED BY</code>, <code>KSAFE</code>, <b>hash-segmentation clause</b> (page 690), or <b>range-segmentation-clause</b> (page 691).</p>

A common use case for a temporary table is to divide complex query processing into multiple steps. Typically, a reporting tool holds intermediate results while reports are generated (for example, first get a result set, then query the result set, and so on). You can also write subqueries.

**Note:** The default is `ON COMMIT DELETE ROWS`, where data is discarded at the end of the transaction or session.

## Global Temporary Tables

Global temporary tables are created in the public schema, and they are visible to all users and sessions. However, the contents (data) of a global table are private to the transaction or session in which the data was inserted. Data is automatically removed when the transaction commits, rolls back, or the session ends. This allows two users to use the same temporary table, concurrently, but see only data specific to his or her own transactions for the duration of those transactions or sessions.

The definition of a global temporary table persists in the database catalogs until explicitly removed by using the **DROP TABLE** (page 709) statement.

## Local Temporary Tables

A local temporary table is created in the `V_TEMP_SCHEMA` namespace and is transparently inserted into the user's search path. It is visible only to the user who creates the table for the duration of the session in which it is created. When the session ends, the table definition is automatically dropped from the database catalogs.

## Superprojection Creation

When you use the `CREATE TEMPORARY TABLE` command, the table is created first and the default superprojection is created immediately after unless you specify `NO PROJECTION`.

A default projection has the following characteristics:

- It uses the default encoding-type `AUTO`.
- It is automatically unsegmented on the initiator node and pinned if you do not specify a segmentation clause (***hash-segmentation-clause*** (page 690), ***range-segmentation-clause*** (page 691), or `UNSEGMENTED`).
- If the table has one or more primary keys defined, the projection is sorted by these columns. Otherwise, the the projection is sorted in the same order as defined in the table column-definition list.
- Temp tables are not recoverable, so the superprojection is not K-Safe (`K-SAFE=0`), and you cannot make the table K-safe.

Advanced users can modify the default projection created through the `CREATE TEMPORARY TABLE` statement by defining any or all of the following parameters:

- ***column-definition*** (page 687) (`ENCODING encoding-type` and `ACCESSRANK integer`)
- `ORDER BY table-column`
- ***hash-segmentation-clause*** (page 690)
- ***range-segmentation-clause*** (page 691)
- `UNSEGMENTED { NODE node | ALL NODES }`
- `NO PROJECTION`

**Note:** Before you define the superprojection in this manner, read *Creating Custom Designs in the Administrator's Guide*.

## Notes

- You cannot add projections to non-empty, session-scoped temporary tables (`ON COMMIT PRESERVE ROWS`). Make sure that projections exist before you load data. See the "Automatic Projection Creation" in the ***CREATE TABLE*** (page 664) statement.
- Although adding projections is allowed for tables with `ON COMMIT DELETE ROWS` specified, be aware that you could lose all the data.
- The `V_TEMP_SCHEMA` namespace is automatically part of the search path. Thus, temporary table names do not need to be preceded with the schema.
- Queries that involve temporary tables have the same restrictions on SQL support as queries that do not use temporary tables.
- Single-node (pinned to the initiator node only) projections are supported.
- `AT EPOCH LATEST` queries that refer to session-scoped temporary tables work the same as those for transaction-scoped temporary tables. Both return all committed and uncommitted data regardless of epoch. For example, you can commit data from a temporary table in one epoch, advance the epoch, and then commit data in a new epoch.
- Moveout and mergeout operations cannot be used on session-scoped temporary data.

- If you issue the **TRUNCATE TABLE** (page 787) statement on a temporary table, only session-specific data is truncated with no affect on data in other sessions.
- The `DELETE ... FROM TEMP TABLE` syntax does not truncate data when the table was created with `PRESERVE`; it marks rows for deletion. See **DELETE** (page 699) for additional details.
- In general, session-scoped temporary table data is not visible using system (virtual) tables.
- Views are supported for temporary tables.
- `ANALYZE_STATISTICS` (page 385) is not supported for temporary tables.
- Table partitions are not supported for temporary tables.
- Temporary tables do not recover. If a node fails, queries that use the temporary table also fail. Restart the session and populate the temporary table.
- Pre-join projections that refer to both temporary and non-temporary tables are not supported.
- You cannot use temporary tables as dimensions when the fact table is non-temporary in pre-join projections. All dimension tables in a pre-join projection must be at least as persistent as the fact table and other snowflake dimension tables. The persistence scale is: Normal > session-scoped temporary table > transaction-scoped temporary table.
  - If the the anchor table of a pre-join projection is a transaction-scoped temp table, you can use any type of table (temp or normal) in the pre-join.
  - If the anchor table of a pre-join is a normal table, you can use only normal tables as dimension tables.
  - If there is a snowflake dimension session-scoped temporary table, its dimension tables may not be a transaction-scoped temporary tables.

## Examples

Session-scoped rows in a GLOBAL temporary table can be preserved for the whole session or for the current transaction only. For example, in the first statement below, `ON COMMIT DELETE ROWS` indicates that data be deleted at the end of the transaction.

```
=> CREATE GLOBAL TEMP TABLE temp_table1 (  
    x NUMERIC,  
    y NUMERIC )  
    ON COMMIT DELETE ROWS;
```

By contrast, `ON COMMIT PRESERVE ROWS` indicates that data be preserved until the end of the session.

```
=> CREATE GLOBAL TEMP TABLE temp_table2 (  
    x NUMERIC,  
    y NUMERIC )  
    ON COMMIT PRESERVE ROWS;
```

The following example specifies that the superprojection created for the temp table use RLE encoding for the y column:

```
=> CREATE LOCAL TEMP TABLE temp_table1 (  
    x NUMERIC,  
    y NUMERIC ENCODING RLE )  
    ON COMMIT DELETE ROWS;
```

The following example specifies that the superprojection created for the temp table use the sort order specified by the ORDER BY clause rather than the order of columns in the column list.

```
=> CREATE GLOBAL TEMP TABLE temp_table1 (
    x NUMERIC,
    y NUMERIC ENCODING RLE,
    b VARCHAR(8),
    z VARCHAR(8) )
ORDER BY z, x;
```

### See Also

**ALTER TABLE** (page 589), **CREATE TABLE** (page 664), **DELETE** (page 699), **DROP TABLE** (page 709)

Subqueries in the Programmer's Guide

Transactions in the Concepts Guide

## column-definition (temp table)

A column definition specifies the name, data type, default, and other characteristics to be applied to a column.

### Syntax

```
column-name data-type [ DEFAULT ] [ NULL | NOT NULL ]
[ ENCODING encoding-type ] [ ACCESSRANK integer ] ]
```

### Parameters

<i>column-name</i>	Specifies the name of the temporary table to be created.
<i>data-type</i>	Specifies one of the following data types: <ul style="list-style-type: none"> <li>▪ BINARY</li> <li>▪ BOOLEAN</li> <li>▪ CHARACTER</li> <li>▪ DATE/TIME</li> <li>▪ NUMERIC</li> </ul>
DEFAULT <i>default</i>	Specifies a default data value for a column if the column is used in an INSERT operation and no value is specified for the column. If there is no value specified for the column and no default, the default is NULL. <p>Default value usage:</p> <ul style="list-style-type: none"> <li>▪ A default value can be set for a column of any data type.</li> <li>▪ The default value can be any variable-free expression, as long as it matches the data type of the column.</li> <li>▪ Variable-free expressions can contain constants, SQL functions, null-handling functions, system information functions, string functions, numeric functions, formatting functions, nested functions, and all Vertica-supported operators</li> </ul> <p>Default value restrictions:</p>

	<ul style="list-style-type: none"> <li>▪ Expressions can contain only constant arguments.</li> <li>▪ Subqueries and cross-references to other columns in the table are not permitted in the expression.</li> <li>▪ The return value of a default expression cannot be NULL.</li> <li>▪ The return data type of the default expression after evaluation either matches that of the column for which it is defined, or an implicit cast between the two data types is possible. For example, a character value cannot be cast to a numeric data type implicitly, but a number data type can be cast to character data type implicitly.</li> <li>▪ Default expressions, when evaluated, conform to the bounds for the column.</li> <li>▪ Volatile functions are not supported when adding columns to existing tables. See <b>ALTER TABLE</b> (page 589).</li> </ul> <p><b>Note:</b> Vertica attempts to check the validity of default expressions, but some errors might not be caught until run time.</p>
NULL	[Default] Specifies that the column is allowed to contain null values.
NOT NULL	Specifies that the column must receive a value during INSERT and UPDATE operations. If no DEFAULT value is specified and no value is provided, the INSERT or UPDATE statement returns an error because no default value exists.
ENCODING <i>encoding-type</i>	[Optional] Specifies the <b>type of encoding</b> (see " <b>encoding-type</b> " on page 644) to use on the column. By default, the encoding-type is auto. <b>Caution:</b> Using the NONE keyword for strings could negatively affect the behavior of string columns.
ACCESSRANK <i>integer</i>	[Optional] Overrides the default access rank for a column. This is useful if you want to increase or decrease the speed at which a column is accessed. See <a href="#">Creating and Configuring Storage Locations and Prioritizing Column Access Speed in the Administrator's Guide</a> .

## column-name-list (temp table)

A column name list is used to rename columns when creating a temporary table from a query (CREATE TEMPORARY TABLE AS SELECT). It can also be used to specify the **encoding type** (see "**encoding-type**" on page 644) and access rank of the column.

### Syntax

```
column-name-list [ ENCODING encoding-type ] [ ACCESSRANK integer ] [ , ... ]
  [ GROUPED( projection-column-reference [,...] ) ]
```

### Parameters

<i>column-name-list</i>	Specifies the new name for the column.
ENCODING <i>encoding-type</i>	[Optional] Specifies the type of encoding to use on the column. By default, the encoding-type is auto. See <b>encoding type</b> (see " <b>encoding-type</b> " on

	<p>page 644) for a complete list.</p> <p>Caution: Using the NONE keyword for strings could negatively affect the behavior of string columns.</p>
ACCESSRANK <i>integer</i>	<p>[Optional] Overrides the default access rank for a column. This is useful if you want to increase or decrease the speed at which a column is accessed. See <a href="#">Creating and Configuring Storage Locations and Prioritizing Column Access Speed</a> in the Administrator's Guide.</p>
GROUPED	<p>Groups two or more columns into a single disk file. This minimizes file I/O for work loads that:</p> <ul style="list-style-type: none"> <li>▪ Read a large percentage of the columns in a table.</li> <li>▪ Perform single row look-ups.</li> <li>▪ Query against many small columns.</li> <li>▪ Frequently update data in these columns.</li> </ul> <p>If you have data that is always accessed together and it is not used in predicates, you can increase query performance by grouping these columns. Once grouped, queries can no longer independently retrieve from disk all records for an individual column independent of the other columns within the group.</p> <p><b>Note:</b> RLE compression is reduced when a RLE column is grouped with one or more non-RLE columns.</p> <p>When grouping columns you can:</p> <ul style="list-style-type: none"> <li>▪ Group some of the columns: <ul style="list-style-type: none"> <li>▪ (a, GROUPED(b, c), d)</li> </ul> </li> <li>▪ Group all of the columns: <ul style="list-style-type: none"> <li>▪ (GROUPED(a, b, c, d))</li> </ul> </li> <li>▪ Create multiple groupings in the same projection: <ul style="list-style-type: none"> <li>▪ (GROUPED(a, b), GROUPED(c, d))</li> </ul> </li> </ul> <p><b>Note:</b> Vertica performs dynamic column-grouping. For example, to provide better read and write efficiency for small loads, Vertica ignores any projection-defined column grouping (or lack thereof) and groups all columns together by default.</p>

**Notes:**

If you are using a CREATE TEMPORARY TABLE AS SELECT statement:

- The data-type cannot be specified for a column in the column name list. It is derived by the column in the query table identified in the FROM clause
- You can supply the encoding type and access rank in either the column name list or the column list in the query, but not both.

The following statements are both allowed:

```
=> CREATE TEMPORARY TABLE temp_table1 (state ENCODING RLE ACCESSRANK 1, zip
ENCODING RLE, ...)
AS SELECT * FROM customer_dimension
ORDER BY customer_state, ... ;
=> CREATE TEMPORARY TABLE temp_table1
AS SELECT * FROM customer_dimension
ORDER BY customer_state
```

```
    ENCODED BY customer_state ENCODING RLE ACCESSRANK 1, customer_zip ENCODING
    RLE ...;
```

The following statement is not allowed:

```
=> CREATE TEMPORARY TABLE temp_table1 (state ENCODING RLE ACCESSRANK 1, zip
    ENCODING RLE, ...)
    AS SELECT * FROM customer_dimension
    ORDER BY customer_state
    ENCODED BY customer_state ENCODING RLE ACCESSRANK 1, customer_zip ENCODING
    RLE ...;
```

### Example

The following example creates a temporary table named temp\_table2 and its associated superprojection. Note that encoding-type RLE is specified for the y column definition:

```
=> CREATE GLOBAL TEMP TABLE temp_table2 (
    x NUMERIC,
    y NUMERIC ENCODING RLE,
    b VARCHAR(8),
    z VARCHAR(8) );
```

The following example creates a table named temp\_table3 from a query that selects data from columns in the customer\_dimension table. RLE encoding is specified for the state column in the column name list.

```
=> CREATE TABLE temp_table3 (name, address, city, state ENCODING RLE, income)
    AS SELECT
        customer_name,
        customer_address,
        customer_city,
        customer_state,
        annual_income
    FROM customer_dimension
    WHERE annual_income > 1000000
    ORDER BY customer_state, annual_income;
```

### hash-segmentation-clause (temp table)

By default, the superprojection for the temp table is unsegmented on the initiator node (a pinned projection). If you prefer, you can choose either hash-segmentation (preferred) or range-segmentation if you have more than one node.

Hash segmentation allows you to segment a projection based on a built-in hash function that provides even distribution of data across some or all of the nodes in a cluster, resulting in optimal query execution. Projections created in this manner are not pinned.

**Note:** Hash segmentation is the preferred method of segmentation. The Database Designer uses hash segmentation by default.

### Syntax

```
SEGMENTED BY expression
```

```
[ ALL NODES | NODES node [ ,... ] ]
```

## Parameters

SEGMENTED BY <i>expression</i>	Can be a general SQL expression, but there is no reason to use anything other than the built-in <b>HASH</b> (page 287) or <b>MODULARHASH</b> (page 290) functions with table columns as arguments. Choose columns that have a large number of unique data values and acceptable skew in their data distribution. Primary key columns that meet the criteria could be an excellent choice for hash segmentation.
ALL NODES	Automatically distributes the data evenly across all nodes at the time the projection is created. The ordering of the nodes is fixed.
NODES <i>node</i> [ ,... ]	Specifies a subset of the nodes in the cluster over which to distribute the data. You can use a specific node only once in any projection. For a list of the nodes in a database, use the View Database command in the Administration Tools.

## Notes

- Table column names must be used in the expression, not the projection column names.
- If you want to use a different SEGMENTED BY expression, the following restrictions apply:
  - All leaf expressions must be either constants or **column-references** (see "**Column References**" on page 46) to a column in the SELECT list of the CREATE PROJECTION command
  - Aggregate functions are not allowed
  - The expression must return the same value over the life of the database.
  - The expression must return non-negative INTEGER values in the range  $0 \leq x < 2^{63}$  (two to the sixty-third power or  $2^{63}$ ), and values are generally distributed uniformly over that range.
  - If *expression* produces a value outside the expected range (a negative value for example), no error occurs, and the row is added to the first segment of the projection.
- The hash-segmentation-clause within the CREATE TEMP TABLE statement does not support the OFFSET keyword, which is available in the CREATE PROJECTION command. The OFFSET is set to zero (0).

## Example

This example segments the default superprojection and its buddies using HASH segmentation based on column 1 (C1).

```
=> CREATE TEMPORARY TABLE ... SEGMENTED BY HASH(C1) ALL NODES;
```

## See Also

**HASH** (page 287) and **MODULARHASH** (page 290)

## range-segmentation-clause (temp table)

By default, the superprojection for the temp table is unsegmented on the initiator node (a pinned projection). If you prefer, you can choose either hash-segmentation (preferred) or range-segmentation if you have more than one node.

Range segmentation allows you to segment a projection based on a known range of values stored in a specific column chosen to provide even distribution of data across a set of nodes, resulting in optimal query execution. Projections created in this manner are not pinned.

**Note:** Vertica recommends that you use hash segmentation, instead of range segmentation.

## Syntax

```
SEGMENTED BY expression
  NODE node VALUES LESS THAN value
  :
  NODE node VALUES LESS THAN MAXVALUE
```

## Parameters (Range Segmentation)

SEGMENTED BY <i>expression</i>	<p>Is a single <b>column reference</b> (see "<b>Column References</b>" on page 46) to a column in the SELECT list of the CREATE PROJECTION statement. Choose a column that has:</p> <ul style="list-style-type: none"> <li>▪ INTEGER or FLOAT data type</li> <li>▪ A known range of data values</li> <li>▪ An even distribution of data values</li> <li>▪ A large number of unique data values</li> </ul> <p>Avoid columns that:</p> <ul style="list-style-type: none"> <li>▪ Are foreign keys</li> <li>▪ Are used in query predicates</li> <li>▪ Have a date/time data type</li> <li>▪ Have correlations with other columns due to functional dependencies.</li> </ul> <p><b>Note:</b> Segmenting on DATE/TIME data types is valid but guaranteed to produce temporal skew in the data distribution and is not recommended. If you choose this option, do not use TIME or TIME TZ because their range is only 24 hours.</p>
NODE <i>node</i>	<p>Is a symbolic name for a node. You can use a specific node only once in any projection. For a list of the nodes in a database, use <code>SELECT * FROM NODE_RESOURCES</code>.</p>
VALUES LESS THAN <i>value</i>	<p>Specifies that this segment can contain a range of data values <i>less than</i> the specified <i>value</i>, except that segments cannot overlap. In other words, the minimum value of the range is determined by the <i>value</i> of the previous segment (if any).</p>
MAXVALUE	<p>Specifies a sub-range with no upper limit. In other words, it represents a value greater than the maximum value that can exist in the data. The maximum value depends on the data type of the segmentation column.</p>

## Notes

- The `SEGMENTED BY expression` syntax allows a general SQL expression but there is no reason to use anything other than a single **column reference** (see "**Column References**" on page 46) for range segmentation. If you want to use a different expression, the following restrictions apply:

- All leaf expressions must be either constants or column-references to a column in the SELECT list of the CREATE PROJECTION command
- Aggregate functions are not allowed
- The expression must return the same value over the life of the database.
- During INSERT or COPY to a segmented projection, if *expression* produces a value outside the expected range (a negative value for example), no error occurs, and the row is added to a segment of the projection.

### See Also

**NODE\_RESOURCES** (page 861)

## CREATE TRANSFORM FUNCTION

Adds a User Defined Transform Function (UDTF) stored in a shared Linux library to the catalog. You must have already loaded this library using the **CREATE LIBRARY** (page 634) statement. When you call the SQL function, Vertica passes the input table to the transform function in the library to process.

### Syntax

```
CREATE TRANSFORM FUNCTION function-name
... AS LANGUAGE 'language' NAME 'factory'
... LIBRARY library_name;
```

### Parameters

<i>function-name</i>	The name to assign to the UDTF. This is the name you use in your SQL statements to call the function.
LANGUAGE ' <i>language</i> '	The programming language used to write the UDTF. Currently, only 'C++' is supported.
NAME ' <i>factory</i> '	The name of the C++ factory class in the shared library that generates the object to handle the function's processing.
LIBRARY <i>library_name</i>	The name of the shared library that contains the C++ object to perform the processing for this function. This library must have been previously loaded using the <b>CREATE LIBRARY</b> (page 634) statement.

### Notes

- The parameters and return value for the function are automatically determined by the CREATE TRANSFORM FUNCTION statement, based on data supplied by the factory class.
- You can assign multiple functions the same name if they have different parameters. When you call a multiply-defined function, Vertica selects the UDF function whose input parameters match the parameters in the function call to perform the processing. This behavior is similar to having multiple signatures for a method or function in other programming languages.

- You can return a list of all UDTFs by querying the system table `V_CATALOG.USER_TRANSFORMS` (page 824). Users see only the functions on which they have `EXECUTE` privileges.

### Permissions

- Only the superuser can create or drop a function that uses a UDF library.
- To use a UDTF, the user must have `USAGE` privileges on the schema and `EXECUTE` privileges on the defined function. See ***GRANT (Function)*** (page 721) and ***REVOKE (Function)*** (page 735).

## CREATE USER

Adds a name to the list of authorized database users.

### Syntax

```
CREATE USER name
... [ ACCOUNT {LOCK | UNLOCK} ]
... [ IDENTIFIED BY 'password' ]
... [ PASSWORD EXPIRE ]
... [ MEMORYCAP {'memory-limit' | NONE} ]
... [ PROFILE {profile | DEFAULT} ]
... [ RESOURCE POOL pool-name ]
... [ RUNTIMECAP {'time-limit' | NONE} ]
... [ TEMPSPACECAP {'space-limit' | NONE} ]
```

### Parameters

name	<p>Specifies the name of the user to create; names that contain special characters must be double-quoted.</p> <p><b>Tip:</b> Vertica database user names are logically separate from user names of the operating system in which the server runs. If all the users of a particular server also have accounts on the server's machine, it makes sense to assign database user names that match their operating system user names. However, a server that accepts remote connections could have many database users who have no local operating system account, and in such cases there need be no connection between database user names and OS user names.</p>
ACCOUNT LOCK   UNLOCK	<p>Locks or unlocks the account. Specifying <code>LOCK</code> prevents the user from logging in. Specifying <code>UNLOCK</code> unlocks the account, allowing the user to log in. In addition to manually locking an account, an account can be locked when a user has more failed login attempts that is allowed.</p>
IDENTIFIED BY ' <i>password</i> '	<p>Sets the password for the user. If this parameter is omitted, then the user does not have a password and is not prompted for one when connecting. If a password is supplied, it must conform to the password complexity policy set by the user's profile (either the one specified in the <code>PROFILE</code> parameter, or the default profile if the <code>PROFILE</code> parameter is omitted).</p>
PASSWORD EXPIRE	<p>Expires the user's password immediately. The user will be forced to change the password when he or she next logs in. The grace period setting (if any) in the user's profile is overridden.</p> <p><b>Note:</b> <code>PASSWORD EXPIRE</code> has no effect when using external password authentication methods such as LDAP or Kerberos.</p>
MEMORYCAP ' <i>memory-limit</i> '   NONE	<p>Limits the amount of memory that the user's requests can use. This value is a number representing the amount of space, followed by a unit (for example, '10G'). The unit can be one of the following:</p>

	<ul style="list-style-type: none"> <li>▪ % percentage of total memory available to the Resource Manager. (In this case value for the size must be 0-100)</li> <li>▪ K Kilobytes</li> <li>▪ M Megabytes</li> <li>▪ G Gigabytes</li> <li>▪ T Terabytes</li> </ul> <p>Setting this value to <code>NONE</code> means the user's sessions have no limits on memory use. This is the default value.</p>
<code>PROFILE <i>profile</i>   DEFAULT</code>	Assigns the user to the profile named <i>profile</i> . Profiles set the user's password policy. See Profiles in the Administrator's Guide for details. Using the value <code>DEFAULT</code> here assigns the user to the default profile. If this parameter is omitted, the user is assigned to the default profile.
<code>RESOURCE POOL <i>pool-name</i></code>	Sets the name of the resource pool from which to request the user's resources. This command creates a usage grant for the user on the resource pool unless the resource pool is publicly usable.
<code>RUNTIMECAP '<i>time-limit</i>'   NONE</code>	Sets the maximum amount of time any of the user's queries can execute. <i>time-limit</i> is an interval, such as '1 minute' or '100 seconds' (see <b>Interval Values</b> (page 30) for details). The maximum duration allowed is one year. Setting this value to <code>NONE</code> means there is no time limit on the user's queries.
<code>TEMPSPACECAP '<i>space-limit</i>'   NONE</code>	Limits the amount of temporary file storage the user's requests can use. This parameter's value has the same format as the <code>MEMORYCAP</code> value.

## Notes

- Only a superuser can create a user.
- User names created with double-quotes are case sensitive. For example:  
=> `CREATE USER "FrEd1";`  
In the above example, the login name must be an exact match. If the user name was created without double-quotes (for example, `FRED1`), then the user can log in as `FRED1`, `FrEd1`, `fred1`, and so on.  
**Note:** `ALTER USER` (page 594) and `DROP USER` (page 711) are case-*insensitive*.
- Newly-created users do not have access to schema `PUBLIC` by default. Make sure to `GRANT USAGE ON SCHEMA PUBLIC` to all users you create.
- You can change a user password by using the `ALTER USER` statement. If you want to configure a user to not have any password authentication, you can set the empty password "" in `CREATE` or `ALTER USER` statements, or omit the `IDENTIFIED BY` parameter in `CREATE USER`.
- By default, users have the right to create temporary tables in the database.

## Examples

```
=> CREATE USER Fred;
=> GRANT USAGE ON SCHEMA PUBLIC to Fred;
```

**See Also**

**ALTER USER** (page 594) and **DROP USER** (page 711)

Managing Workloads in the Administrator's Guide

**CREATE VIEW**

Defines a new view.

**Syntax**

```
CREATE [ OR REPLACE ] VIEW viewname [ ( column-name [, ...] ) ] AS query ]
```

**Parameters**

[ OR REPLACE ]	When you supply this option, Vertica overwrites any existing view with the name <i>viewname</i> . If you do not supply this option and a view with that name already exists, CREATE VIEW returns an error.
<i>viewname</i>	Specifies the name of the view to create. The view name must be unique. Do not use the same name as any table, view, or projection within the database. If the view name is not provided, the user name is used as the view name.
<i>column-name</i>	[Optional] Specifies the list of names to be used as column names for the view. Columns are presented from left to right in the order given. If not specified, Vertica automatically deduces the column names from the query.
<i>query</i>	Specifies the query that the view executes. Vertica also uses the query to deduce the list of names to be used as columns names for the view if they are not specified. Use a <b>SELECT</b> (page 747) statement to specify the query. The SELECT statement can refer to tables, temp tables, and other views.

**Notes**

Views are read only. You cannot perform insert, update, delete, or copy operations on a view.

When Vertica processes a query that contains a view, the view is treated as a subquery because the view name is replaced by the view's defining query. The following example defines a view (ship) and illustrates how a query that refers to the view is transformed.

**View:** CREATE VIEW ship AS SELECT \* FROM public.shipping\_dimension;

**Original Query:** SELECT \* FROM ship;

**Transformed query:** SELECT \* FROM (SELECT \* FROM public.shipping\_dimension) AS ship;

Use the **DROP VIEW** (page 711) statement to drop a view. Only the specified view is dropped. Vertica does not support CASCADE functionality for views, and it does not check for dependencies. Dropping a view causes any view that references it to fail.

**Restrictions**

To create a view, the user must be a superuser or have the following privileges:

- **CREATE** on the schema in which the view is created.
- **SELECT** on all the tables and views referenced within the view's defining query.
- **USAGE** on all the schemas that contain the tables and views referenced within the view's defining query.

### Example

```
=> CREATE VIEW myview AS
  SELECT SUM(annual_income), customer_state
  FROM public.customer_dimension
  WHERE customer_key IN
    (SELECT customer_key
     FROM store.store_sales_fact)
  GROUP BY customer_state
  ORDER BY customer_state ASC;
```

The following example uses the *myview* with a **WHERE** clause that limits the results to combined salaries of greater than 2,000,000,000.

```
=> SELECT * FROM myview WHERE SUM > 2000000000;
      SUM      | customer_state
-----+-----
 2723441590 | AZ
 29253817091 | CA
 4907216137  | CO
 3769455689  | CT
 3330524215  | FL
 4581840709  | IL
 3310667307  | IN
 2793284639  | MA
 5225333668  | MI
 2128169759  | NV
 2806150503  | PA
 2832710696  | TN
 14215397659 | TX
 2642551509  | UT
(14 rows)
```

### See Also

**SELECT** (page 747)

**DROP VIEW** (page 711), **GRANT (View)** (page 728)

**REVOKE (View)** (page 742)

## DELETE

Marks tuples as no longer valid in the current epoch. `DELETE` does not delete data from disk storage for base tables. By default, delete uses the WOS and if the WOS fills up overflows to the ROS.

### Syntax

```
DELETE [ /*+ direct */ ] FROM [schema_name.]table WHERE clause (on page 752)
```

### Parameters

<code>/*+ direct */</code>	Writes the data directly to disk (ROS) bypassing memory (WOS). <b>Note:</b> If you delete using the <code>direct</code> hint, you still need to issue a <code>COMMIT</code> or <code>ROLLBACK</code> command to finish the transaction.
<code>[schema_name.]</code>	Specifies the name of an optional schema.
<code>table</code>	Specifies the name of a base table or temporary table.

### Notes

- Subqueries and joins are permitted in `DELETE` statements, which is useful for deleting values in a table based on values that are stored in other tables. See Examples section below.

The delete operation deletes rows that satisfy the `WHERE` clause from the specified table. If the `WHERE` clause is absent, all table rows are deleted. The result is a valid, even though the statement leaves an empty table. On successful completion, a delete operation returns a count, which represents the number of rows deleted. A count of 0 is not an error; it means that no rows matched the condition.

- To remove all rows from a temporary table, use a `DELETE` statement with no `WHERE` clause. In this special case, the rows are not stored in the system, which greatly improves performance. The effect is similar to when a `COMMIT` is issued, in that all rows are removed, but the columns, projections, and constraints are preserved, thus making it easy to re-populate the table.

If you include a `WHERE` clause when performing delete operations on temporary tables, `DELETE` behaves the same as for base tables, marking all delete vectors for storage, and you lose any performance benefits.

`DELETE FROM temp_table` is the only way to truncate a temporary table without ending the transaction.

- If the delete operation succeeds on temporary tables, you cannot roll back to a prior savepoint.
- `DELETE` marks records for deletion in the WOS.
- You cannot delete records from a projection.
- When using more than one schema, specify the schema that contains the table in your `DELETE` statement.

- To use `DELETE` or `UPDATE` (page 792) commands with a `WHERE` clause, the user must have both `SELECT` (page 747) and `DELETE` privileges on the table.

## Examples

The following command truncates a temporary table called `temp1`:

```
=> DELETE FROM temp1;
```

The following command deletes all records from base table `T` where `C1 = C2 - C1`.

```
=> DELETE FROM T WHERE C1=C2-C1;
```

The following command deletes all records from the customer table in the retail schema where the state attribute is in `MA` or `NH`:

```
=> DELETE FROM retail.customer WHERE state IN ('MA', 'NH');
```

For examples on how to nest a subquery within a `DELETE` statement, see Subqueries in `UPDATE` and `DELETE` in the Programmer's Guide.

## See Also

***DROP TABLE*** (page 709) and ***TRUNCATE TABLE*** (page 787)

Deleting Data and Best Practices for `DELETE` and `UPDATE` in the Administrator's Guide

## DISCONNECT

Closes a previously-established connection to another Vertica database.

### Syntax

```
DISCONNECT database-name
```

### Parameters

<i>database-name</i>	The name of the database whose connection should be closed
----------------------	--

### Example

```
=> DISCONNECT ExampleDB;  
DISCONNECT
```

## See Also

***CONNECT*** (page 605)

## DROP FUNCTION

Drops a SQL Macro or User Defined Function (UDF) from the Vertica catalog.

### Syntax

```
DROP FUNCTION [ schema-name. ] function-name [, ...]
```

```
... ( [ [ argname ] argtype [, ...] ] )
```

## Parameters

<code>[<i>schema-name</i>.]<i>function-name</i></code>	Specifies a name of the SQL Macro (function) to drop. If the function name is schema-qualified, the function is dropped from the specified schema
<code><i>argname</i></code>	Specifies the name of the argument, typically a column name.
<code><i>argtype</i></code>	Specifies the data type for argument(s) that are passed to the function. Argument types must match Vertica type names. See <b>SQL Data Types</b> (page 64).

## Notes

- To drop a function, you must specify the argument types because there could be several functions that share the same name with different parameters.
- Vertica does not check for dependencies, so if you drop a SQL Macro where other objects reference it (such as views or other SQL Macros), Vertica returns an error when those objects are used and not when the function is dropped.

## Permissions

Only the superuser or owner can drop the function.

## Example

The following command drops the `zerowhennull` function in the `macros` schema:

```
=> DROP FUNCTION macros.zerowhennull(x INT);
DROP FUNCTION
```

## See Also

**ALTER FUNCTION** (page 578)

**CREATE FUNCTION** (page 629)

**GRANT (Function)** (page 721)

**REVOKE (Function)** (page 735)

**V\_CATALOG.USER\_FUNCTIONS** (page 822)

Using SQL Macros in the Programmer's Guide

## DROP LIBRARY

Removes a shared library from the database. The user defined functions (UDFs) in the library are no longer available. See Using User Defined Functions in the Programmer's Guide for details.

## Syntax

```
DROP LIBRARY [schema.]library_name [CASCADE]
```

## Parameters

<code>[<i>schema</i>.]<i>library_name</i></code>	The name of the library to drop. This must be the same name given to <b>CREATE LIBRARY</b> (page 634) to load the library.
<code>[CASCADE]</code>	Drops any functions that have been defined using the library. This statement fails if CASCADE is not specified and there is one or more UDFs that have been defined using the library.

## Notes

- Only the superuser can drop a library.
- The library file is deleted from the managed directories on the Vertica nodes.

## Example

To drop the library named MyFunctions:

```
=> DROP LIBRARY MyFunctions CASCADE;
```

## DROP PROCEDURE

Removes an external procedure from Vertica.

## Syntax

```
DROP PROCEDURE [schema-name.]name ( [ argname ] argtype [, ...] )
```

## Parameters

<code>[<i>schema-name</i>.]</code>	[Optional] Specifies the name of a schema. When using more than one schema, specify the schema that contains the procedure to drop.
<code><i>name</i></code>	Specifies the name of the procedure to be dropped.
<code><i>argname</i></code>	The argument name or names used when creating the procedure.
<code><i>argtype</i></code>	The argument type or types used when creating the procedure.

## Note

- Only the database superuser can drop procedures.
- Only the reference to the procedure is removed. The external file remains in the `<database>/procedures` directory on each node in the database.

## Example

```
=> DROP PROCEDURE helloplanet(arg1 varchar);
```

**See Also**

**CREATE PROCEDURE** (page 635)

## DROP PROFILE

Removes a profile from the database. Only the superuser can drop a profile.

**Syntax**

```
DROP PROFILE name [, ...] [ CASCADE ]
```

**Parameters**

<i>name</i>	The name of one or more profiles (separated by commas) to be removed.
CASCADE	Moves all users assigned to the profile or profiles being dropped to the DEFAULT profile. If you do not include CASCADE in the DROP PROFILE command and a targeted profile has users assigned to it, the command returns an error.

**Note:** You cannot drop the DEFAULT profile.

## DROP PROJECTION

Marks a projection to be dropped from the catalog so it is unavailable to user queries.

### Syntax

```
DROP PROJECTION { base-projname | projname-node [ , ... ] }
... [ RESTRICT | CASCADE ]
```

### Parameters

<i>base-projname</i>	Drops the base projection and all its replicated buddies on all nodes simultaneously. When using more than one schema, specify the schema that contains the projection. projname can be 'projname' or 'schema.projname'.
<i>projname-node</i>	Drops only the specified projection on the specified node. When using more than one schema, specify the schema that contains the projection. projname can be 'projname' or 'schema.projname'.
RESTRICT	Drops the projection only if it does not contain any objects. RESTRICT is the default.
CASCADE	Drops the projection even if it contains one or more objects.

### Notes

To prevent data loss and inconsistencies, tables must contain one superprojection, so DROP PROJECTION fails if a projection is the table's only superprojection. In such cases, use the DROP TABLE command.

To a drop all projections:

```
=> DROP PROJECTION prejoin_p;
```

To drop the projection on node 2:

```
=> DROP PROJECTION prejoin_p_site02;
```

Alternatively, you can issue a command like the following, which drops projections on a particular schema:

```
=> DROP PROJECTION schemal.fact_proj_a, schemal.fact_proj_b;
```

If you want to drop a set of buddy projections, you could be prevented from dropping them individually using a sequence of DROP PROJECTION statements due to K-safety violations. See **MARK\_DESIGN\_KSAFE** (page 436) for details.

### See Also

**CREATE PROJECTION** (page 640), **DROP TABLE** (page 709), **GET\_PROJECTIONS** (page 429), **GET\_PROJECTION\_STATUS** (page 428), and **MARK\_DESIGN\_KSAFE** (page 436)

## DROP RESOURCE POOL

Drops a user-created resource pool. All memory allocated to the pool is returned back to the `GENERAL pool` (page 652).

Any requests queued against the pool are transferred to the `GENERAL pool` according to the priority of the pool compared to the `GENERAL pool`. If the pool's priority is higher than the `GENERAL pool`, the requests are placed at the head of the queue; otherwise the requests are placed at the end of the queue.

Any users who are using the pool are switched to use the `GENERAL pool` with a `NOTICE`:

`NOTICE:` Switched the following users to the General pool: username

`DROP RESOURCE POOL` returns an error if a user using the pool doesn't have permission to use the `GENERAL pool`. Existing sessions are transferred to the `GENERAL pool` regardless of whether the session's user has permission to use the `GENERAL pool`. This can result in additional user privileges if the pool being dropped is more restrictive than the `GENERAL pool`. To prevent giving users additional privileges, follow this procedure to drop restrictive pools:

- 1 **Revoke the permissions on the pool** (page 736) for all users.
- 2 Close any sessions that had permissions on the pool.
- 3 Drop the resource pool.

### Syntax

```
DROP RESOURCE POOL pool-name
```

### Parameters

<i>pool-name</i>	Specifies the name of the resource pool to be dropped.
------------------	--

### Example

The following command drops the resource pool that was created for the CEO:

```
=> DROP RESOURCE POOL ceo_pool;
```

### See Also

**`ALTER RESOURCE POOL`** (page 582)

**`CREATE RESOURCE POOL`** (page 649)

Managing Workloads in the Administrator's Guide

## DROP ROLE

Remove a role from the database.

## Syntax

```
DROP ROLE role [CASCADE];
```

## Parameters

<i>role</i>	The name of the role to drop
CASCADE	Revoke the role from users and other roles before dropping the role

## Notes

- Only the database superuser can drop a role.
- DROP ROLE fails if the role is assigned to one or more users or roles, if you do not supply the CASCADE option.

## Example

```
=> DROP ROLE appadmin;  
NOTICE: User bob depends on Role appadmin  
ROLLBACK: DROP ROLE failed due to dependencies  
DETAIL: Cannot drop Role appadmin because other objects depend on it  
HINT: Use DROP ROLE ... CASCADE to remove granted roles from the dependent  
users/roles  
=> DROP ROLE appadmin CASCADE;  
DROP ROLE
```

## DROP SCHEMA

Removes a schema from the database permanently. Be sure that you want to remove the schema and all its objects before you drop it because DROP SCHEMA is an irreversible process.

## Syntax

```
DROP SCHEMA name [, ...] [ CASCADE | RESTRICT ]
```

## Parameters

<i>name</i>	Specifies the name of the schema to drop.
CASCADE	Drops the schema even if it contains one or more objects.
RESTRICT	Drops the schema only if it does not contain any objects (the default).

## Restrictions

- By default, a schema cannot be dropped if it contains one or more objects. To force a drop, use the CASCADE statement.
- The PUBLIC schema cannot be dropped.
- A schema can only be dropped by its owner or a superuser.

## Notes

- A schema owner can drop a schema even if the owner does not own all the objects within the schema. All the objects within the schema is also dropped.
- If a user is accessing any object within a schema that is in the process of being dropped, the schema is not deleted until the transaction completes.
- Canceling a DROP SCHEMA statement can cause unpredictable results.

## Examples

The following example drops schema S1 only if it doesn't contain any objects:

```
=> DROP SCHEMA S1;
```

The following example drops schema S1 whether or not it contains objects:

```
=> DROP SCHEMA S1 CASCADE;
```

## DROP TRANSFORM FUNCTION

Drops a User Defined Transform Function (UDTF) from the Vertica catalog.

### Syntax

```
DROP TRANSFORM FUNCTION [ schema-name. ] name [, ...]
... ( [ [ argname ] argtype [, ...] ] )
```

### Parameters

<i>[schema-name.]</i>	[Optional] Specifies the name of a schema. When using more than one schema, specify the schema that contains the function to drop.
<i>name</i>	Specifies a name for the SQL Macro (function) to drop.
<i>argname</i>	Specifies the name of the argument, typically a column name.
<i>argtype</i>	Specifies the data type for argument(s) that are passed to the function. Argument types must match Vertica type names. See <b>SQL Data Types</b> (page 64).

### Notes

To drop a transform function, you must specify the argument types because there could be several functions that share the same name with different parameters.

### Permissions

Only the superuser or owner can drop the function.

### Example

The following command drops the `tokenize` UDTF in the `macros` schema:

```
=> DROP TRANSFORM FUNCTION macros.tokenize(varchar);
```

DROP TRANSFORM FUNCTION

**See Also****CREATE TRANSFORM FUNCTION** (page 693)

Using SQL Macros in the Programmer's Guide

## DROP SEQUENCE

Removes the specified sequence number generator.

**Syntax**DROP SEQUENCE [*schema-name.*]*name* [ , ... ]**Parameters**

[ <i>schema-name.</i> ]	[Optional] Specifies the name of a schema. When using more than one schema, specify the schema that contains the sequence to drop.
<i>name</i>	Specifies the name of the sequence to drop.

**Notes**

- A sequence can only be dropped by its owner or by a superuser.
- For sequences mentioned in a table's default expression, the default expression fails the next time you try to load data. Vertica does not check for these instances.
- The CASCADE keyword is not supported. Sequences used in a default expression of a column cannot be dropped until all references to the sequence are removed from the default expression.

**Example**The following command drops the sequence named `sequential`.

```
=> DROP SEQUENCE sequential;
```

**See Also****ALTER SEQUENCE** (page 586)**CREATE SEQUENCE** (page 658)**CURRVAL** (page 310)**GRANT (Sequence)** (page 725)**NEXTVAL** (page 308)

Using Sequences and Sequence Privileges in the Administrator's Guide

## DROP TABLE

Removes a table and, optionally, its associated projections.

### Syntax

```
DROP TABLE [ IF EXISTS ] [ schema-name. ] table [, ...] [ CASCADE ]
```

### Parameters

[ IF EXISTS ]	If specified, DROP TABLE does not report an error if one or of the tables to be dropped doesn't exist. This clause is useful in SQL scripts where you want to drop a table if it exists before recreating it.
[ <i>schema-name.</i> ]	[Optional] Specifies the name of a schema. When using more than one schema, specify the schema that contains the table to drop.
<i>table</i>	Specifies the name of a schema table. When using more than one schema, specify the schema that contains the table in the DROP TABLE statement.
CASCADE	[Optional] Drops all projections that include the table.

If you try to drop a table that has associated projections, a message listing the projections displays. For example:

```
=> DROP TABLE d1;
NOTICE: Constraint - depends on Table d1
NOTICE: Projection d1p1 depends on Table d1
NOTICE: Projection d1p2 depends on Table d1
NOTICE: Projection d1p3 depends on Table d1
NOTICE: Projection f1d1p1 depends on Table d1
NOTICE: Projection f1d1p2 depends on Table d1
NOTICE: Projection f1d1p3 depends on Table d1
ERROR: DROP failed due to dependencies: Cannot drop Table d1 because other objects
depend on it
HINT: Use DROP ... CASCADE to drop the dependent objects too.
=> DROP TABLE d1 CASCADE;
DROP TABLE
=> CREATE TABLE mytable (a INT, b VARCHAR(256));
CREATE TABLE
=> DROP TABLE IF EXISTS mytable;
DROP TABLE
=> DROP TABLE IF EXISTS mytable; -- Doesn't exist
NOTICE: Nothing was dropped
DROP TABLE
```

### Notes

- The table owner, schema owner, or superuser can drop a table.  
**Note:** The schema owner can drop a table but cannot truncate a table.
- Canceling a DROP TABLE statement can cause unpredictable results.
- Make sure that all other users have disconnected before using DROP TABLE.

- Views that reference a table that is dropped and then replaced by another table with the same name continue to function and use the contents of the new table, as long as the new table contains the same columns and column names.
- Use the multiple projection syntax in K-safe clusters.

**See Also**

***DELETE*** (page 699)

***DROP PROJECTION*** (page 704)

***TRUNCATE TABLE*** (page 787)

Adding Nodes and Deleting Data in the Administrator's Guide

## DROP USER

Removes a name from the list of authorized database users.

### Syntax

```
DROP USER name [, ...] [ CASCADE ]
```

### Parameters

<i>name</i>	Specifies the name or names of the user to drop.
CASCADE	[Optional] Drops all user-defined objects created by the user dropped, including schema, table and all views that reference the table, and the table's associated projections.

### Examples

DROP USER <name> fails if objects exist that were created by the user, such as schemas, tables and their associated projections:

```
=> DROP USER user1;
NOTICE: Table T_tbd1 depends on User user1
ROLLBACK: DROP failed due to dependencies
DETAIL: Cannot drop User user1 because other objects depend on it
HINT: Use DROP ... CASCADE to drop the dependent objects too
```

DROP USER <name> CASCADE succeeds regardless of any pre-existing user-defined objects. The statement forcibly drops all user-defined objects, such as schemas, tables and their associated projections:

```
=> DROP USER user1 CASCADE;
```

**Caution:** Tables owned by the user being dropped cannot be recovered after you issue DROP USER CASCADE.

DROP USER <username> succeeds if no user-defined objects exist (no schemas, tables or projections defined by the user):

```
=> CREATE USER user2;
=> DROP USER user2;
```

## DROP VIEW

Removes the specified view.

### Syntax

```
DROP VIEW name [ , ... ]
```

## Parameters

<i>name</i>	Specifies the name of the view to drop.
-------------	---

## Notes

- Only the specified view is dropped. Vertica does not support cascade functionality for views and it does not check for dependencies. Dropping a view causes any view that references it to fail.
- Views that reference a view or table that is dropped and then replaced by another view or table with the same name continue to function using the contents of the new view or table if it contains the same column names. If the column data type changes, the server coerces the old data type to the new one, if possible. Otherwise, it returns an error.

## Restrictions

To drop a view, the user must be either a superuser or the person who created the view.

## Examples

```
=> DROP VIEW myview;
```

## EXPLAIN

Outputs the query plan.

### Syntax

```
EXPLAIN { SELECT... | INSERT... | UPDATE... }
```

### Notes

- The EXPLAIN command is provided as a support feature and is not fully described here. For information on how to interpret the output, contact *Technical Support* (on page 1).
- EXPLAIN plans are annotated with a statistics status; for example, when the histograms are out of date or not available at all. See *Reacting to Stale Statistics in the Administrator's Guide*.
- The PATH ID annotation correlates EXPLAIN plans with profiling counters and join error messages. See *EXECUTION\_ENGINE\_PROFILES* (page 849).

### Textual Output

A compact human-readable representation of the query plan, laid out hierarchically. For example:

```
-----
QUERY PLAN DESCRIPTION:
-----

EXPLAIN SELECT * FROM hTicks h FULL OUTER JOIN aTicks a ON (h.time = a.time);
Access Path:
+-JOIN HASH [FullOuter] [Cost: 31, Rows: 4 (NO STATISTICS)] (PATH ID: 1)
|  Join Cond: (h."time" = a."time")
|  Execute on: Query Initiator
| +-- Outer -> STORAGE ACCESS for h [Cost: 15, Rows: 4 (NO STATISTICS)] (PATH ID:
2)
| |      Projection: public.HTicks_node0001
| |      Materialize: h.stock, h."time", h.price
| |      Execute on: Query Initiator
| +-- Inner -> STORAGE ACCESS for a [Cost: 15, Rows: 4 (NO STATISTICS)] (PATH ID:
3)
| |      Projection: public.ATicks_node0001
| |      Materialize: a.stock, a."time", a.price
| |      Execute on: Query Initiator
```

### Graphical Output

A GraphViz format of the graph for display in a graphical format. Graphviz is a graph plotting utility with layout algorithms, etc. You can obtain a Fedora Core 4 RPM for GraphViz from:

```
yum -y install graphviz
```

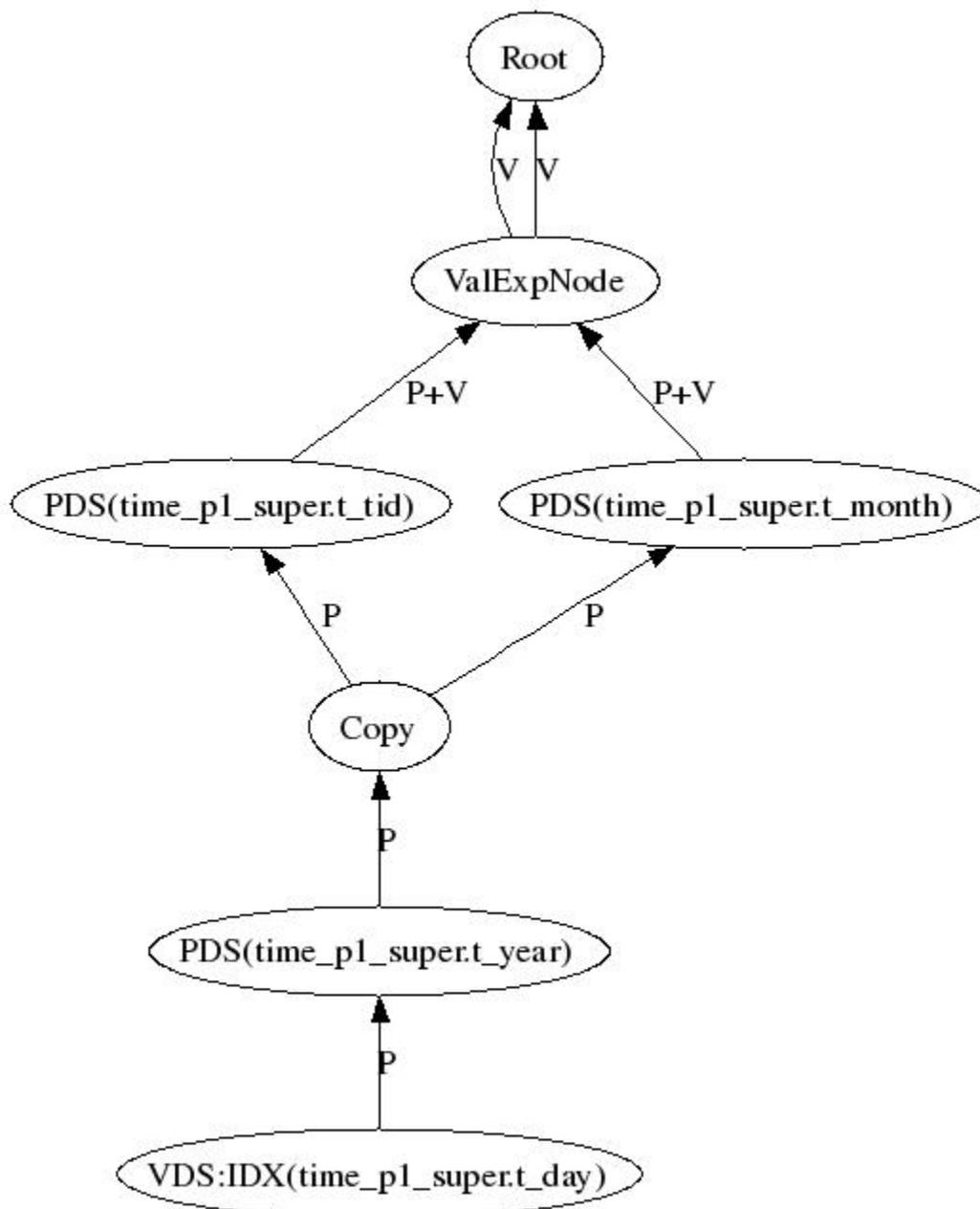
An example of a GraphViz graph for a Vertica plan:

```
digraph G {
graph [rankdir=BT]
0[label="Root"];
```

```
1[label="ValExpNode"];
2[label="VDS:DVIDX(P0.c_state)"];
3[label="PDS(P0.c_gender)"];
4[label="PDS(P0.c_name)"];
5[label="Copy"];
6[label="PDS(P0.c_cid)"];
7[label="PDS(P0.c_state)"];
1->0 [label="V"];
1->0 [label="V"];
2->3 [label="P"];
3->4 [label="P"];
4->5 [label="P"];
5->6 [label="P"];
5->7 [label="P"];
6->1 [label="P+V"];
7->1 [label="P+V"]; }
```

To create a picture of the plan, copy the output above to a file, in this example /tmp/x.txt:

- `dot -Tps /tmp/x.txt > /tmp/x.ps`
- `ggv x.ps` [evince x.ps works if you don't have ggv]
- Alternative: `dot -Tps | ghostview` - and paste in the digraph.
- Alternative: generate jpg using `-Tjpg`.
- To scale an image for printing (8.5"x11" in this example):
- Portrait: `dot -Tps -Gsize="7.5,10" -Gmargin="0.5" ...`
- Landscape: `dot -Tps -Gsize="10,7.5" -Gmargin="0.5" -Grotate="90" ...`

**Example**

**Note:** For GraphViz information, see <http://www.graphviz.org/Documentation.php> (<http://www.graphviz.org/Documentation.php>)

**Correlating EXPLAIN plan output with error messages**

EXPLAIN plans are correlated with join error messages and EE profiling counters. These mechanisms enhance your database's usability by letting you quickly troubleshoot issues, such as providing insight into what part of your plan failed.

Each operation in a plan contains a numeric ID; for example, (PATH ID: 1).

The following sample command returns the EXPLAIN plan's textual output with a PATH ID for each operation:

```
=> EXPLAIN SELECT * FROM fact JOIN dim ON x=y JOIN ext on y=x;

Access Path:
+-JOIN HASH [Cost: 41K, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
 | Join Filter: (dim.y = fact.x)
 | Materialize at Input: fact.x
 | +-- Outer -> JOIN MERGEJOIN(inputs presorted) [Cost: 1K, Rows: 10K (NO
STATISTICS)] (PATH ID: 2)
 | | Join Cond: (fact.x = dim.y)
 | | +-- Outer -> STORAGE ACCESS for fact [Cost: 604, Rows: 10K (NO STATISTICS)]
(PATH ID: 3)
 | | | Projection: public.fact_super
 | | | Materialize: fact.x
 | | +-- Inner -> STORAGE ACCESS for dim [Cost: 604, Rows: 10K (NO STATISTICS)]
(PATH ID: 4)
 | | | Projection: public.dim_super
 | | | Materialize: dim.y
 | +-- Inner -> STORAGE ACCESS for ext [Cost: 604, Rows: 10K (NO STATISTICS)] (PATH
ID: 6)
 | | Projection: public.ext_super
 | | Materialize: ext.z
```

Any error messages that refer to joins also include the PATH ID number:

```
ERROR: Join inner did not fit in memory ((B x A)
using B_sp and A_sp (PATH ID: 1))
```

The PATH ID is also available via the EXECUTION\_ENGINE\_PROFILES profiling counters, letting you correlate EE operations with the corresponding portion of the EXPLAIN plan's textual output.

For example, the following series of sample commands occur in three parts:

- 1 Run an EXPLAIN command
- 2 Run a PROFILE command
- 3 Query the EXECUTION\_ENGINE\_PROFILES system table to show the counters with path\_id

The following command outputs the query plan description:

### Part 1: run EXPLAIN

```
=> EXPLAIN SELECT * FROM fact JOIN dim ON x=y JOIN ext on y=z;

Access Path:
+-JOIN MERGEJOIN(inputs presorted) [Cost: 815, Rows: 10K (NO STATISTICS)] (PATH
ID: 1)
 | Join Cond: (dim.y = ext.z)
 | Materialize at Output: fact.x
 | Execute on: All Nodes
 | +-- Outer -> JOIN MERGEJOIN(inputs presorted) [Cost: 408, Rows: 10K (NO
STATISTICS)] (PATH ID: 2)
 | | Join Cond: (fact.x = dim.y)
 | | Execute on: All Nodes
```

```

| | +-- Outer -> STORAGE ACCESS for fact [Cost: 202, Rows: 10K (NO STATISTICS)]
(PATH ID: 3)
| | | Projection: public.fact_super
| | | Materialize: fact.x
| | | Execute on: All Nodes
| | +-- Inner -> STORAGE ACCESS for dim [Cost: 202, Rows: 10K (NO STATISTICS)]
(PATH ID: 4)
| | | Projection: public.dim_super
| | | Materialize: dim.y
| | | Execute on: All Nodes
| +-- Inner -> STORAGE ACCESS for ext [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH
ID: 5)
| | Projection: public.ext_super
| | Materialize: ext.z
| | Execute on: All Nodes

```

## Part 2: run PROFILE

Now profile the query:

```
=> PROFILE select * from fact JOIN dim ON x=y JOIN ext on y=z;
```

## Part 3: query the system table

Run the EXECUTION\_ENGINE\_PROFILES system table to show the counters with path\_id:

```
=> SELECT node_name, operator_name, counter_name, path_id
FROM execution_engine_profiles
WHERE operator_name LIKE 'Join%' AND counter_name LIKE '%rows%';
```

node_name	operator_name	counter_name	path_id
e0	JoinManyFewMerge	rows produced	1
e0	JoinManyFewMerge	rle rows produced	1
e0	JoinManyFewMerge	estimated rows produced	1
e0	JoinManyFewMerge	rows produced	2
e0	JoinManyFewMerge	rle rows produced	2
e0	JoinManyFewMerge	estimated rows produced	2
e1	JoinManyFewMerge	rows produced	1
e1	JoinManyFewMerge	rle rows produced	1
e1	JoinManyFewMerge	estimated rows produced	1
e1	JoinManyFewMerge	rows produced	2
e1	JoinManyFewMerge	rle rows produced	2
e1	JoinManyFewMerge	estimated rows produced	2
initiator	JoinManyFewMerge	rows produced	1
initiator	JoinManyFewMerge	rle rows produced	1
initiator	JoinManyFewMerge	estimated rows produced	1
initiator	JoinManyFewMerge	rows produced	2
initiator	JoinManyFewMerge	rle rows produced	2
initiator	JoinManyFewMerge	estimated rows produced	2

(18 rows)

## See Also

**EXECUTION\_ENGINE\_PROFILES** (page 849)

## EXPORT

Exports an entire table, columns from a table, or the results of a **SELECT** (page 747) statement to another Vertica database. Before you can export data to another database, you need connect to it using the **CONNECT** (page 605) statement. Exported data is written into the target database using AUTO mode.

### Syntax

```
EXPORT TO database.[dest-schema.]dest-table
... [(dest-column [,dest-column2,...])]
... { AS SELECT select-expression
... | FROM [source-schema.]source-table
... [(source-column [,source-column2,...])]};
```

### Parameters

<i>database</i>	A string containing the name of the database to receive the exported data. There must be an active connection to this database for the export to succeed.
<i>.[dest-schema.]</i> <i>dest-table</i>	The table to store the exported data (schema specification is optional). This table must already exist.
<i>dest-column</i> [, <i>dest-column2</i> ,...]	A list of columns in the target table to store the exported data.
AS SELECT <i>select-expression</i>	A standard SELECT expression that selects the data to be exported. See <b>SELECT</b> (page 747) for the syntax.
FROM [ <i>source-schema</i> .] <i>source-table</i>	The table that contains the data to be exported (schema optional)
<i>source-column</i> [, <i>source-column2</i> ,...]	A list of the columns in the source table to export. If present, only these columns are exported.

### Notes

- The user must have **SELECT** (page 747) privileges on the source table, and the user account used to connect to the target database must have insert privileges for the destination table.
- Auto-projections for the target table are similar to the projections for the source table.
- If you do not supply a list of source and destination columns, EXPORT attempts to match columns in the source table with corresponding columns in the destination table. See the following section or details.

## Source and Destination Column Mapping

The EXPORT statement needs to map columns in the source table to columns in the destination table. You can optionally supply lists of either source columns to be copied, columns in the destination table where data should be stored, or both. These lists let you choose a subset of the columns in the source table to be copied to the destination table. Since neither of these lists are required, there are a few different behaviors depending on which list (if any) are present, as explained in the following table.

	No Source Column List	Source Column List Supplied
No Destination Column List	All columns in the source table are matched to columns in the destination table. The number of columns in the two tables do not have to match, but the destination table must have at least as many columns as the source.	Only the content from the columns in the source list are exported. Columns in the destination table are matched to columns in the source list. The destination table must have at least the number of columns in the source list. It can have more columns.
Destination Column List Supplied	The columns in the destination column list are matched to columns in the source. The number of columns in the destination list must match the number of columns in the source table.	Columns are matched in the to column lists. The lists must have the same number of columns.

### Examples

First, open the connection to the other database, then perform a simple export of an entire table to an identical table in the target database.

```
=> CONNECT TO VERTICA testdb USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> EXPORT TO VERTICA testdb.customer_dimension FROM customer_dimension;
Rows Exported
-----
          23416
(1 row)
```

The following statement demonstrates exporting a portion of a table using a simple **SELECT** (page 747) statement.

```
=> EXPORT TO VERTICA testdb.ma_customers AS SELECT customer_key, customer_name,
annual_income
-> FROM customer_dimension WHERE customer_state = 'MA';
Rows Exported
-----
          3429
(1 row)
```

This statement exports several columns from one table to several different columns in the target database table using column lists. Remember that when supplying both a source and destination column list, the number of columns must match.

```
=> EXPORT TO VERTICA testdb.people (name, gender, age) FROM customer_dimension
-> (customer_name, customer_gender, customer_age);
```

```
Rows Exported
-----
                23416
(1 row)
```

## GRANT (Database)

Grants the right to create schemas within the database to a user or role.

### Syntax

```
GRANT {
... { CREATE [, ...]
... | { TEMPORARY | TEMP }
... | ALL [ PRIVILEGES ]
... | CONNECT } }
... ON DATABASE database-name [, ...]
... TO { username | rolename } [, ...]
... [ WITH GRANT OPTION ]
```

### Parameters

CREATE	Allows the user to create schemas within the specified database.
TEMPORARY   TEMP	Allows the user to create temp tables in the database. <b>Note:</b> This privilege is provided by default with <b>CREATE USER</b> (page 695).
CONNECT	Allows the user to connect to a database.
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
<i>database-name</i>	Identifies the database in which to grant the privilege.
<i>username</i>   <i>rolename</i>	Grants the privilege to the specified user or role.
WITH GRANT OPTION	Allows the recipient of the privilege to grant it to other users.

### Notes

By default, only the superuser has the right to create a database schema.

### Example

The following example grants Fred the right to create schemas on vmartdb.

```
=> GRANT CREATE ON DATABASE vmartdb TO Fred;
```

## GRANT (Function)

Grants the `EXECUTE` privilege on a SQL Macro or User Defined Function to a database user or role.

### Syntax

```
GRANT EXECUTE
... ON FUNCTION [schema-name.]function-name [, ...]
... ( [ argname ] argtype [ ,... ] )
... TO { username | role | PUBLIC } [ , ... ]
```

### Parameters

<code>[<i>schema-name</i>.]<i>function-name</i></code>	Specifies the SQL Macro on which to grant the <code>EXECUTE</code> privilege. When using more than one schema, specify the schema that contains the function.
<code><i>argname</i></code>	Specifies the argument name(s).
<code><i>argtype</i></code>	Specifies the argument data types(s).
<code><i>username</i></code>	Grants the privilege to the specified user.
<code><i>role</i></code>	Grants the privilege to the specified role.
<code>PUBLIC</code>	Grants the privilege to all users.

### Permissions

- Only the superuser and owner can grant `EXECUTE` privilege on a SQL Macro.
- Additionally, users must have `USAGE` privileges on the schema that contains the function. See **GRANT (Schema)** (page 725).

### Example

The following command grants `EXECUTE` privileges to user Fred on the `zeroifnull` function:

```
=> GRANT EXECUTE ON FUNCTION zeroifnull (x INT) TO Fred;
```

### See Also

**REVOKE (Function)** (page 735)

## GRANT (Procedure)

Grants the execute privilege on a procedure to a database user or role.

### Syntax

```
GRANT EXECUTE
... ON PROCEDURE [schema-name.]procedure-name [, ...]
... ( [ argname ] argtype [ ,... ] )
... TO { username | role | PUBLIC } [ , ... ]
```

## Parameters

<code>[<i>schema-name</i>.]<i>procedure-name</i></code>	Specifies the procedure on which to grant the execute privilege. When using more than one schema, specify the schema that contains the procedure.
<code><i>argname</i></code>	Specifies the argument name or names used when creating the procedure.
<code><i>argtype</i></code>	Specifies the argument types used when creating the procedure.
<code><i>username</i></code>	Grants the privilege to the specified user.
<code><i>role</i></code>	Grants the privilege to the specified role.
<code>PUBLIC</code>	Grants the privilege to all users.

## Notes

- Only the superuser can grant `USAGE` on a procedure.
- Additionally, users must have privileges on the schema that contains the procedure.

## See Also

***REVOKE (procedure)*** (page 736)

## GRANT (Resource Pool)

Grants access privilege for a resource pool to a database user or role.

## Syntax

```
GRANT USAGE
... ON RESOURCE POOL resource-pool
... TO { username | role | PUBLIC } [ , ... ]
```

## Parameters

<code><i>resource-pool</i></code>	Specifies the resource pool on which to grant the usage privilege.
<code><i>username</i></code>	Grants the privilege to the specified user.
<code><i>role</i></code>	Grants the privilege to the specified role.
<code>PUBLIC</code>	Grants the privilege to all users.

## Notes

Once granted usage rights, users can switch to using the resource pool using ***ALTER USER*** (page 594) (by passing their own username) or ***SET SESSION RESOURCE POOL*** (page 778).

## See Also

***REVOKE (Resource Pool)*** (page 736)

## GRANT (Role)

Adds a role to users or other roles. When a new role is granted to users, they can enable the role for their session, giving them the permissions granted by the role. When a role is granted to another role, the newly granted role's permissions are combined with the role's existing permissions.

### Syntax

```
GRANT role [,...]
... TO { user | role | PUBLIC } [, ...]
... [WITH ADMIN OPTION];
```

### Parameters

<code><i>role</i> [,...]</code>	The name of one or more roles to be granted to users or roles
<code><i>user</i>   <i>role</i>   PUBLIC</code>	The name of a user or other role to be granted the role. If the keyword PUBLIC is supplied, then all users have access to the role.
<code>WITH ADMIN OPTION</code>	Grants users and roles administrative privileges for the role. They are able to grant the role to and revoke the role from other users or roles.

### Notes

- No special consideration is given to users who are granted roles by the database superuser. For example, if two users are granted a role with administrative privileges, one user can revoke the role for the other user.
- Grants of one role to another are checked for circular references. If one is detected, the grant fails with an error.

### Examples

Create some roles and assign one to a user.

```
=> CREATE ROLE appdata;
CREATE ROLE
=> CREATE ROLE applogs;
CREATE ROLE
=> CREATE ROLE appadmin;
CREATE ROLE
=> GRANT appdata TO bob;
GRANT ROLE
```

Grant two roles to another role:

```
=> GRANT appdata, applogs TO appadmin; -- grant to other roles
GRANT ROLE
```

Now, any privileges assigned to either appdata or applogs are automatically assigned to appadmin as well.

When assigning roles to other roles, be sure you do not create circular references.

```
=> GRANT appadmin TO appdata;  
WARNING: Circular assignation of roles is not allowed  
HINT: Cannot grant appadmin to appdata  
GRANT ROLE
```

Users with administrative privileges can revoke roles for other users.

```
=> GRANT appadmin TO bob, alice WITH ADMIN OPTION;  
GRANT ROLE  
=> \connect - bob  
You are now connected as user "bob".  
=> REVOKE appadmin FROM alice;  
REVOKE ROLE
```

## GRANT (Schema)

Grants privileges on a schema to a database user or role.

### Syntax

```
GRANT {
... { CREATE | USAGE } [ , ... ]
... | ALL [ PRIVILEGES ] }
... ON SCHEMA schemaname [ , ... ]
... TO { username | role | PUBLIC } [ , ... ]
... [ WITH GRANT OPTION ]
```

### Parameters

CREATE	Allows the user read access to the schema and the right to create tables and views within the schema.
USAGE	Allows the user access to the objects contained within the schema. This allows the user to look up objects within the schema. Note that the user must also be granted access to the individual objects. See the <b>GRANT TABLE</b> (page 727) and <b>GRANT VIEW</b> (page 728) statements.
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
<i>schemaname</i>	Is the name of the schema for which privileges are being granted.
<i>username</i>	Grants the privilege to a specific user.
<i>role</i>	Grants the privilege to a specific role.
PUBLIC	Grants the privilege to all users.
WITH GRANT OPTION	Allows the recipient of the privilege to grant it to other users.

### Notes

Newly-created users do not have access to schema PUBLIC by default. Make sure to grant USAGE on schema PUBLIC to all users you create.

## GRANT (Sequence)

Grants privileges on a sequence generator to a user or role.

### Syntax

```
GRANT {
... { USAGE | SELECT | UPDATE }
... | ALL [ PRIVILEGES ] }
... ON SEQUENCE [schema-name.]sequence_name [ , ... ]
... TO { username | role | PUBLIC } [ , ... ]
... [ WITH GRANT OPTION ]
```

**Parameters**

USAGE	Allows the user to use both the CURRVAL and NEXTVAL functions on the specified sequence.
SELECT	Allows the user to use the CURRVAL function on the specified sequence.
UPDATE	Allows the user to use the NEXTVAL function on the specified sequence.
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
[ <i>schema-name</i> .] <i>sequence_name</i>	Specifies the sequence on which to grant the privileges. When using more than one schema, specify the schema that contains the sequence on which to grant privileges.
<i>username</i>	Grants the privilege to the specified user.
<i>role</i>	Grants the privilege to the specified role.
PUBLIC	Grants the privilege to all users.
WITH GRANT OPTION	Allows the user to grant the same privileges to other users.

**Notes**

The user must also be granted USAGE on the schema that contains the sequence. See **GRANT (Schema)** (page 725).

## GRANT (Table)

Grants privileges on a table to a user or role.

### Syntax

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | REFERENCES } [ , ... ]
... | ALL [ PRIVILEGES ] }
... ON [ TABLE ] [ schema-name. ] tablename [ , ... ]
... TO { username | role | PUBLIC } [ , ... ]
... [ WITH GRANT OPTION ]
```

### Parameters

SELECT	Allows the user to SELECT from any column of the specified table.
INSERT	Allows the user to INSERT tuples into the specified table and to use the <b>COPY</b> (page 607) command to load the table. <b>Note:</b> COPY FROM STDIN is allowed to any user granted the INSERT privilege, while COPY FROM <file> is an admin-only operation.
UPDATE	Allows the user to UPDATE tuples in the specified table.
DELETE	Allows DELETE of a row from the specified table.
REFERENCES	Is necessary to have this privilege on both the referencing and referenced tables in order to create a foreign key constraint.
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
[ schema-name. ] tablename	Specifies the table on which to grant the privileges. When using more than one schema, specify the schema that contains the table on which to grant privileges.
username	Grants the privilege to the specified user.
role	Grants the privilege to the specified role.
PUBLIC	Grants the privilege to all users.
WITH GRANT OPTION	Allows the user to grant the same privileges to other users.

### Notes

- The user must also be granted USAGE on the schema that contains the table. See **GRANT (Schema)** (page 725).
- To use the **DELETE** (page 699) or **UPDATE** (page 792) commands with a **WHERE clause** (page 752), a user must have both SELECT and UPDATE and DELETE privileges on the table.
- The user can be granted privileges on a global temporary table, but not a local temporary table.

## GRANT (Transform Function)

Grants `EXECUTE` privileges on a User Defined Transform Function to a database user or role.

### Syntax

```
GRANT EXECUTE
... ON TRANSFORM FUNCTION [schema-name.]function-name [, ...]
... ( [ argname ] argtype [ ,... ] )
... TO { username | role | PUBLIC } [ , ... ]
```

### Parameters

<code>[<i>schema-name</i>.]<i>function-name</i></code>	Specifies the SQL Macro on which to grant the <code>EXECUTE</code> privilege. When using more than one schema, specify the schema that contains the function.
<code><i>argname</i></code>	Specifies the argument name(s).
<code><i>argtype</i></code>	Specifies the argument data types(s).
<code><i>username</i></code>	Grants the privilege to the specified user.
<code><i>role</i></code>	Grants the privilege to the specified role.
<code>PUBLIC</code>	Grants the privilege to all users.

### Permissions

- Only the superuser and owner can grant `EXECUTE` privilege on a transform function.
- Additionally, users must have `USAGE` privileges on the schema that contains the function. See **GRANT (Schema)** (page 725).

### Example

The following command grants `EXECUTE` privileges to user Bob on the `tokenize` function:

```
=> GRANT EXECUTE ON TRANSFORM FUNCTION tokenize(varchar) TO Bob;
```

## GRANT (View)

Grants privileges on a view to a database user or role.

### Syntax

```
GRANT {
... { SELECT }
... | ALL [ PRIVILEGES ] }
... ON [schema-name.]viewname [, ...]
... TO { username | role | PUBLIC } [, ...]
... [ WITH GRANT OPTION ]
```

## Parameters

SELECT	Allows the user to perform SELECT operations on a view and the resources referenced within it.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
ALL	Applies to all privileges.
<i>[schema-name.]viewname</i>	Specifies the view on which to grant the privileges. When using more than one schema, specify the schema that contains the view.
<i>username</i>	Grants the privilege to the specified user.
<i>role</i>	Grants the privilege to the specified role.
PUBLIC	Grants the privilege to all users.
WITH GRANT OPTION	Allows the user to grant the same privileges to other users.

## Notes

If `userA` wants to grant `userB` access to a view, `userA` must specify `WITH GRANT OPTION` on the base table, in addition to the view, regardless of whether `userB` (grantee) has access to the base table.

## INSERT

Inserts values into all projections of a table. By default, Insert first uses the WOS and if the WOS is full then overflows to the ROS.

**Note:** If a table has no associated projections, Vertica creates a default superprojection for the table in which to insert the data.

### Syntax

```
INSERT [ /*+ direct */ ]
... INTO [schema-name.]table
... [ ( column [, ...] ) ]
... { DEFAULT VALUES
... | VALUES ( { expression | DEFAULT } [, ...] )
... | SELECT... (page 747) }
```

### Parameters

<code>/*+ direct */</code>	Writes the data directly to disk (ROS) bypassing memory (WOS). <b>Notes:</b> <ul style="list-style-type: none"> <li>Vertica accepts optional spaces before and after the plus sign (e.g., between the <code>/*</code> and the <code>+</code>).</li> <li>If you insert using the <code>direct</code> hint, you still need to issue a <code>COMMIT</code> or <code>ROLLBACK</code> command to finish the transaction.</li> </ul>
<code>[schema-name.]table</code>	Specifies the name of a table in the schema. You cannot INSERT tuples into a projection. When using more than one schema, specify the schema that contains the table.
<code>column</code>	Specifies a column of the table.
<code>DEFAULT VALUES</code>	Fills all columns with their default values as specified in <b>CREATE TABLE</b> (page 664).
<code>VALUES</code>	Specifies a list of values to store in the corresponding columns. If no value is supplied for a column, Vertica implicitly adds a <code>DEFAULT</code> value, if present. Otherwise Vertica inserts a <code>NULL</code> value or, if the column is defined as <code>NOT NULL</code> , returns an error.
<code>expression</code>	Specifies a value to store in the corresponding column.
<code>DEFAULT</code>	Stores the default value in the corresponding column.
<code>SELECT...</code>	Specifies a query ( <b>SELECT</b> (page 747) statement) that supplies the rows to be inserted.

### Notes

- An `INSERT ... SELECT` statement refers to tables in both its `INSERT` and `SELECT` clauses. Isolation level applies only to the `SELECT` clauses and work just like a normal query.

- You can list the target columns in any order. If no list of column names is given at all, the default is all the columns of the table in their declared order; or the first N column names, if there are only N columns supplied by the VALUES clause or query. The values supplied by the VALUES clause or query are associated with the explicit or implicit column list left-to-right.
- You must insert one complete tuple at a time.
- Do not use meta-functions in INSERT statements.

## Examples

```
=> INSERT INTO FACT VALUES (101, 102, 103, 104);
=> INSERT INTO CUSTOMER VALUES (10, 'male', 'DPR', 'MA', 35);
=> INSERT INTO Retail.T1 (C0, C1) VALUES (1, 1001);
=> INSERT INTO films
    SELECT * FROM tmp_films
    WHERE date_prod < '2004-05-07';
```

When doing an INSERT /\*+ direct \*/ Vertica takes optional spaces before and after the plus sign (e.g., between the /\* and the +). Both of the following commands, for example, load data into the ROS:

```
=> CREATE TABLE rostab(x TIMESTAMP);
=> INSERT /*+ direct */ INTO rostab VALUES ('2011-02-10 12:01:00');
=> INSERT /*+ direct */ INTO rostab VALUES ('2011-02-15 12:01:00');
=> SELECT wos_row_count, ros_row_count FROM column_storage
    WHERE anchor_table_name = 'rostab';
 wos_row_count | ros_row_count
-----+-----
              0 |              2
              0 |              2
(2 rows)
```

## LCOPY

Loads a data file from a client system into the database. This statement is nearly identical to the **COPY** (page 607) statement with a few exceptions:

- It loads data from a client system, rather than a cluster host. This means LCOPY does not support 'pathToData' ON nodename.
- It does not support the FORMAT parameter that COPY uses to specify the format of date/time and binary data types.
- Instead of REJECTED DATA, LCOPY has a REJECTEDFILE parameter that takes the path of a file on the client system where it should save a list of rejected data. The row numbers of rejected rows are written to this file rather than the full content.
- The LCOPY command is only available via the ODBC interface. You cannot use it interactively from vsql.

**Note:** LCOPY cannot handle files with filenames longer than 252 characters. You need to rename any files that have filenames longer than this limit.

LCOPY converts the end of line sequence of the client platform into the standard end of line sequence used in Vertica. On Windows platforms, the end of line sequence is `\r\n` (carriage return character followed by a newline character) while on Linux platforms, the end of line sequence is just `\n` (newline). The Vertica client driver converts these end of line sequences as it loads the data.

For example, loading the following text file using LCOPY (with `|` set as the record terminator, and the escaped characters `\r` and `\n` replaced by actual control characters) is interpreted differently depending on the client system's platform:

```
1|2\r\n|3|
```

On a Windows platform, the `\r\n` is recognized as the end of line sequence, so the Vertica client driver translates it into the standard end of line sequence (`\n`). The result is a single-column table with the entries:

a
1
2\n
3

The `\r` is missing, since the `\r\n` sequence was translated to a single `\n`.

Using LCOPY from a Linux platform, only the `\n` is recognized as an end of line character, so the resulting table is:

a
1
2\r\n
3

To avoid confusion, you should ensure that the text files you load using LCOPY use the native line end sequence for your client's operating system.

## Example

The following code loads the table TEST from the file `C:\load.dat` located on a system where the code is run.

```
ODBCConnection<ODBCDriverConnect> test("VerticaSQL");
test.connect();
char *sql = "LCOPY test FROM 'C:\load.dat' REJECTEDFILE 'c:\rejects.log' DELIMITER
'|';";
ODBCStatement stm(test.conn);
stm.execute(sql);
```

## PROFILE

Profiles a single SQL statement.

### Syntax

```
PROFILE { SELECT ... }
```

## Output

Writes a hint to `stderr`, as described in the example below.

## Notes

To profile a single statement add the `PROFILE` keyword to the beginning of the statement:

```
=> PROFILE SELECT customer_name, annual_income
      FROM public.customer_dimension
      WHERE (customer_gender, annual_income) IN (
          SELECT customer_gender, MAX(annual_income)
          FROM public.customer_dimension
          GROUP BY customer_gender);
```

`PROFILE < SELECT ...>` saves profiling information for future analysis.

A hint is written to `stderr` (the standard error stream and default destination for error messages and other diagnostic warnings, which are typically output to the screen) while the statement is executing:

```
NOTICE: Statement is being profiled.
HINT: select * from v_monitor.execution_engine_profiles where
transaction_id=45035996273740886 and statement_id=10;
NOTICE: Initiator memory estimate for query:
[on pool general: 1418047 KB, minimum: 192290 KB]
NOTICE: Total memory required by query: [1418047 KB]
  customer_name | annual_income
-----+-----
Meghan U. Miller |          999960
Michael T. Jackson |          999981
(2 rows)
```

**Tip:** Use the statement returned by the hint as a starting point for reviewing the query's profiling data.

To see what counters are available, issue the following command:

```
=> SELECT DISTINCT(counter_name) FROM EXECUTION_ENGINE_PROFILES;
```

## RELEASE SAVEPOINT

Destroys a savepoint without undoing the effects of commands executed after the savepoint was established.

### Syntax

```
RELEASE [ SAVEPOINT ] savepoint_name
```

### Parameters

<code>savepoint_name</code>	Specifies the name of the savepoint to destroy.
-----------------------------	---

## Notes

Once destroyed, the savepoint is unavailable as a rollback point.

## Example

The following example establishes and then destroys a savepoint called `my_savepoint`. The values 101 and 102 are both inserted at commit.

```
=> INSERT INTO product_key VALUES (101);
=> SAVEPOINT my_savepoint;
=> INSERT INTO product_key VALUES (102);
=> RELEASE SAVEPOINT my_savepoint;
=> COMMIT;
```

## See Also

**SAVEPOINT** (page 745) and **ROLLBACK TO SAVEPOINT** (page 744)

## REVOKE (Database)

Revokes the right for the specified user or role to create schemas in the specified database.

## Syntax

```
REVOKE [ GRANT OPTION FOR ]
... { CREATE | { TEMPORARY | TEMP } [ , ... ] }
... | CONNECT
... | ALL [ PRIVILEGES ] }
... ON DATABASE database-name [ , ... ]
... FROM { username | role } [ , ... ]
... [ CASCADE ]
```

## Parameters

GRANT OPTION FOR	Revokes the grant option for the privilege, not the privilege itself. If omitted, revokes both the privilege and the grant option.
CREATE	Revokes the right to create schemas in the specified database.
TEMPORARY   TEMP	Revokes the right to create temp tables in the database. <b>Note:</b> This privilege is provided by default with <b>CREATE USER</b> (page 695).
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
<i>database-name</i>	Identifies the database from which to revoke the privilege.
<i>username</i>	Identifies the user from whom to revoke the privilege.
<i>role</i>	Identifies the role from which to revoke the privilege.
CASCADE	Revokes the privilege from the specified user/role and all other users and roles that were granted the same privilege along a

	chain of recipients that can be traced back to the specified user.
--	--

**Example**

The following example revokes Fred's right to create schemas on vmartdb:

```
=> REVOKE CREATE ON DATABASE vmartdb FROM Fred;
```

The following revokes Fred's right to create temporary tables in vmartdb:

```
=> REVOKE TEMPORARY ON DATABASE vmartdb FROM Fred;
```

**See Also**

Granting and Revoking Privileges

**REVOKE (Function)**

Revokes the `EXECUTE` privilege on a SQL Macro or User Defined Transform Function (UDTF) from a database user or role.

**Syntax**

```
REVOKE EXECUTE
... ON FUNCTION [ schema-name. ] function-name [ , ... ]
... ( [ argname ] argtype [ , ... ] )
... FROM { username | PUBLIC | role } [ , ... ]
... [ CASCADE ]
```

**Parameters**

<i>[schema-name.]function-name</i>	Specifies the SQL Macro or UDTF from which to revoke the <code>EXECUTE</code> privilege. When using more than one schema, specify the schema that contains the function.
<i>argname</i>	Specifies the argument name or names.
<i>argtype</i>	Specifies the argument data type or types.
<i>username</i>	Revokes the privilege from the specified user.
PUBLIC	Revokes the privilege from all users.
<i>role</i>	Revokes the privileges from a role.
CASCADE	Revokes the privilege from the specified user/role and all other users and roles that were granted the same privilege along a chain of recipients that can be traced back to the specified user.

**Permissions**

Only the superuser and owner can revoke `EXECUTE` privilege on a SQL Macro or UDTF.

**Example**

The following command revokes `EXECUTE` privileges from user Fred on the `zeroifnull` function:

```
=> REVOKE EXECUTE ON FUNCTION zeroifnull (x INT) FROM Fred;
```

## See Also

**GRANT (Function)** (page 721)

Granting and Revoking Privileges

## REVOKE (Procedure)

Revokes the execute privilege on a procedure from a user or role.

### Syntax

```
REVOKE EXECUTE
... ON [schema-name.]procedure-name [ , ... ]
... ( [ argname ] argtype [ ,... ] )
... FROM { username | PUBLIC | role } [ , ... ]
...[ CASCADE ]
```

### Parameters

<i>[schema-name.]procedure-name</i>	Specifies the procedure on which to revoke the execute privilege. When using more than one schema, specify the schema that contains the procedure.
<i>argname</i>	Specifies the argument names used when creating the procedure.
<i>argtype</i>	Specifies the argtypes used when creating the procedure.
<i>username</i>	Specifies the user from whom to revoke the privilege.
PUBLIC	Revokes the privilege from all users.
<i>role</i>	Specifies the role from whom to revoke the privilege.
CASCADE	Revokes the privilege from the specified user/role and all other users and roles that were granted the same privilege along a chain of recipients that can be traced back to the specified user.

### Notes

Only the superuser can revoke USAGE on a procedure.

## See Also

**GRANT (Procedure)** (page 721)

## REVOKE (Resource Pool)

Revokes a user's or role's access privilege to a resource pool.

## Syntax

```
REVOKE USAGE
... ON RESOURCE POOL resource-pool
... FROM { username | PUBLIC | role } [ , ... ]
...[ CASCADE ]
```

## Parameters

<i>resource-pool</i>	Specifies the resource pool from which to revoke the usage privilege.
<i>username</i>	Revokes the privilege from the specified user.
PUBLIC	Revokes the privilege from all users.
<i>role</i>	Revokes the privilege from the specified role.
CASCADE	Revokes the privilege from the specified user/role and all other users and roles that were granted the same privilege along a chain of recipients that can be traced back to the specified user.

## Notes

- Vertica checks resource pool permissions when a user initially switches to the pool, rather than on each access. Revoking a user's permission to use a resource pool does not affect existing sessions. You need to close the user's open sessions that are accessing the resource pool if you want to prevent them from continuing to use the pool's resources.
- It is an error to revoke a user's access permissions for the resource pool to which they are assigned (their default pool). You must first change the pool they are assigned to using **ALTER USER ... RESOURCE POOL** (page 594) (potentially using **GRANT USAGE ON RESOURCE POOL** (page 722) first to allow them to access the new pool) before revoking their access.

## See Also

**GRANT (Resource Pool)** (page 722)

## REVOKE (Role)

Revokes a user's or another role's access to a role.

## Syntax

```
REVOKE [ ADMIN OPTION FOR ] role [ , ... ]
... FROM { user | role | PUBLIC } [ , ... ]
...[ CASCADE ];
```

## Parameters

ADMIN OPTION FOR	Revokes just the user's or role's administration access to the role.
<i>role</i>	The name of one or more roles you want to revoke access to.
<i>user</i>   <i>role</i>   PUBLIC	The name of a user or role whose permission you want to revoke. You can use the PUBLIC

	option to revoke access to a role that was previously made public.
CASCADE	Revokes the privilege from the specified user/role and all other users and roles that were granted the same privilege along a chain of recipients that can be traced back to the specified user.

**Note:** A user that has administrator access to a role can revoke the role for other users.

## REVOKE (Schema)

Revokes privileges on a schema from a user or role.

**Note:** In a database with trust authentication, the GRANT and REVOKE statements appear to work as expected but have no actual effect on the security of the database.

### Syntax

```
REVOKE [ GRANT OPTION FOR ] {
... { CREATE | USAGE } [ , ... ]
... | ALL [ PRIVILEGES ] }
... ON SCHEMA schema-name [ , ... ]
... FROM { username | PUBLIC | role } [ , ... ]
...[ CASCADE ]
```

### Parameters

GRANT OPTION FOR	Revokes the grant option for the privilege, not the privilege itself. If omitted, revokes both the privilege and the grant option.
CREATE	Allows the user read access to the schema and the right to create tables and views within the schema.
USAGE	Allows the user access to the objects contained within the schema. This allows the user to look up objects within the schema. Note that the user must also be granted access to the individual objects. See the <b>GRANT TABLE</b> (page 727) and <b>GRANT VIEW</b> (page 728) statements.
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
<i>schemaname</i>	Is the name of the schema for which privileges are being granted.
<i>username</i>	Grants the privilege to a specific user.
PUBLIC	Grants the privilege to all users.
<i>role</i>	Grants the privilege to a specific role.
CASCADE	Revokes the privilege from the specified user/role and all other users and roles that were granted the same privilege along a chain of recipients that can be traced back to the specified user.

## REVOKE (Sequence)

Revokes privileges on a sequence generator from a user or role.

## Syntax

```
REVOKE [ GRANT OPTION FOR ]
... { { USAGE | SELECT | UPDATE }
... | ALL [ PRIVILEGES ] }
... ON SEQUENCE [schema-name.]sequence_name [ , ... ]
... FROM { username | PUBLIC | role } [ , ... ]
...[ CASCADE ]
```

## Parameters

USAGE	Revokes the right to use both the CURRVAL and NEXTVAL functions on the specified sequence.
SELECT	Revokes the right to use the CURRVAL function on the specified sequence.
UPDATE	Revokes the right to use the NEXTVAL function on the specified sequence.
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
<i>[schema-name].sequence_name</i>	Specifies the sequence from which to revoke privileges. When using more than one schema, specify the schema that contains the sequence from which to revoke privileges.
<i>username</i>	Revokes the privilege from the specified user.
PUBLIC	Revokes the privilege from all users.
<i>role</i>	Revokes the privilege from the specified role.
CASCADE	Revokes the privilege from the specified user/role and all other users and roles that were granted the same privilege along a chain of recipients that can be traced back to the specified user.

## REVOKE (Table)

Revokes privileges on a table from a user or role.

**Note:** In a database with trust authentication, the GRANT and REVOKE statements appear to work as expected but have no actual effect on the security of the database.

### Syntax

```
REVOKE [ GRANT OPTION FOR ]
... { { SELECT | INSERT | UPDATE | DELETE | REFERENCES } [ , ... ]
... | ALL [ PRIVILEGES ] }
... ON [ TABLE ] [ schema-name. ] tablename [ , ... ]
... FROM { username | PUBLIC | role } [ , ... ]
... [ CASCADE ]
```

### Parameters

GRANT OPTION FOR	Revokes the grant option for the privilege, not the privilege itself. If omitted, revokes both the privilege and the grant option.
SELECT	Allows the user to SELECT from any column of the specified table.
INSERT	Allows the user to INSERT tuples into the specified table and to use the <b>COPY</b> (page 607) command to load the table. <b>Note:</b> COPY FROM STDIN is allowed to any user granted the INSERT privilege, while COPY FROM <file> is an admin-only operation.
UPDATE	Allows the user to UPDATE tuples in the specified table.
DELETE	Allows DELETE of a row from the specified table.
REFERENCES	Is necessary to have this privilege on both the referencing and referenced tables in order to create a foreign key constraint.
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
[ <i>schema-name.</i> ] <i>tablename</i>	Specifies the table on which to grant the privileges. When using more than one schema, specify the schema that contains the table on which to grant privileges.
<i>username</i>	Grants the privilege to the specified user.
PUBLIC	Grants the privilege to all users.
<i>role</i>	Grants the privilege to the specified role.
CASCADE	Revokes the privilege from the specified user/role and all other users and roles that were granted the same privilege along a chain of recipients that can be traced back to the specified user.

## REVOKE (Transform Function)

Revokes the `EXECUTE` privilege on a User Defined Transform Function (UDTF) from a database user or role.

### Syntax

```
REVOKE EXECUTE
... ON TRANSFORM FUNCTION [schema-name.]function-name [, ...]
... ( [ argname ] argtype [ ,... ] )
... FROM { username | PUBLIC | role } [ , ... ]
... [ CASCADE ]
```

### Parameters

<i>[schema-name].function-name</i>	Specifies the UDTF from which to revoke the <code>EXECUTE</code> privilege. When using more than one schema, specify the schema that contains the function.
<i>argname</i>	Specifies the argument name or names.
<i>argtype</i>	Specifies the argument data type or types.
<i>username</i>	Revokes the privilege from the specified user.
PUBLIC	Revokes the privilege from all users.
<i>role</i>	Revokes the privilege from the specified role.
CASCADE	Revokes the privilege from the specified user and all other users who were granted the same privilege along a chain of recipients that can be traced back to the specified user.

### Permissions

Only the superuser and owner can revoke `EXECUTE` privilege on a UDTF.

### Example

The following command revokes `EXECUTE` privileges from user Bob on the `tokenize` function:

```
=> REVOKE EXECUTE ON TRANSFORM FUNCTION tokenize(varchar) FROM Bob;
```

### See Also

***GRANT (Transform Function)*** (page 728)

## REVOKE (View)

Revokes privileges on a view from a user.

**Note:** In a database with trust authentication, the `GRANT` and `REVOKE` statements appear to work as expected but have no actual effect on the security of the database.

**Syntax**

```

REVOKE [ GRANT OPTION FOR ]
... { { SELECT } }
... ON [ VIEW ] [schema-name.]viewname [ , ... ]
... FROM { username | PUBLIC } [ , ... ]
... [ CASCADE ]

```

**Parameters**

GRANT OPTION FOR	Revokes the grant option for the privilege, not the privilege itself. If omitted, revokes both the privilege and the grant option.
SELECT	Allows the user to perform SELECT operations on a view and the resources referenced within it.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
[ <i>schema-name</i> .] <i>viewname</i>	Specifies the view on which to revoke the privileges. When using more than one schema, specify the schema that contains the view.
<i>username</i>	Revokes the privilege from the specified user.
PUBLIC	Revokes the privilege from all users.
CASCADE	Revokes the privilege from the specified user and all other users who were granted the same privilege along a chain that can be traced back to the specified users.

**See Also**

Granting and Revoking Privileges

## ROLLBACK

Ends the current transaction and discards all changes that occurred during the transaction.

### Syntax

```
ROLLBACK [ WORK | TRANSACTION ]
```

### Parameters

WORK TRANSACTION	Have no effect; they are optional keywords for readability.
---------------------	---

### Notes

When an operation is rolled back, any locks that are acquired by the operation are also rolled back.

## ROLLBACK TO SAVEPOINT

Rolls back all commands that have been entered within the transaction since the given savepoint was established.

### Syntax

```
ROLLBACK TO [SAVEPOINT] savepoint_name
```

### Parameters

<i>savepoint_name</i>	Specifies the name of the savepoint to roll back to.
-----------------------	--

### Notes

- The savepoint remains valid and can be rolled back to again later if needed.
- When an operation is rolled back, any locks that are acquired by the operation are also rolled back.
- ROLLBACK TO SAVEPOINT implicitly destroys all savepoints that were established after the named savepoint.

### Example

The following example rolls back the values 102 and 103 that were entered after the savepoint, `my_savepoint`, was established. Only the values 101 and 104 are inserted at commit.

```
=> INSERT INTO product_key VALUES (101);
=> SAVEPOINT my_savepoint;
=> INSERT INTO product_key VALUES (102);
=> INSERT INTO product_key VALUES (103);
=> ROLLBACK TO SAVEPOINT my_savepoint;
=> INSERT INTO product_key VALUES (104);
=> COMMIT;
```

**See Also**

**RELEASE SAVEPOINT** (page 733) and **SAVEPOINT** (page 745)

**SAVEPOINT**

Creates a special mark, called a savepoint, inside a transaction. A savepoint allows all commands that are executed after it was established to be rolled back, restoring the transaction to the state it was in at the point in which the savepoint was established.

**Tip:** Savepoints are useful when creating nested transactions. For example, a savepoint could be created at the beginning of a subroutine. That way, the result of the subroutine could be rolled back if necessary.

**Syntax**

```
SAVEPOINT savepoint_name
```

**Parameters**

<i>savepoint_name</i>	Specifies the name of the savepoint to create.
-----------------------	--

**Notes**

- Savepoints are local to a transaction and can only be established when inside a transaction block.
- Multiple savepoints can be defined within a transaction.
- If a savepoint with the same name already exists, it is replaced with the new savepoint.

**Example**

The following example illustrates how a savepoint determines which values within a transaction can be rolled back. The values 102 and 103 that were entered after the savepoint, `my_savepoint`, was established are rolled back. Only the values 101 and 104 are inserted at commit.

```
=> INSERT INTO T1 (product_key) VALUES (101);
=> SAVEPOINT my_savepoint;
=> INSERT INTO T1 (product_key) VALUES (102);
=> INSERT INTO T1 (product_key) VALUES (103);
=> ROLLBACK TO SAVEPOINT my_savepoint;
=> INSERT INTO T1 (product_key) VALUES (104);
=> COMMIT;
=> SELECT product_key FROM T1;
--
101
104
(2 rows)
```

**See Also**

**RELEASE SAVEPOINT** (page 733) and **ROLLBACK TO SAVEPOINT** (page 744)



## SELECT

Retrieves a result set from one or more tables.

### Syntax

```
[ AT EPOCH LATEST ] | [ AT TIME 'timestamp' ]
SELECT [ ALL | DISTINCT ] ( expression [, ...] ) ] ]
... *
... | expression [ AS ] output_name ] [ , ... ]
... [ INTO (page 748) ]
... [ FROM (page 750) [, ...] ]
... [ WHERE (page 752) condition ]
... [ TIMESERIES (page 761) slice_time ]
... [ GROUP BY (page 754) expression [, ...] ]
... [ HAVING (page 756) condition [, ...] ]
... [ WINDOW window_name AS ( window_definition_clause ) [ ,... ] ]
... [ UNION (page 788) ]
... [ ORDER BY (page 757) expression [ ASC | DESC ] [ ,... ] ]
... [ LIMIT (page 759) { count | ALL } ]
... [ OFFSET (page 760) start ]
... [ FOR UPDATE [ OF table_name [ , ... ] ] ]
```

### Parameters

AT EPOCH LATEST	Queries all data in the database up to but not including the current epoch without holding a lock or blocking write operations. See Snapshot Isolation for more information. AT EPOCH LATEST is ignored when applied to temporary tables (all rows are returned). By default, queries run under the READ COMMITTED isolation level, which means: <ul style="list-style-type: none"> <li>▪ AT EPOCH LATEST includes data from the latest committed DML transaction.</li> <li>▪ Each epoch contains exactly one transaction—the one that modified the data.</li> <li>▪ The Tuple Mover can perform moveout and mergeout operations on committed data immediately.</li> </ul>
AT TIME 'timestamp'	Queries all data in the database up to and including the epoch representing the specified date and time without holding a lock or blocking write operations. This is called a historical query. AT TIME is ignored when applied to temporary tables (all rows are returned).
*	Is equivalent to listing all columns of the tables in the FROM Clause. Vertica recommends that you avoid using SELECT * for performance reasons. An extremely large and wide result set can cause swapping.
DISTINCT	Removes duplicate rows from the result set (or group). The DISTINCT set quantifier must immediately follow the SELECT keyword. Only one DISTINCT keyword can appear in the select list.
expression	Forms the output rows of the SELECT statement. The expression can contain:

	<ul style="list-style-type: none"> <li>▪ <b>Column references</b> (page 46) to columns computed in the <code>FROM</code> clause</li> <li>▪ <b>Literals</b> (page 17) (constants)</li> <li>▪ <b>Mathematical operators</b> (page 40)</li> <li>▪ <b>String concatenation operators</b> (page 42)</li> <li>▪ <b>Aggregate expressions</b> (page 44)</li> <li>▪ <b>CASE expressions</b> (page 45)</li> <li>▪ <b>SQL functions</b> (page 109)</li> </ul>
<code>output_name</code>	Specifies a different name for an output column. This name is primarily used to label the column for display. It can also be used to refer to the column's value in <code>ORDER BY</code> and <code>GROUP BY</code> clauses, but not in the <code>WHERE</code> or <code>HAVING</code> clauses.
<code>FOR UPDATE</code>	Is most often used from <code>READ COMMITTED</code> isolation. When specified, the <code>SELECT</code> statement takes an X lock on all tables in the query. The <code>FOR UPDATE</code> keywords require update/delete permissions on the tables involved and cannot be issued from a read-only transaction.

### Example

When multiple clients run transactions like in the following example query, deadlocks can occur if `FOR UPDATE` is not used. Two transactions acquire an S lock, and when both attempt to upgrade to an X lock, they encounter deadlocks:

```
=> SELECT balance FROM accounts WHERE account_id=3476 FOR UPDATE; ...
=> UPDATE accounts SET balance = balance+10 WHERE account_id=3476;
=> COMMIT;
```

### See Also

**LOCKS** (page 858)

**Analytic Functions** (page 128)

Using SQL Analytics and Using Time Series Analytics in the Programmer's Guide

Subqueries and Joins in the Programmer's Guide

## INTO Clause

Creates a new table from the results of a query and fills it with data from the query.

### Syntax

```
INTO [ { GLOBAL | LOCAL } { TEMPORARY | TEMP } ]
... [ TABLE ] table-name
... [ON COMMIT { PRESERVE | DELETE } ROWS ]
```

## Parameters

GLOBAL	[Optional] Specifies that the table definition is visible to all sessions.
LOCAL	[Optional] Specifies that the table is visible only to the user who creates it for the duration of the session. When the session ends, the table definition is automatically dropped from the database catalogs.
TABLE	[Optional] Specifies that a table is to be created.
<i>table-name</i>	Specifies the name of the table to be created.
ON COMMIT { PRESERVE   DELETE } ROWS	<p>[Optional] Specifies whether data is transaction- or session-scoped:</p> <ul style="list-style-type: none"> <li>▪ DELETE marks a temporary table for transaction-scoped data. Vertica truncates the table (delete all its rows) after each commit. DELETE ROWS is the default.</li> <li>▪ PRESERVE marks a temporary table for session-scoped data, which is preserved beyond the lifetime of a single transaction. Vertica truncates the table (delete all its rows) when you terminate a session.</li> </ul>

## Example

The following statement creates a table called newtable and fills it with the data from customer\_dimension:

```
=> SELECT * INTO newtable FROM customer_dimension;
```

The following statement creates a temporary table called newtable and fills it with the data from customer\_dimension:

```
=> SELECT * INTO temp TABLE newtable FROM customer_dimension;
```

The following example creates a local temporary table and inserts the contents from mytable into it:

```
=> SELECT * INTO LOCAL TEMP TABLE ltt FROM mytable;
WARNING: No rows are inserted into table "v_temp_schema"."ltt" because ON
COMMIT DELETE ROWS
is the default for create temporary table
HINT: Use "ON COMMIT PRESERVE ROWS" to preserve the data in temporary table
CREATE TABLE
```

## See Also

Creating Temporary Tables in the Administrator's Guide

## FROM Clause

Specifies one or more source tables from which to retrieve rows.

### Syntax

```
FROM table-reference (on page 750) [ , ... ]
... [ subquery ] [AS] name ...
```

### Parameters

<i>table-reference</i>	Is a <b><i>table-primary</i></b> (on page 750) or a <b><i>joined-table</i></b> (on page 751).
------------------------	---

### Example

The following example returns all records from the `customer_dimension` table:

```
=> SELECT * FROM customer_dimension
```

### table-reference

#### Syntax

```
table-primary (on page 750) | joined-table (on page 751)
```

#### Parameters

<i>table-primary</i>	Specifies an optionally qualified table name with optional table aliases, column aliases, and outer joins.
<i>joined-table</i>	Specifies an outer join.

### table-primary

#### Syntax

```
{ table-name [ AS ] alias
  [ ( column-alias [ , ... ] ) ] [ , ... ] ]
| ( joined-table (on page 751) ) }
```

#### Parameters

<i>table-name</i>	Specifies a table in the logical schema. Vertica selects a suitable projection to use.
<i>alias</i>	Specifies a temporary name to be used for references to the table.
<i>column-alias</i>	Specifies a temporary name to be used for references to the column.
<i>joined-table</i>	Specifies an outer join.

## joined-table

### Syntax

*table-reference* *join-type* *table-reference*  
ON *join-predicate* (on page 58)

### Parameters

<i>table-reference</i>	Is a <i>table-primary</i> (page 750) or another <i>joined-table</i> .
<i>join-type</i>	Is one of the following: INNER JOIN LEFT [ OUTER ] JOIN RIGHT [ OUTER ] JOIN FULL [ OUTER ] JOIN
<i>join-predicate</i>	An equi-join based on one or more columns in the joined tables.

### Notes

A query that uses INNER JOIN syntax in the FROM clause produces the same result set as a query that uses the WHERE clause to state the join-predicate. See Joins in the Programmer's Guide for more information.

## WHERE Clause

Eliminates rows from the result table that do not satisfy one or more predicates.

### Syntax

```
WHERE boolean-expression
      [ subquery ] ...
```

### Parameters

<i>boolean-expression</i>	Is an expression that returns true or false. Only rows for which the expression is true become part of the result set.
---------------------------	--

The *boolean-expression* can include **Boolean operators** (on page 37) and the following elements:

- **BETWEEN-predicate** (on page 51)
- **Boolean-predicate** (on page 52)
- **Column-value-predicate** (on page 53)
- **IN-predicate** (on page 54)
- **Join-predicate** (on page 58)
- **LIKE-predicate** (on page 59)
- **NULL-predicate** (on page 63)

### Notes

You can use parentheses to group expressions, predicates, and boolean operators. For example:

```
=> ... WHERE NOT (A=1 AND B=2) OR C=3;
```

### Example

The following example returns the names of all customers in the Eastern region whose name starts with 'Amer'. Without the WHERE clause filter, the query returns *all* customer names in the customer\_dimension table.

```
=> SELECT DISTINCT customer_name
      FROM customer_dimension
      WHERE customer_region = 'East'
      AND customer_name ILIKE 'Amer%';
customer_name
-----
Americare
Americom
Americore
Americorp
Ameridata
Amerigen
Amerihope
Amerimedia
Amerishop
Ameristar
```

Ameritech  
(11 rows)

## GROUP BY Clause

Divides a query result set into sets of rows that match an expression.

### Syntax

```
GROUP BY expression [ ,... ]
```

### Parameters

<i>expression</i>	Is any expression including constants and <b>references to columns</b> (see " <b>Column References</b> " on page 46) in the tables specified in the FROM clause. For example:  column1, ..., column_n, aggregate_function ( <i>expression</i> )
-------------------	--

### Notes

- The *expression* cannot include **aggregate functions** (page 110); however, the GROUP BY clause is often used with **aggregate functions** (page 110) to return summary values for each group.
- The GROUP BY clause without aggregates is similar to using SELECT DISTINCT. For example, the following two queries are equal:  

```
SELECT DISTINCT household_id from customer_dimension;
SELECT household_id from customer_dimension GROUP BY household_id;
```
- All non-aggregated columns in the SELECT list must be included in the GROUP BY clause.
- Using the WHERE clause with the GROUP BY clause is useful in that all rows that do not satisfy the WHERE clause conditions are eliminated before any grouping operations are computed.
- The GROUP BY clause does not order data. If you want to sort data a particular way, place the **ORDER BY clause** (page 757) after the GROUP BY clause.

### Examples

In the following example, the WHERE clause filters out all employees whose last name does not begin with S. The GROUP BY clause returns the groups of last names that begin with S, and the SUM aggregate function computes the total vacation days for each group.

```
=> SELECT employee_last_name, SUM(vacation_days)
   FROM employee_dimension
   WHERE employee_last_name ILIKE 'S%'
   GROUP BY employee_last_name;
employee_last_name | SUM
-----+-----
Sanchez            | 2892
Smith              | 2672
Stein              | 2660
(3 rows)
=> SELECT vendor_region, MAX(deal_size) as "Biggest Deal"
   FROM vendor_dimension
   GROUP BY vendor_region;
vendor_region | Biggest Deal
```

-----+-----	
East	990889
MidWest	699163
NorthWest	76101
South	854136
SouthWest	609807
West	964005

(6 rows)

The only difference between the following query and the one before it is the HAVING clause filters the groups to deal sizes greater than \$900,000:

```
=> SELECT vendor_region, MAX(deal_size) as "Biggest Deal"
      FROM vendor_dimension
      GROUP BY vendor_region
      HAVING MAX(deal_size) > 900000;
```

-----+-----	
East	990889
West	964005

(2 rows)

## HAVING Clause

Restricts the results of a **GROUP BY clause** (page 754).

### Syntax

```
HAVING condition [, ...]
```

### Parameters

<i>condition</i>	Must unambiguously reference a grouping column, unless the reference appears within an aggregate function
------------------	---

### Notes

- Semantically the having clause occurs after the group by operation.
- You can use expressions in the HAVING clause.
- The HAVING clause was added to the SQL standard because you cannot use WHERE with **aggregate functions** (page 110).

### Example

The following example returns the employees with salaries greater than \$50,000:

```
=> SELECT employee_last_name, MAX(annual_salary) as "highest_salary"
      FROM employee_dimension
      GROUP BY employee_last_name
      HAVING MAX(annual_salary) > 50000;
employee_last_name | highest_salary
```

```
-----+-----
Bauer              |          920149
Brown              |          569079
Campbell          |          649998
Carcetti           |          195175
Dobisz             |          840902
Farmer             |          804890
Fortin             |          481490
Garcia             |          811231
Garnett            |          963104
Gauthier           |          927335
(10 rows)
```

## ORDER BY Clause

Sorts a query result set on one or more columns.

### Syntax

```
ORDER BY expression [ ASC | DESC ] [, ...]
```

### Parameters

<i>expression</i>	<p>Can be:</p> <ul style="list-style-type: none"> <li>▪ The name or <b>ordinal number</b> (<a href="http://en.wikipedia.org/wiki/Ordinal_number">http://en.wikipedia.org/wiki/Ordinal_number</a>) of a SELECT list item</li> <li>▪ An arbitrary expression formed from columns that do not appear in the SELECT list</li> <li>▪ A <b>CASE</b> (page 45) expression</li> </ul>
-------------------	---

### Notes

- The ordinal number refers to the position of the result column, counting from the left beginning at one. This makes it possible to order by a column that does not have a unique name. (You can assign a name to a result column using the AS clause.)
- Vertica uses the ASCII collating sequence to store data and to compare character strings. The ordering varies by collation.
- For INTEGER, INT, and DATE/TIME data types, NULL appears first (smallest) in ascending order.
- For FLOAT, BOOLEAN, CHAR, and VARCHAR, NULL appears last (largest) in ascending order.

### Example

The follow example returns all the city and deal size for customer Metamedia, sorted by deal size in descending order.

```
=> SELECT customer_city, deal_size
   FROM customer_dimension
   WHERE customer_name = 'Metamedia'
   ORDER BY deal_size DESC;
customer_city | deal_size
-----+-----
El Monte     | 4479561
Athens       | 3815416
Ventura      | 3792937
Peoria       | 3227765
Arvada       | 2671849
Coral Springs | 2643674
Fontana      | 2374465
Rancho Cucamonga | 2214002
Wichita Falls | 2117962
Beaumont    | 1898295
Arvada      | 1321897
```

Waco		1026854
Joliet		945404
Hartford		445795

(14 rows)

## LIMIT Clause

Specifies the maximum number of result set rows to return.

### Syntax

```
LIMIT { rows | ALL }
```

### Parameters

<code>rows</code>	Specifies the maximum number of rows to return
<code>ALL</code>	Returns all rows (same as omitting LIMIT)

### Notes

When both LIMIT and **OFFSET** (page 760) are used, Vertica skips the specified number of rows before it starts to count the rows to be returned.

You can use LIMIT without an **ORDER BY clause** (page 757) that includes all columns in the select list, but the query could produce nondeterministic results.

**Nondeterministic:** Omits the ORDER BY clause and returns *any* five records from the customer\_dimension table:

```
=> SELECT customer_city
      FROM customer_dimension
      LIMIT 5;
customer_city
-----
Baltimore
Nashville
Allentown
Clarksville
Baltimore
(5 rows)
```

**Deterministic:** Specifies the ORDER BY clause:

```
=> SELECT customer_city
      FROM customer_dimension
      ORDER BY customer_city
      LIMIT 5;
customer_city
-----
Abilene
Abilene
Abilene
Abilene
Abilene
(5 rows)
```

## OFFSET Clause

Omits a specified number of rows from the beginning of the result set.

### Syntax

```
OFFSET rows
```

### Parameters

<i>rows</i>	specifies the number of result set rows to omit.
-------------	--

### Notes

- When both **LIMIT** (page 759) and OFFSET are specified, specified number of rows are skipped before starting to count the rows to be returned.
- When using OFFSET, use an **ORDER BY clause** (page 757). Otherwise the query returns an undefined subset of the result set.

### Example

The following example is similar to the the example used in the **LIMIT clause** (page 759). If you want to see just records 6-10, however, use the OFFSET clause to skip over the first five cities:

```
=> SELECT customer_city
      FROM customer_dimension
      WHERE customer_name = 'Metamedia'
      ORDER BY customer_city
      OFFSET 5;
      customer_city
-----
El Monte
Fontana
Hartford
Joliet
Peoria
Rancho Cucamonga
Ventura
Waco
Wichita Falls
(9 rows)
```

The following are the results without the OFFSET clause:

```
      customer_city
-----
Arvada
Arvada
Athens
Beaumont
Coral Springs
El Monte
Fontana
Hartford
Joliet
```

```

Peoria
Rancho Cucamonga
Ventura
Waco
Wichita Falls
(14 rows)

```

## TIMESERIES Clause

Provides gap-filling and interpolation (GFI) computation, an important component of time series analytics computation. See Using Time Series Analytics in the Programmer's Guide for details and examples.

### Syntax

```

TIMESERIES slice_time AS 'length_and_time_unit_expression' OVER (
... [ window_partition_clause (page 130) [ , ... ] ]
... ORDER BY time_expression )
... [ ORDER BY table_column [ , ... ] ]

```

### Parameters

<i>slice_time</i>	A time column produced by the TIMESERIES clause, which stores the time slice start times generated from gap filling. Note: This parameter is an alias, so you can use any name that an alias would take.
' <i>length_and_time_unit_expression</i> '	Is INTERVAL (DAY TO SECOND) that represents the length of time unit of time slice computation; for example, TIMESERIES <i>slice_time</i> AS '3 seconds' ...
OVER()	Specifies partitioning and ordering for the function. OVER() also specifies that the time series function operates on a query result set (the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses have been evaluated).
PARTITION BY	Partitions the data by expressions ( <i>column1</i> ..., <i>column_n</i> , <i>slice_time</i> ).
<i>expression</i>	Expressions on which to partition the data, where each partition is sorted by <i>time_expression</i> . Gap filling and interpolation is performed on each partition separately.
ORDER BY	Sorts the data by <i>time_expression</i> .
<i>time_expression</i>	An expression that computes the time information of the time series data. The <i>time_expression</i> can be TIMESTAMP data type only.

### Notes

If the *window\_partition\_clause* is not specified in TIMESERIES OVER(), for each defined time slice, exactly one output record is produced; otherwise, one output record is produced per partition per time slice. Interpolation is computed there.

Given a query block that contains a TIMESERIES clause, the following are the semantic phases of execution (after evaluating the FROM and the optional WHERE clauses):

- 1 Compute *time\_expression*.
- 2 Perform the same computation as the TIME\_SLICE() function on each input record based on the result of *time\_expression* and '*length\_and\_time\_unit\_expression*'.
  1. Perform gap filling to generate time slices missing from the input.
  2. Name the result of this computation as *slice\_time*, which represents the generated “time series” column (alias) after gap filling.
- 3 Partition the data by *expression*, *slice\_time*. For each partition, do step 4.
- 4 Sort the data by *time\_expression*. Interpolation is computed here.

There is semantic overlap between the TIMESERIES clause and the **TIME\_SLICE** (page 225) function with the following key differences:

- TIMESERIES only supports the DAY TO SECOND *interval-qualifier* (on page 34) (does not allow YEAR TO MONTH).
- Unlike TIME\_SLICE, the time slice length and time unit expressed in *length\_and\_time\_unit\_expr* must be constants so gaps in the time slices are well-defined.
- TIMESERIES performs gap filling; the TIME\_SLICE function does not.
- TIME\_SLICE can return the start or end time of a time slice, depending on the value of its fourth input parameter (*start\_or\_end*). TIMESERIES, on the other hand, always returns the start time of each time slice. To output the end time of each time slice, you can write a SELECT statement like the following:

```
SELECT slice_time + <slice_length>;
```

### Restrictions

- When the TIMESERIES clause occurs in a SQL query block, only SELECT, FROM, WHERE, and ORDER BY clauses can be used in that same query block. GROUP BY and HAVING clauses are not allowed.

If a GROUP BY operation is needed before or after gap-filling and interpolation (GFI), use a subquery and place the GROUP BY in the outer query. For example:

```
=> SELECT symbol, AVG(first_bid) as avg_bid FROM (  
    SELECT symbol, slice_time, TS_FIRST_VALUE(bid1) AS first_bid  
    FROM Tickstore  
    WHERE symbol IN ('MSFT', 'IBM')  
    TIMESERIES slice_time AS '5 seconds' OVER (PARTITION BY symbol  
    ORDER BY ts)  
    ) AS resultOfGFI
```

```
GROUP BY symbol;
```

- When the TIMESERIES clause is present in the SQL query block, only time series aggregate functions (such as **TS\_FIRST\_VALUE** (page 471) and **TS\_LAST\_VALUE** (page 472)), the *slice\_time* column, PARTITION BY expressions, and **TIME\_SLICE** (page 225) are allowed in the SELECT list. For example, the following two queries would return a syntax error because bid1 was not a PARTITION BY or GROUP BY column:

```
=> SELECT bid, symbol, TS_FIRST_VALUE(bid) FROM Tickstore  
    TIMESERIES slice_time AS '5 seconds' OVER (PARTITION BY symbol ORDER  
    BY ts);  
ERROR: column "Tickstore.bid" must appear in the PARTITION BY list  
of Timeseries clause or be used in a Timeseries Output function
```

```
=> SELECT bid, symbol, AVG(bid) FROM Tickstore
      GROUP BY symbol;
      ERROR: column "Tickstore.bid" must appear in the GROUP BY clause or
      be used in an aggregate function
```

- If you use the analytic `OVER(window_order_clause)` (page 131), you can order the data by a `TIMESTAMP` column only, not by, for example, an `INTEGER` column.

## Examples

See Gap Filling and Interpolation (GFI) in the Programmer's Guide.

## See Also

`TIME_SLICE` (page 225), `TS_FIRST_VALUE` (page 471), and `TS_LAST_VALUE` (page 472)

Using Time Series Analytics in the Programmer's Guide

## MATCH Clause

A SQL extension that lets you screen large amounts of historical data in search of event patterns, the `MATCH` clause provides subclasses for analytic partitioning and ordering and matches rows from the result table based on a pattern you define.

You specify a pattern as a regular expression, which is composed of event types defined in the `DEFINE` subclass, where each event corresponds to a row in the input table. Then you can search for the pattern within a sequence of input events. Pattern matching returns the contiguous sequence of rows that conforms to `PATTERN` subclass. For example, pattern `P (A B* C)` consist of three event types: A, B, and C. When Vertica finds a match in the input table, the associated pattern instance must be an event of type A followed by 0 or more events of type B, and an event of type C.

Pattern matching is particularly useful for clickstream analysis where you might want to identify users' actions based on their Web browsing behavior (page clicks). A typical online clickstream funnel is:

Company home page -> product home page -> search -> results -> purchase online

Using the above clickstream funnel, you can search for a match on the user's sequence of web clicks and identify that the user:

- landed on the company home page
- navigated to the product page
- ran a search
- clicked a link from the search results
- made a purchase

For examples that use this clickstream model, see Event Series Pattern Matching in the Programmer's Guide.

## Syntax

```
MATCH ( [ PARTITION BY table_column ] ORDER BY table_column
... DEFINE event_name AS boolean_expr [, ...]
... PATTERN pattern_name AS ( regex )
```

```
... [ ROWS MATCH ( ALL EVENTS | FIRST EVENT ) ] )
```

## Parameters

PARTITION BY	[Optional] Defines the window data scope in which the pattern, defined in the PATTERN subclause, is matched. The partition clause partitions the data by matched patterns defined in the PATTERN subclause. For each partition, data is sorted by the ORDER BY clause. If the partition clause is omitted, the entire data set is considered a single partition.
ORDER BY	Defines the window data scope in which the pattern, defined in the PATTERN subclause, is matched. For each partition, the order clause specifies how the input data is ordered for pattern matching. <b>Note:</b> The ORDER BY clause is mandatory.
DEFINE	Defines the <b>boolean</b> (page 52) expressions that make up the event types in the regular expressions. For example: <pre>DEFINE Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%', Onsite AS PageURL ILIKE '%website2.com%' AND Action='V', Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'</pre> The DEFINE subclause accepts a maximum of 52 events. See Event Series Pattern Matching in the Programmer's Guide for examples.
<i>event_name</i>	Is the name of the event to evaluate for each row; for example, Entry, Onsite, Purchase. <b>Note:</b> Event names are case insensitive and follow the same naming conventions as those used for tables and columns.
<i>boolean_expr</i>	Is an expression that returns true or false. <i>boolean_expr</i> can include <b>Boolean operators</b> (on page 37) and relational ( <b>comparison</b> ) (see " <b>Comparison Operators</b> " on page 37) operators. For example: <pre>Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'</pre>
PATTERN <i>pattern_name</i>	Is the name of the pattern, which you define in the PATTERN subclause; for example, P is the pattern name defined below: <pre>PATTERN P AS (...)</pre> PATTERN defines the regular expression composed of event types that you specify in the DEFINE subclause. <b>Note:</b> Vertica supports one pattern per query.
<i>regex</i>	Is the pattern that is composed of the event types defined in the DEFINE subclause and which returns the contiguous sequence of rows that conforms to the PATTERN subclause. <ul style="list-style-type: none"><li>* Match 0 or more times</li><li>*? Match 0 or more times, not greedily</li><li>+ Match 1 or more times</li><li>+? Match 1 or more times, not greedily</li><li>? Match 0 or 1 time</li><li>?? Match 0 or 1 time, not greedily</li></ul>

	<p>*+ Match 0 or more times, possessive</p> <p>++ Match 1 or more times, possessive</p> <p>?+ Match 0 or 1 time, possessive</p> <p><b>Note:</b> Syntax for regular expressions is compatible with the Perl 5 regular expression syntax. See the <i>Perl Regular Expressions Documentation</i> (<a href="http://perldoc.perl.org/perlre.html">http://perldoc.perl.org/perlre.html</a>) for details.</p>
ROWS MATCH	<p>[Optional] Defines how to resolve more than one event evaluating to true for a single row.</p> <ul style="list-style-type: none"> <li>▪ If you use ROWS MATCH ALL EVENTS (default), Vertica returns the following run-time error if more than one event evaluates to true for a single row: <ul style="list-style-type: none"> <li>▪ <code>ERROR: pattern events must be mutually exclusive</code></li> <li>▪ <code>HINT: try using ROWS MATCH FIRST EVENT</code></li> </ul> </li> <li>▪ For ROWS MATCH FIRST EVENT, if more than one event evaluates to true for a single row, Vertica chooses the event defined first in the SQL statement to be the event it uses for the row.</li> </ul>

### Pattern Semantic Evaluation

- The semantic evaluating ordering of the SQL clauses is: FROM -> WHERE -> PATTERN MATCH -> SELECT.
- Data is partitioned as specified in the the PARTITION BY clause. If the partition clause is omitted, the entire data set is considered a single partition.
- For each partition, the order clause specifies how the input data is ordered for pattern matching.
- Events are evaluated for each row. A row could have 0, 1, or *N* events evaluate to true. If more than one event evaluates to true for the same row, Vertica returns a run-time error unless you specify ROWS MATCH FIRST EVENT. If you specify ROWS MATCH FIRST EVENT and more than one event evaluates to TRUE for a single row, Vertica chooses the event that was defined first in the SQL statement to be the event it uses for the row.
- Pattern matching is performed by finding the contiguous sequence of rows that conforms to the pattern defined in the PATTERN subclause.  
For each match, the rows that contribute to the match are output. Rows not part of the match (do not satisfy one or more predicates) are not output.
- Vertica reports only non-overlapping matches. If an overlap occurs, Vertica chooses the first match found in the input stream. After finding the match, Vertica looks for the next match, starting at the end of the previous match.
- Vertica reports the longest possible match, not a subset of a match. For example, consider pattern: A\*B with input: AAAB. Because A uses the greedy regular expression quantifier (\*), Vertica reports all A inputs (AAAB), not AAB, AB, or B.

### Notes and Restrictions

- There is no maximum imposed on the number of rows per partition or number of matches per partition, but if you encounter errors, you can increase the limits; for example, to increase the number of rows per partition:

```
=> SELECT set_config_parameter('PatternMatchingMaxPartition',  
    'NUMBER_OF_ROWS');
```

To increase the number of matches per partition:

```
=> SELECT set_config_parameter('PatternMatchingMaxPartitionMatches',  
    'NUMBER_OF_MATCHES');
```

- DISTINCT and GROUP BY/HAVING clauses are not allowed in pattern match queries.
- The following expressions are not allowed in the DEFINE subclause:
  - Subqueries: DEFINE X AS c IN (SELECT c FROM table1)
  - Analytic functions: DEFINE X AS c < LEAD(1) OVER (ORDER BY 1)
  - Aggregate functions: DEFINE X AS c < MAX(1)
- You cannot use the same pattern name to define a different event; for example, the following is not allowed:

```
DEFINE  
  X AS c1 < 3  
  X AS c1 >= 3
```
- Used with MATCH clause, Vertica *pattern matching functions* (page 519) provide additional data about the patterns it finds. For example, you can use the functions to return values representing the name of the event that matched the input row, the sequential number of the match, or a partition-wide unique identifier for the instance of the pattern that matched.

### Example

See Event Series Pattern Matching in the Programmer's Guide.

### See Also

*Pattern matching functions* (page 519): *EVENT\_NAME* (page 419), *MATCH\_ID* (page 437), and *PATTERN\_ID* (page 443)

Pattern Matching Partition Parameters in the Administrator's Guide.

*Perl Regular Expressions Documentation* (<http://perldoc.perl.org/perlre.html>)

## SET DATESTYLE

Changes the DATESTYLE run-time parameter for the current session.

### Syntax

```
SET DATESTYLE TO { value | 'value' } [ , ... ]
```

### Parameters

The DATESTYLE parameter can have multiple, non-conflicting values:

Value	Interpretation	Example
MDY	month-day-year	12/17/2007
DMY	day-month-year	17/12/2007
YMD	year-month-day	2007-12-17
ISO	ISO 8601/SQL standard (default)	2007-12-17 07:37:16-08
SQL	traditional style	12/17/2007 07:37:16.00 PST
GERMAN	regional style	17.12.2007 07:37:16.00 PST

In the SQL style, day appears before month if DMY field ordering has been specified, otherwise month appears before day. (See *Date/Time Literals* (page 28) for how this setting also affects interpretation of input values.) The table below shows an example.

DATESTYLE	Input Ordering	Example Output
SQL, DMY	day/month/year	17/12/2007 15:37:16.00 CET
SQL, MDY	month/day/year	12/17/2007 07:37:16.00 PST

### Notes

- Vertica ISO output for DATESTYLE is ISO long form, but several input styles are accepted. If year appears first YMD is used for input and output.
- The SQL standard requires the use of the ISO 8601 format. The name of the "SQL" output format is a historical accident.
- INTERVAL output looks like the input format, except that units like CENTURY or WEEK are converted to years and days, and AGO is converted to an appropriate sign. In ISO mode the output looks like  

```
[ quantity unit [ ... ] ] [ days ] [ hours:minutes:seconds ]
```
- The **SHOW** (page 785) command displays the run-time parameters.

### Example

```
=> SET DATESTYLE TO SQL, MDY;
=> SHOW DATESTYLE;
```

```
name      | setting
-----+-----
datestyle | ISO, MDY
(1 row)
```

## SET ESCAPE\_STRING\_WARNING

Issues a warning when a backslash is used in a string literal during the current session.

### Syntax

```
SET ESCAPE_STRING_WARNING TO { ON | OFF }
```

### Parameters

ON	[Default] Issues a warning when a back slash is used in a string literal. <b>Tip:</b> Organizations that have upgraded from earlier versions of Vertica can use this as a debugging tool for locating backslashes that used to be treated as escape characters, but are now treated as literals.
OFF	Ignores back slashes within string literals.

### Notes

- This statement works under vsql only.
- Turn off standard conforming strings before you turn on this parameter.

**Tip:** To set escape string warnings across all sessions, use the `EscapeStringWarnings` configuration parameter. See the Internationalization Parameters in the Administrator's Guide.

### Examples

The following example shows how to turn OFF escape string warnings for the session.

```
=> SET ESCAPE_STRING_WARNING TO OFF;
```

### See Also

**STANDARD\_CONFORMING\_STRINGS** (page 781)

## SET INTERVALSTYLE

Changes the INTERVALSTYLE run-time parameter for the current session.

### Syntax

```
SET INTERVALSTYLE TO [ plain | units (see "interval-literal" on page 32) ]
```

## Parameters

plain	[Default] Returns no units on output. When interval units are enabled, their format is controlled by <b>DATESTYLE</b> (page 767)
units	Returns units on output.

## Notes

- Use the **SHOW** (page 785) command displays the run-time parameters.
- When units are enabled, their format is controlled by **DATESTYLE** (page 767). If you are expecting units on output but not seeing them, issue the `SHOW DATESTYLE` command. **DATESTYLE** must be set to `ISO` for `INTERVAL` to display units on output.

## Examples

The following command sets the `INTERVALSTYLE` to show units on output:

```
=> SET INTERVALSTYLE TO UNITS;
SET
=> SELECT INTERVAL '3 2' DAY TO HOUR;
?column?
-----
3 days 02:00
(1 row)
```

The following command sets the `INTERVALSTYLE` to no units on output:

```
=> SET INTERVALSTYLE TO PLAIN;
SET
=> SELECT INTERVAL '3 2' DAY TO HOUR;
?column?
-----
3 02
(1 row)
```

Use the **SHOW** (page 785) command to display the run-time parameters:

```
=> SHOW INTERVALSTYLE;
name      | setting
-----+-----
intervalstyle | plain
(1 row)
```

## See Also

**INTERVAL** (page 74)

## SET LOCALE

Specifies the locale for the current session.

## Syntax

```
SET LOCALE TO < ICU-locale-identifier >
```

## Parameters

< ICU-locale-identifier >	<p>Specifies the ICU locale identifier to use.</p> <p>By default, the locale for the database is en_US@collation=binary (English as in the United States of America).</p> <p>ICU Locales were developed by the ICU Project. See the ICU User Guide (<a href="http://userguide.icu-project.org/locale">http://userguide.icu-project.org/locale</a>) for a complete list of parameters that can be used to specify a locale.</p> <p><b>Note:</b> The only keyword Vertica supports is the COLLATION keyword.</p>
---------------------------	--

## Notes

Though not inclusive, the following are some commonly-used locales:

- German (Germany) de\_DE
- English (Great Britain) en\_GB
- Spanish (Spain) es\_ES
- French (France) fr\_FR
- Portuguese (Brazil) pt\_BR
- Portuguese (Portugal) pt\_PT
- Russian (Russia) ru\_RU
- Japanese (Japan) ja\_JP
- Chinese (China, simplified Han) zh\_CN
- Chinese (Taiwan, traditional Han) zh\_Hant\_TW

## Session related:

- The locale setting is session scoped and applies to queries only (no DML/DDDL) run in that session. You cannot specify a locale for an individual query.
- The default locale for new sessions can be set using a configuration parameter

## Query related:

The following restrictions apply when queries are run with locale other than the default en\_US@collation=binary:

- Multicolumn NOT IN subqueries are not supported when one or more of the left-side NOT IN columns is of CHAR or VARCHAR data type. For example:
 

```
=> CREATE TABLE test (x VARCHAR(10), y INT);
=> SELECT ... FROM test WHERE (x,y) NOT IN (SELECT ...);
      ERROR: Multi-expression NOT IN subquery is not supported because a
      left hand expression could be NULL
```

**Note:** An error is reported even if columns test.x and test.y have a "NOT NULL" constraint.

- Correlated HAVING clause subqueries are not supported if the outer query contains a GROUP BY on a CHAR or a VARCHAR column. In the following example, the GROUP BY x in the outer query causes the error:
 

```
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT COUNT(*) FROM test t GROUP BY x HAVING x
      IN (SELECT x FROM test WHERE t.x||'a' = test.x||'a' );
      ERROR: subquery uses ungrouped column "t.x" from outer query
```
- Subqueries that use analytic functions in the HAVING clause are not supported. For example:
 

```
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT MAX(x)OVER(PARTITION BY 1 ORDER BY 1)
      FROM test GROUP BY x HAVING x IN (
      SELECT MAX(x) FROM test);
      ERROR: Analytics query with having clause expression that involves
      aggregates
      and subquery is not supported
```
- The operators LIKE/ILIKE do not currently respect UTF-8 character boundaries. Therefore, expressions such as 'SS' LIKE 'ß' and 'SS' ILIKE 'ß' always return false even in locales where 'SS' = 'ß' return true.

#### DML/DDDL related:

- SQL identifiers (such as table names, column names, and so on) are restricted to ASCII characters. For example, the following CREATE TABLE statement fails because it uses the non-ASCII ß in the table name:
 

```
=> CREATE TABLE straÙe(x int, y int);
      ERROR: Non-ASCII characters are not supported in names
```
- Projection sort orders are made according to the default en\_US@collation=binary collation. Thus, regardless of the session setting, issuing the following command creates a projection sorted by coll according to the binary collation:
 

```
=> CREATE PROJECTION p1 AS SELECT * FROM table1 ORDER BY coll;
```

Note that in such cases, *straÙe* and *strasse* would not be near each other on disk. Sorting by binary collation also means that sort optimizations do not work in locales other than binary. Vertica returns the following warning if you create tables or projections in a non-binary locale:

```
WARNING: Projections are always created and persisted in the default
Vertica locale. The current locale is de_DE
```
- When creating pre-join projections, the projection definition query does not respect the locale or collation setting. This means that when you insert data into the fact table of a pre-join projection, referential integrity checks are not locale or collation aware.

For example:

```
\locale LDE_S1      -- German
=> CREATE TABLE dim (coll varchar(20) primary key);
=> CREATE TABLE fact (coll varchar(20) references dim(coll));
=> CREATE PROJECTION pj AS SELECT * FROM fact JOIN dim
      ON fact.coll = dim.coll UNSEGMENTED ALL NODES;
=> INSERT INTO dim VALUES ('ß');
```

```
=> COMMIT;
```

The following INSERT statement fails with a "nonexistent FK" error even though 'ß' is in the dim table, and in the German locale 'SS' and 'ß' refer to the same character.

```
=> INSERT INTO fact VALUES('SS');
      ERROR:  Nonexistent foreign key value detected in FK-PK join (fact
x dim)
      using subquery and dim_node0001; value SS
=> => ROLLBACK;
=> DROP TABLE dim, fact CASCADE;
```

- When the locale is non-binary, the collation function is used to transform the input to a binary string which sorts in the proper order.

This transformation increases the number of bytes required for the input according to this formula:

$$\text{result\_column\_width} = \text{input\_octet\_width} * \text{CollationExpansion} + 4$$

CollationExpansion defaults to 5 and should be changed only under the supervision of Vertica **Technical Support** (on page 1).

- CHAR fields are displayed as fixed length, including any trailing spaces. When CHAR fields are processed internally, they are first stripped of trailing spaces. For VARCHAR fields, trailing spaces are usually treated as significant characters; however, trailing spaces are ignored when sorting or comparing either type of character string field using a non-BINARY locale.

## Examples

This example sets the locale for the session to en\_GB (English as in Great Britain).

```
SET LOCALE TO en_GB;
SET LOCALE TO en_GB;
INFO:  Locale: 'en_GB'
INFO:    English (United Kingdom)
INFO:  Short form: 'LEN'
```

You can also use the short form of a locale in this command:

```
SET LOCALE TO LEN;
INFO:  Locale: 'en'
INFO:    English
INFO:  Short form: 'LEN'
```

## See Also

Implement Locales for International Data Sets and Appendix: Locales in the Administrator's Guide

## SET ROLE

Enables a role for the current user's current session. The user will gain the permissions that have been granted to the role.

## Syntax

```
SET ROLE { role [, ...] | NONE | ALL | DEFAULT }
```

## Parameters

<pre>role [, ...]   NONE   ALL   DEFAULT</pre>	<p>The name of one or more roles to set as the current role, or one of the following keywords:</p> <ul style="list-style-type: none"> <li>▪ NONE disables all roles for the current session.</li> <li>▪ ALL enables all of the roles to which the user has access.</li> <li>▪ DEFAULT sets the current role to the user's default role.</li> </ul>
--	--

## Notes

- You can only set roles active that have been granted to the user. Use `SHOW AVAILABLE_ROLES` to get a list of the roles available to the user.
- The default role is the first role that was assigned to the user or set by the superuser via the **`ALTER USER`** (page 594) statement's `DEFAULT ROLE` argument.
- Enabling a role does not affect any other roles that are currently enabled. A user session can have more than one role enabled at a time. The user's permissions are the union of all the roles that are currently active, plus any permissions granted directly to the user.

## Example

```
=> SHOW ENABLED_ROLES; -- Which roles are active now.
      name          | setting
-----+-----
enabled roles |
(1 row)

=> SHOW AVAILABLE_ROLES; -- All roles the user can access
      name          |          setting
-----+-----
available roles | applogs, appadmin, appuser
(1 row)

=> SET ROLE applogs;
SET
=> SHOW ENABLED_ROLES;
      name          | setting
-----+-----
enabled roles | applogs
(1 row)

=> SET ROLE appuser;
SET
=> SHOW ENABLED_ROLES;
      name          |          setting
-----+-----
enabled roles | applogs, appuser
```

```
(1 row)

=> SET ROLE NONE; -- disable all roles
SET
dbadmin=> SHOW ENABLED_ROLES;
      name      | setting
-----+-----
enabled roles |
(1 row)
```

## SET SEARCH\_PATH

Specifies the order in which Vertica searches schemas when a SQL statement contains an unqualified table name.

Vertica provides the SET search\_path statement instead of the CURRENT\_SCHEMA statement found in some other databases.

### Syntax

```
SET SEARCH_PATH TO schemaname [ , ... ]
```

### Parameters

<i>schemaname</i>	<p>A comma-delimited list of schemas that indicates the order in which Vertica searches schemas when a SQL statement contains an unqualified table name. The default value for this parameter is "\$user", public'</p> <p>Where:</p> <ul style="list-style-type: none"> <li>▪ \$User is the schema with the same name as the current user. If the schema does not exist, \$User is ignored.</li> <li>▪ public is the public database. Public is ignored if there is no schema named 'public'.</li> </ul>
-------------------	--

### Notes

The first schema named in the search path is called the current schema. The current schema is the first schema that Vertica searches. It is also the schema in which new tables are created if the **CREATE TABLE** (page 664) command does not specify a schema name.

### Restrictions

None

### Examples

The following example shows the current search path settings:

```
=> SHOW SEARCH_PATH;
      name      | setting
-----+-----
```

```
search_path | "$user", public, v_catalog, v_monitor, v_internal  
(1 row)
```

The following example sets the order in which Vertica searches schemas to T1, U1, and V1:

```
=> SET SEARCH_PATH TO T1, U1, V1;
```

## SET SESSION CHARACTERISTICS

Sets the transaction characteristics for subsequent transactions of a user session. These are the isolation level and the access mode (read/write or read-only).

### Syntax

```
SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode [, ...]
where transaction_mode is one of:
    ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ
UNCOMMITTED }
    READ { ONLY | WRITE }
```

### Parameters

Isolation level, described in the following table, determines what data the transaction can access when other transactions are running concurrently. It does not apply to temporary tables. The isolation level cannot be changed after the first query (`SELECT`) or DML statement (`INSERT`, `DELETE`, `UPDATE`) if a transaction has been run. A transaction retains its isolation level until it completes, even if the session's transaction isolation level has changed mid-transaction. Vertica internal processes (such as the Tuple Mover and refresh operations) and DDL operations are run at `SERIALIZABLE` isolation to ensure consistency.

<code>SERIALIZABLE</code>	Provides the strictest level of SQL transaction isolation. This level emulates transactions run one after another, serially, rather than concurrently. It holds locks and blocks write operations and is thus not recommended for normal query operations.
<code>REPEATABLE READ</code>	Is automatically converted to <code>SERIALIZABLE</code> by Vertica.
<code>READ COMMITTED</code>	(Default) Allows concurrent transactions. Use <code>READ COMMITTED</code> isolation or Snapshot Isolation for normal query operations, but be aware that there is a subtle difference between them. (See section below this table.)
<code>READ UNCOMMITTED</code>	Is automatically converted to <code>READ COMMITTED</code> by Vertica.
<code>READ WRITE</code> <code>READ ONLY</code>	Determines whether the transaction is read/write or read-only. Read/write is the default. When a transaction is read-only, the following SQL commands are disallowed: <ul style="list-style-type: none"> <li>▪ <code>INSERT</code>, <code>UPDATE</code>, <code>DELETE</code>, and <code>COPY</code> if the table they would write to is not a temporary table</li> <li>▪ All <code>CREATE</code>, <code>ALTER</code>, and <code>DROP</code> commands</li> <li>▪ <code>GRANT</code>, <code>REVOKE</code>, and <code>EXPLAIN</code> if the command it would run is among those listed.</li> </ul> This is a high-level notion of read-only that does not prevent all writes to disk.

### READ COMMITTED vs. Snapshot Isolation

By itself, `AT EPOCH LATEST` produces purely historical query behavior. However, with `READ COMMITTED`, `SELECT` queries return the same result set as `AT EPOCH LATEST` plus any changes made by the current transaction.

This is standard ANSI SQL semantics for ACID transactions. Any select query within a transaction sees the transactions's own changes regardless of isolation level.

### Notes

- SERIALIZABLE isolation does not apply to temporary tables, which are isolated by their transaction scope.
- Applications using SERIALIZABLE must be prepared to retry transactions due to serialization failures.

### Example

```
=> SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET
```

## SET SESSION MEMORYCAP

Specifies a limit on the amount of memory that any request issued by the session can consume.

### Syntax

```
SET SESSION MEMORYCAP 'memory-limit' | = default
```

### Parameters

<code><i>memory-limit</i>   = <i>default</i></code>	<p>The maximum amount of memory the session can use. To set a value, supply number followed by a unit. Units can be one of the following:</p> <ul style="list-style-type: none"> <li>▪ % percentage of total memory available to the Resource Manager. (In this case, size must be 0-100).</li> <li>▪ K Kilobytes</li> <li>▪ M Megabytes</li> <li>▪ G Gigabytes</li> <li>▪ T Terabytes</li> </ul> <p>If you use the value = default the session's MEMORYCAP is set to the user's MEMORYCAP value.</p>
---	---

### Notes

- The MEMORYCAP limit is per user session, not an overall cap on the user's memory usage. A user could spawn multiple sessions, each of which could use up to the limit set by the MEMORYCAP.
- This command requires superuser privileges if the MEMORYCAP is being increased over the user's MEMORYCAP limit (see **CREATE USER** (page 695) for details).
- Non-superusers can change this value to anything below or equal to their MEMORYCAP limit.

### Example

The following command sets a memorycap of 4 gigabytes on the session:

```
=> SET SESSION MEMORYCAP '4G';
```

To return the memorycap to the previous setting:

```
=> SET SESSION MEMORYCAP NONE;
=> SHOW MEMORYCAP;
  name      | setting
-----+-----
memorycap  | UNLIMITED
(1 row)
```

**See Also****ALTER RESOURCE POOL** (page 582)**CREATE RESOURCE POOL** (page 649)**CREATE USER** (page 695)**DROP RESOURCE POOL** (page 705)**SET SESSION RESOURCE POOL** (page 778)

Managing Workloads in the Administrator's Guide

## SET SESSION RESOURCE POOL

Associates the user session with the specified resource pool.

**Syntax**

```
SET SESSION RESOURCE POOL pool-name | = default
```

**Parameters**

<i>pool-name</i>   = default	Specifies the name of the resource pool to be associated with session. If you use the value = default, then the session's resource pool is set to the default resource pool for the user.
------------------------------	---

**Notes**

- The pool must have been created beforehand.
- This command requires non-superusers to have usage privileges for the resource pool.
- Superusers can assign their session to any resource pool they want.

**See Also****ALTER RESOURCE POOL** (page 582)**CREATE RESOURCE POOL** (page 649)**CREATE USER** (page 695)**DROP RESOURCE POOL** (page 705)**GRANT (Resource Pool)** (page 722)**SET SESSION MEMORYCAP** (page 777)

## Managing Workloads in the Administrator's Guide

## SET SESSION RUNTIMECAP

Sets the maximum amount of time a session's query can run.

### Syntax

```
SET SESSION RUNTIMECAP [ 'duration' | NONE | = default ]
```

### Parameters

'duration'   NONE   DEFAULT	<p>One of three values:</p> <ul style="list-style-type: none"> <li>▪ An interval such as '1 minute' or '100 seconds' (see <i>Interval Values</i> (page 30) for a full explanation) setting the maximum amount of time this session's queries should be allowed to run.</li> <li>▪ NONE which eliminates any limit on the amount of time the session's queries can run (the default value).</li> <li>▪ = default which sets the session's RUNTIMECAP to the user's RUNTIMECAP value.</li> </ul>
-----------------------------	--

### Notes

- The largest allowable RUNTIMECAP value is 1 year (365 days).
- This command requires superuser privileges if the RUNTIMECAP is being increased over the user's RUNTIMECAP limit.
- Normal users can change the RUNTIMECAP of their own sessions to any value below their own RUNTIMECAP. They cannot increase the RUNTIMECAP beyond any limit set for them by the superuser.
- The timeout is not precise, so a query may run a little longer than the value set in RUNTIMECAP.
- Queries that violate the RUNTIMECAP are terminated with the error message `Execution time exceeded run time cap.`

### Example

The following command sets the session's runtimecap to 10 minutes:

```
=> SET SESSION RUNTIMECAP '10 minutes';
```

To return the RUNTIMECAP to the user's default setting:

```
=> SET SESSION RUNTIMECAP DEFAULT;
SET
=> SHOW RUNTIMECAP;
   name      | setting
-----+-----
 runtimecap | UNLIMITED
(1 row)
```

**See Also****CREATE USER** (page 695)**ALTER USER** (page 594)

Managing Workloads in the Administrator's Guide

## SET SESSION TEMPSPACECAP

Sets the maximum amount of temporary file storage space that any request issued by the session can consume.

**Syntax**

```
SET SESSION TEMPSPACECAP 'space-limit' | = default | NONE
```

**Parameters**

<i>'space-limit'</i>	<p>The maximum amount of temporary file space the session can use. To set a limit, use a numeric value followed by a unit (for example: '10G'). The unit can be one of the following:</p> <ul style="list-style-type: none"><li>▪ % percentage of total temporary storage space available. (In this case, the numeric value must be 0-100).</li><li>▪ K Kilobytes</li><li>▪ M Megabytes</li><li>▪ G Gigabytes</li><li>▪ T Terabytes</li></ul> <p>Setting this value to <code>= default</code> sets the session's TEMPSPACECAP to the user's TEMPSPACECAP value.</p> <p>Setting this value to <code>NONE</code> results in the session having unlimited temporary storage space. This is the default value.</p>
----------------------	--

**Notes**

- This limit is per session, not per user. A user could open multiple sessions, each of which could use up to the TEMPSPACECAP.
- This command requires superuser privileges to increase the TEMPSPACECAP over the user's TEMPSPACECAP limit.
- Regular users can change the TEMPSPACECAP associated with their own sessions to any value less than or equal to their own TEMPSPACECAP. They cannot increase its value beyond their own TEMPSPACECAP value.
- Any execution plan that exceeds its TEMPSPACECAP usage results in the error:  
`ERROR: Exceeded temp space cap.`

**Example**

The following command sets a TEMPSPACECAP of 20gigabytes on the session:

```
=> SET SESSION TEMPSPACECAP '20G';
SET
=> SHOW TEMPSPACECAP;
   name      | setting
-----+-----
tempSPACEcap | 20971520
(1 row)
```

**Note:** SHOW displays the TEMPSPACECAP in kilobytes.

To return the memorycap to the previous setting:

```
=> SET SESSION TEMPSPACECAP NONE;
SET
=> SHOW TEMPSPACECAP;
   name      | setting
-----+-----
tempSPACEcap | UNLIMITED
(1 row)
```

### See Also

**ALTER USER** (page 594)

**CREATE USER** (page 695)

Managing Workloads in the Administrator's Guide

## SET STANDARD\_CONFORMING\_STRINGS

Treats backslashes as escape characters for the current session.

### Syntax

```
SET STANDARD_CONFORMING_STRINGS TO { ON | OFF }
```

### Parameters

ON	Makes ordinary string literals ('...') treat back slashes (\) literally. This means that back slashes are treated as string literals, not escape characters. (This is the default.)
OFF	Treats back slashes as escape characters.

### Notes

- This statement works under vsql only.
- When standard conforming strings are on, Vertica supports SQL-2008 string literals within Unicode escapes.
- Standard conforming strings must be ON to use Unicode-style string literals (U&' \nnnn').

**TIP:** To set conforming strings across all sessions (permanently), use the `StandardConformingStrings` as described in Internationalization Parameters in the Administrator's Guide.

## Examples

The following example shows how to turn off conforming strings for the session.

```
=> SET STANDARD_CONFORMING_STRINGS TO OFF;
```

The following command lets you verify the settings:

```
=> SHOW STANDARD_CONFORMING_STRINGS;
      name                | setting
-----+-----
standard_conforming_strings | off
(1 row)
```

The following example shows how to turn on conforming strings for the session.

```
=> SET STANDARD_CONFORMING_STRINGS TO ON;
```

## See Also

**ESCAPE\_STRING\_WARNING** (page 768)

## SET TIME ZONE

Changes the TIME\_ZONE run-time parameter for the current session.

### Syntax

```
SET TIME ZONE TO { value | 'value' }
```

### Parameters

<i>value</i>	<p>Is one of the following:</p> <ul style="list-style-type: none"> <li>▪ One of the time zone names specified in the tz database, as described in <b>Sources for Time Zone and Daylight Saving Time Data</b> <a href="http://www.twinsun.com/tz/tz-link.htm">http://www.twinsun.com/tz/tz-link.htm</a>. <b>Time Zone Names for Setting TIME_ZONE</b> (page 783) listed in the next section are for convenience only and could be out of date.</li> <li>▪ A signed integer representing an offset from UTC in hours</li> <li>▪ An <b>interval value</b> (page 30)</li> </ul>
--------------	---

### Notes

- TIME\_ZONE is a synonym for TIMEZONE. Both are allowed in Vertica syntax.
- The built-in constants LOCAL and DEFAULT, which set the time zone to the one specified in the TZ environment variable or, if TZ is undefined, from the operating system time zone. See Set the Default Time Zone and Using Time Zones with Vertica in the Installation Guide.
- When using a Country/City name, do not omit the country or the city. For example:  

```
SET TIME_ZONE TO 'Africa/Cairo'; -- valid
SET TIME_ZONE TO 'Cairo'; -- invalid
```
- Include the required keyword TO.
- Positive integer values represent an offset east from UTC.
- The **SHOW** (page 785) command displays the run-time parameters.

- If you set the timezone using POSIX format, the timezone abbreviation you use overrides the default timezone abbreviation. If the DATESTYLE is set to POSTGRES, the timezone abbreviation you use is also used when converting a timestamp to a string.

### Examples

```
=> SET TIME ZONE TO DEFAULT;
=> SET TIME ZONE TO 'PST8PDT'; -- Berkeley, California
=> SET TIME ZONE TO 'Europe/Rome'; -- Italy
=> SET TIME ZONE TO '-7'; -- UDT offset equivalent to PDT
=> SET TIME ZONE TO INTERVAL '-08:00 HOURS';
```

### See Also

Using Time Zones with Vertica in the Installation Guide

## Time Zone Names for Setting TIME\_ZONE

The following time zone names are recognized by Vertica as valid settings for the SQL time zone (the TIME\_ZONE run-time parameter).

**Note:** The names listed here are for convenience only and could be out of date. Refer to the **Sources for Time Zone and Daylight Saving Time Data** <http://www.twinsun.com/tz/tz-link.htm> page for precise information.

These names are not the same as the names shown in `/opt/<DBMS_LOWER_CASE>/share/timezonesets`, which are recognized by Vertica in date/time input values. The TIME\_ZONE names shown below imply a local daylight-savings time rule, where date/time input names represent a fixed offset from UTC.

In many cases there are several equivalent names for the same zone. These are listed on the same line. The table is primarily sorted by the name of the principal city of the zone.

In addition to the names listed in the table, Vertica accepts time zone names of the form *STDoffset* or *STDoffsetDST*, where *STD* is a zone abbreviation, *offset* is a numeric offset in hours west from UTC, and *DST* is an optional daylight-savings zone abbreviation, assumed to stand for one hour ahead of the given offset. For example, if *EST5EDT* were not already a recognized zone name, it would be accepted and would be functionally equivalent to USA East Coast time. When a daylight-savings zone name is present, it is assumed to be used according to USA time zone rules, so this feature is of limited use outside North America. Be wary that this provision can lead to silently accepting bogus input, since there is no check on the reasonableness of the zone abbreviations. For example, `SET TIME_ZONE TO FOOBANKO` works, leaving the system effectively using a rather peculiar abbreviation for GMT.

Time Zone
Africa
America
Antarctica

Asia
Atlantic
Australia
CET
EET
Etc/GMT
Europe
Factory
GMT GMT+0 GMT-0 GMT0 Greenwich Etc/GMT Etc/GMT+0 Etc/GMT-0 Etc/GMT0 Etc/Greenwich
Indian
MET
Pacific
UCT Etc/UCT
UTC Universal Zulu Etc/UTC Etc/Universal Etc/Zulu
WET

## SHOW

Displays run-time parameters for the current session.

### Syntax

```
SHOW { name | ALL }
```

### Parameters

<i>name</i>	<p>Is one of the following:</p> <ul style="list-style-type: none"> <li>▪ AVAILABLE_ROLES lists all roles available to the user.</li> <li>▪ DATESTYLE displays the current style of date values. See <b>SET DATESTYLE</b> (page 767).</li> <li>▪ ENABLED_ROLES displays the roles enabled for the current session. See <b>SET ROLE</b> (page 772).</li> <li>▪ ESCAPE_STRING_WARNING displays whether warnings are issued when backslash escapes are found in strings. See <b>SET ESCAPE_STRING_WARNING</b> (page 768).</li> <li>▪ INTERVALSTYLE which displays whether units are output when printing intervals. See <b>SET INTERVALSTYLE</b> (page 768).</li> <li>▪ LOCALE displays the current locale. See <b>SET LOCALE</b>.</li> <li>▪ MEMORYCAP displays the maximum amount of memory that any request use. See <b>SET MEMORYCAP</b> (page 777).</li> <li>▪ RESOURCE_POOL displays the resource pool that the session is using. See <b>SET RESOURCE POOL</b> (page 778).</li> <li>▪ RUNTIMECAP displays the maximum amount of time this session's queries can run. See <b>SET RUNTIMECAP</b> (page 779)</li> <li>▪ SEARCH_PATH displays the order in which schemas are searched. See <b>SET SEARCH_PATH</b> (page 774).</li> <li>▪ SESSION_CHARACTERISTICS displays the transaction characteristics. See <b>SET</b></li> </ul>
-------------	--

	<p><b>SESSION CHARACTERISTICS</b> (page 776).</p> <ul style="list-style-type: none"> <li>▪ <b>STANDARD_CONFORMING_STRINGS</b> displays whether backslash escapes are enabled for this session. See <b>SET STANDARD_CONFORMING_STRINGS</b> (page 781)</li> <li>▪ <b>TEMPSPACECAP</b> displays the maximum amount of temporary file space this session's queries can use. See <b>SET TEMPSPACECAP</b> (page 780).</li> <li>▪ <b>TIMEZONE</b> displays the timezone set in the current session. See <b>SET TIMEZONE</b> (page 782).</li> </ul>
ALL	Shows all run-time parameters.

## Examples

The following command returns all the run-time parameter settings:

```
=> SHOW ALL;
```

name	setting
locale	en_US@collation=binary (LEN_KBINARY)
standard_conforming_strings	on
escape_string_warning	on
datestyle	ISO, MDY
intervalstyle	plain
timezone	US/Eastern
search_path	"\$user", public, v_catalog, v_monitor, v_internal
transaction_isolation	READ COMMITTED
transaction_read_only	false
resource_pool	general
memorycap	UNLIMITED
tempstoragecap	UNLIMITED
runtimecap	UNLIMITED
enabled roles	
available roles	applogs, appadmin

(15 rows)

The following command returns the search path settings:

```
=> SHOW SEARCH_PATH;
```

name	setting
search_path	"\$user", public, v_catalog, v_monitor, v_internal

(1 row)

The following commands shows the session transaction isolation level:

```
=> SHOW TRANSACTION ISOLATION LEVEL;
```

name	setting
transaction_isolation	READ COMMITTED

(1 row)

The next command returns the `read` setting for `SESSION CHARACTERISTICS AS TRANSACTION`. False indicates that the default read/write is the current setting:

```
=> SHOW transaction_read_only;
      name          | setting
-----+-----
 transaction_read_only | false
(1 row)
```

To change to read only, you'd need to enter:

```
=> SET SESSION CHARACTERISTICS AS TRANSACTION READ ONLY;
```

Now the same `SHOW` command returns true:

```
=> SHOW transaction_read_only;
      name          | setting
-----+-----
 transaction_read_only | true
(1 row)
```

## TRUNCATE TABLE

Removes all storage associated with a table, while preserving the table definitions. `TRUNCATE TABLE` auto-commits the current transaction after statement execution and cannot be rolled back.

### Syntax

```
TRUNCATE TABLE [schema_name.] table
```

### Parameters

<i>[schema_name.]</i>	Specifies the name of an optional schema.
<i>table</i>	Specifies the name of a base table or temporary table.

### Notes

- The superuser, database owner, and table owner can truncate a table.
- The schema owner can drop a table but cannot truncate a table.
- `TRUNCATE TABLE` is useful for testing; you can remove all table data without having to recreate projections when you reload table data.
- `TRUNCATE TABLE` commits the entire transaction, even if the `TRUNCATE` statement fails.
- If the truncated table is a large single (fact) table that contains pre-join projections, the projections show 0 rows after the transaction completes and are ready for data reload.
- If the truncated table is a dimension table, the system returns the following error:  
Cannot truncate a dimension table with pre-joined projections  
Drop the pre-join projection first, and then issue the `TRUNCATE` command.

- If the truncated table has out-of-date projections, those projections are cleared and marked up-to-date after the truncation operation completes.
- TRUNCATE TABLE takes an O (Owner) lock on the table until the truncation process completes, when savepoint is then released.
- To truncate an ON COMMIT DELETE ROWS temporary table without ending the transaction, use **DELETE FROM temp\_table** (page 699) syntax.

**Note:** The effect of DELETE FROM depends on the table type. If the table is specified as ON COMMIT DELETE ROWS, then DELETE FROM works like TRUNCATE TABLE; otherwise it behaves like a normal delete in that it does not truncate the table.

- After truncate operations complete, the data recovers from that current epoch onward. Because TRUNCATE TABLE removes table history, AT EPOCH queries return nothing. TRUNCATE TABLE behaves the same when you have data in WOS, ROS, or both, as well as for unsegmented/segmented projections.

### See Also

**DELETE** (page 699), **DROP TABLE** (page 709), and **LOCKS** (page 858)

Transactions in the Concepts Guide

Deleting Data and Best Practices for DELETE and UPDATE in the Administrator's Guide

## UNION

Combines the results of two or more select statements.

### Syntax

```
SELECT
... UNION [ ALL ] select
... [ UNION [ ALL ] select ]...
... [ ORDER BY { column-name
... | ordinal-number }
... [ ASC | DESC ] [ , ... ] ]
... [ LIMIT { integer | ALL } ]
... [ OFFSET integer ]
```

**Note:** SELECT statements can contain ORDER BY, LIMIT or OFFSET clauses if the statement is enclosed within parentheses.

### Notes

The results of several SELECT statements can be combined into a larger result using UNION. Each SELECT statement produces results in which the UNION combines all those results into a final single result. Specifically, a row in the results of a UNION operation must have existed in the results from one of the SELECT statements. Each SELECT statement must have the same number of items in the select list as well as compatible data types. If the data types are incompatible, Vertica returns an error.

The results of a UNION contain only distinct rows. so use UNION ALL to keep duplicate rows. UNION pays the performance price of eliminating duplicates; therefore, unless duplicate rows are not wanted, use UNION ALL for its performance benefits.

A SELECT statement containing ORDER BY, LIMIT, or OFFSET clauses must be enclosed in parentheses . If the statement is not enclosed in parentheses an error is returned. However, the rightmost ORDER BY, LIMIT, or OFFSET clause in the UNION query does not need to be enclosed in parentheses to the rightmost query. This indicates to perform these operations on results of the UNION operation. GROUP BY and HAVING operations cannot be applied to the results.

The ordering of the results of a UNION operation does not necessarily depend on the ordering of the results for each SELECT statement. The resulting rows can be ordered by adding an ORDER BY to the UNION operation, as in the syntax above. If ORDER BY is used, only integers and column names from the first (leftmost) SELECT statement are allowed in the order by list. The integers specify the position of the columns on which to sort. The column names displayed in the results are the same column names that display for the first (leftmost) select statement.

UNION correlated and noncorrelated subquery predicates are also supported.

```
=> SELECT * FROM T1
      WHERE T1.x IN
            (SELECT MAX(c1) FROM T2
             UNION ALL
              SELECT MAX(cc1) FROM T3
             UNION ALL
              SELECT MAX(d1) FROM T4);
```

## Examples

Consider the following two tables:

### Company\_A

Id	emp_lname	dept	sales
1234	Vincent	auto parts	1000
5678	Butch	auto parts	2500
9012	Marcellus	floral	500

### Company B

Id	emp_lname	dept	sales
4321	Marvin	home goods	250
9012	Marcellus	home goods	500
8765	Zed	electronics	20000

The following query lists all **distinct** IDs and surnames of employees:

```
=> SELECT id, emp_lname
      FROM company_A
      UNION
      SELECT id, emp_lname
      FROM company_B;
```

```
id    | emp_lname
```

```
-----+-----
1234 | Vincent
4321 | Marvin
5678 | Butch
8765 | Zed
9012 | Marcellus
(5 rows)
```

The following query lists *all* IDs and surnames of employees:

```
=> SELECT id, emp_lname
     FROM company_A
     UNION ALL
     SELECT id, emp_lname
     FROM company_B;
```

```
id | emp_lname
-----+-----
1234 | Vincent
5678 | Butch
9012 | Marcellus
4321 | Marvin
8765 | Zed
9012 | Marcellus
(6 rows)
```

The next example returns the top two performing salespeople in each company combined:

```
=> (SELECT id, emp_lname, sales
     FROM company_A
     ORDER BY sales
     LIMIT 2)
     UNION ALL
     (SELECT id, emp_lname, sales
     FROM company_B
     ORDER BY sales
     LIMIT 2);
```

```
id | emp_lname | sales
-----+-----+-----
4321 | Marvin      | 250
9012 | Marcellus | 500
9012 | Marcellus | 500
1234 | Vincent   | 1000
(4 rows)
```

In this example, return all employee orders by sales. Note that the ORDER BY clause is applied to the entire result:

```
=> SELECT id, emp_lname, sales
     FROM company_A
     UNION
     SELECT id, emp_lname, sales
     FROM company_B
     ORDER BY sales;
```

```
id | emp_lname | sales
```

```

-----+-----+-----
4321 | Marvin      | 250
9012 | Marcellus | 500
1234 | Vincent   | 1000
5678 | Butch     | 2500
8765 | Zed       | 20000
(5 rows)

```

And now sum the sales for each company, ordered by sales in descending order, and grouped by department:

```

=> (SELECT 'company a' as company, dept, SUM(sales)
    FROM company_a
    GROUP BY dept
    ORDER by 2 DESC)
    UNION
    (SELECT 'company b' as company, dept, SUM(sales)
    FROM company_b
    GROUP BY dept
    ORDER by 2 DESC)
    ORDER BY 1;

```

```

company | dept          | sum
-----+-----+-----
company a | auto parts   | 3500
company a | floral       | 500
company b | electronics  | 20000
company b | home goods   | 750
(4 rows)

```

The final query shows the results of a mismatched data types:

```

=> SELECT id, emp_lname
    FROM company_a
    UNION
    SELECT emp_lname, id
    FROM company_b;
ERROR: UNION types int8 and character varying cannot be matched

```

### See Also

***SELECT*** (page 747)

Subqueries and UNION in Subqueries in the Programmer's Guide

## UPDATE

Replaces the values of the specified columns in all rows for which a specific condition is true. All other columns and rows in the table are unchanged. By default UPDATE uses the WOS and if the WOS fills up, overflows to the ROS.

### Syntax

```
UPDATE [ /*+ direct */ ] [schemaname.]table SET column =
... { expression | DEFAULT } [ , ... ]
... [ FROM from-list ]
... [ WHERE clause (on page 752) ]
```

### Parameters

<code>/*+ direct */</code>	Writes the data directly to disk (ROS) bypassing memory (WOS). <b>Note:</b> If you update using the <code>direct</code> hint, you still need to issue a <code>COMMIT</code> or <code>ROLLBACK</code> command to finish the transaction.
<code>[schemaname.]table</code>	Specifies the name of a table in the schema. When using more than one schema, specify the schema that contains the table. You cannot update a projection.
<code>column</code>	Specifies the name of a non-key column in the table.
<code>expression</code>	Specifies a value to assign to the column. The expression can use the current values of this and other columns in the table. For example: <code>UPDATE T1 SET C1 = C1+1;</code>
<code>from-list</code>	A list of table expressions, allowing columns from other tables to appear in the <code>WHERE</code> condition and the <code>UPDATE</code> expressions. This is similar to the list of tables that can be specified in the <code>FROM</code> (see "FROM Clause" on page 750) clause of a <code>SELECT</code> command. Note that the target table must not appear in the <code>from-list</code> .

### Notes

- Subqueries and joins are permitted in UPDATE statements, which is useful for updating values in a table based on values that are stored in other tables. UPDATE changes the values of the specified columns in all rows that satisfy the condition. Only the columns to be modified need to be specified in the SET clause. Columns that are not explicitly modified retain their previous values. On successful completion, an UPDATE operation returns a count, which represents the number of rows updated. A count of 0 is not an error; it means that no rows matched the condition.

See Subqueries in UPDATE and DELETE Statements in the Programmer's Guide.

- The table specified in the UPDATE list cannot also appear in the from-list (no self joins); for example, the following is not allowed:

```
=> BEGIN;
=> UPDATE result_table
   SET address='new' || r2.address
```

```
FROM result_table r2
WHERE r2.cust_id = result_table.cust_id + 10;
ERROR: Self joins in UPDATE statements are not allowed
DETAIL: Target relation result_table also appears in the FROM list
```

- If the joins specified in the *WHERE* predicate produce more than one copy of the row in the table to be updated, the new value of the row in the table is chosen arbitrarily.
- *UPDATE* inserts new records into the *WOS* and marks the old records for deletion.
- You cannot *UPDATE* columns that have primary key or foreign key referential integrity constraints.
- To use the ***DELETE*** (page 699) or ***UPDATE*** (page 792) commands with a ***WHERE clause*** (page 752), you must have both *SELECT* and *DELETE* privileges on the table.

### Examples

```
=> UPDATE FACT SET PRICE = PRICE - COST * 80 WHERE COST > 100;
=> UPDATE Retail.CUSTOMER SET STATE = 'NH' WHERE CID > 100;
```

For examples on how to nest a subquery within an *UPDATE* statement, see Subqueries in *UPDATE* and *DELETE* in the Programmer's Guide.

# SQL System Tables (Monitoring APIs)

---

Vertica provides system tables that let you monitor the health of your database. These tables can be queried the same way you perform query operations on base or temporary tables using `SELECT`. Queries against system tables may use expressions, predicates, aggregates, analytics, subqueries, joins, and historical query syntax. It is also possible to save the results of a system table query into a user table for future analysis using, for example, `INSERT INTO <user_table> SELECT * FROM SYSTEM_TABLES;`

System tables are grouped into the following schemas:

- `V_CATALOG` (page 795) – information about persistent objects in the catalog
- `V_MONITOR` (page 828) – information about transient system state

These schemas reside in the default search path so there is no need to specify `schema.table` in your queries unless you change the search path to exclude `V_MONITOR` or `V_CATALOG` or both.

## Notes and Restrictions

- You can use external monitoring tools or scripts to query the system tables and act upon the information, as necessary. For example, when a host failure causes the K-safety level to fall below a desired level, the tool or script can notify the database administrator and/or appropriate IT personnel of the change, typically in the form of an e-mail.

**Note:** When a cluster is in a recovering state, the database refuses connection requests and cannot be monitored using the SQL monitoring API.

- To view all of the system tables issue the following command:  
`=> SELECT * FROM system_tables;`
- DDL and DML operations are not supported on system tables.
- With the exception of the `PROJECTION_REFRESHES` (page 863) table, system tables do not hold historical data.
- Vertica reserves some memory to help monitor busy systems. Using simple system table queries makes it easier to troubleshoot issues. See also `sysquery` and `sysdata` pools under **Built-in pools** (page 653) topic in SQL Reference Manual.
- In `V_CATALOG.TABLES`, columns `TABLE_SCHEMA` and `TABLE_NAME` are case sensitive when equality (`=`) predicates are used in queries. For example, given the following schema:

```
=> CREATE SCHEMA SS;
=> CREATE TABLE SS.TT (c1 int);
=> INSERT INTO ss.tt VALUES (1);
```

If you run a query using the `=` predicate, Vertica returns 0 rows:

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE
       table_schema = 'ss';
 table_schema | table_name
-----+-----
(0 rows)
```

Use the case-insensitive `ILIKE` predicate to return the expected results:

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE
       table_schema ILIKE 'ss';
       table_schema | table_name
-----+-----
       SS           | TT
(1 row)
```

### Querying case-sensitive data in system tables

The `V_CATALOG.TABLES` (page 820).`TABLE_SCHEMA` and `TABLE_NAME` columns are case sensitive when used with an equality (`=`) predicate in queries. For example, given the following sample schema, if you execute a query using the `=` predicate, Vertica returns 0 rows:

```
=> CREATE SCHEMA SS;
=> CREATE TABLE SS.TT (c1 int);
=> INSERT INTO ss.tt VALUES (1);
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema = 'ss';
```

```
       table_schema | table_name
-----+-----
(0 rows)
```

**TIP:** Use the case-insensitive **ILIKE** predicate to return the expected results.

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema ILIKE
       'ss';
       table_schema | table_name
-----+-----
       SS           | TT
(1 row)
```

### See Also

Using the SQL Monitoring API in the Administrator's Guide

## V\_CATALOG Schema

The system tables in this section reside in the `v_catalog` schema. These tables provide information (metadata) about the objects in a database; for example, tables, constraints, users, projections, and so on.

### COLUMNS

Provides table column information.

Column Name	Data Type	Description
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the table.
TABLE_SCHEMA	VARCHAR	The schema name for which information is listed.
TABLE_NAME	VARCHAR	The table name for which information is listed.
IS_SYSTEM_TABLE	BOOLEAN	Indicates whether the table is a system table, where <i>t</i> is

		true and <i>f</i> is false.
COLUMN_NAME	VARCHAR	The column name for which information is listed in the database.
DATA_TYPE	VARCHAR	The data type assigned to the column; for example VARCHAR.
DATA_TYPE_DESCRIPTION	VARCHAR	The description of the data type; for example VARCHAR(16).
DATA_TYPE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the data type.
DATA_TYPE_LENGTH	INTEGER	The maximum allowable length of the data type.
CHARACTER_MAXIMUM_LENGTH	VARCHAR	The maximum allowable length of the column.
NUMERIC_PRECISION	INTEGER	The number of significant decimal digits.
NUMERIC_SCALE	INTEGER	The number of fractional digits.
DATETIME_PRECISION	INTEGER	For TIMESTAMP data type, returns the declared precision; returns null if no precision was declared.
INTERVAL_PRECISION	INTEGER	The number of fractional digits retained in the seconds field.
ORDINAL_POSITION	VARCHAR	The position of the column relative to other columns in the table.
IS_NULLABLE	BOOLEAN	Indicates whether the column can contain null values, where <i>t</i> is true and <i>f</i> is false.
COLUMN_DEFAULT	VARCHAR	The default value of a column, such as empty or expression.

**Example**

```
=> SELECT table_schema, table_name, column_name, data_type, is_nullable
      FROM columns WHERE table_schema = 'store' AND data_type = 'Date';
 table_schema | table_name | column_name | data_type | is_nullable
```

```
-----+-----+-----+-----+-----
store      | store_dimension | first_open_date | Date | f
store      | store_dimension | last_remodel_date | Date | f
store      | store_orders_fact | date_ordered | Date | f
store      | store_orders_fact | date_shipped | Date | f
store      | store_orders_fact | expected_delivery_date | Date | f
store      | store_orders_fact | date_delivered | Date | f
6 rows)
```

NULL results indicate that those columns were not defined. For example, given the following table, the result for the Datetime\_precision column is NULL because no precision was declared:

```
=> CREATE TABLE c (c TIMESTAMP);
CREATE TABLE
```

```
=> SELECT table_name, column_name, datetime_precision FROM columns WHERE
table_name = 'c';
 table_name | column_name | datetime_precision
-----+-----+-----
 c          | c           |
(1 row)
```

In this example, the `datetime_precision` column returns 4 because the precision was declared as 4 in the `CREATE TABLE` statement:

```
=> DROP TABLE c;
=> CREATE TABLE c (c TIMESTAMP(4));
CREATE TABLE
=> SELECT table_name, column_name, datetime_precision FROM columns WHERE
table_name = 'c';
 table_name | column_name | datetime_precision
-----+-----+-----
 c          | c           | 4
```

## COMMENTS

Returns information about comments associated with objects in the database.

Column Name	Data Type	Description
COMMENT_ID	INTEGER	The comment's internal ID number
OBJECT_ID	INTEGER	The internal ID number of the object associated with the comment
OBJECT_TYPE	VARCHAR	The type of object associated with the comment. Possible values are: <ul style="list-style-type: none"> <li>▪ COLUMN</li> <li>▪ CONSTRAINT</li> <li>▪ FUNCTION</li> <li>▪ LIBRARY</li> <li>▪ NODE</li> <li>▪ PROJECTION</li> <li>▪ SCHEMA</li> <li>▪ SEQUENCE</li> <li>▪ TABLE</li> <li>▪ VIEW</li> </ul>
OBJECT_SCHEMA	VARCHAR	The schema containing the object.
OBJECT_NAME	VARCHAR	The name of the object associated with the comment.
OWNER_ID	VARCHAR	The internal ID of the owner of the object.
OWNER_NAME	VARCHAR	The object owner's name.
CREATION_TIME	TIMESTAMPTZ	When the comment was created.
LAST_MODIFIED_TIME	TIMESTAMPTZ	When the comment was last modified.

COMMENT	VARCHAR	The text of the comments.
---------	---------	---------------------------

## DATABASES

Provides information about the databases in this Vertica installation.

Column Name	Data Type	Description
DATABASE_ID	INTEGER	The database's internal ID number
DATABASE_NAME	VARCHAR	The database's name
OWNER_ID	INTEGER	The database owner's ID
OWNER_NAME	INTEGER	The database owner's name
START_TIME	TIMESTAMPZ	The date and time the database last started
COMPLIANCE_MESSAGE	VARCHAR	Message describing the current state of the database's license compliance.

## DUAL

DUAL is a single-column "dummy" table with one record whose value is X; for example:

```
=> SELECT * FROM DUAL;
  dummy
-----
  X
(1 row)
```

You can now write the following types of queries:

```
mydb=> SELECT 1 FROM dual;
  ?column?
-----
  1
(1 row)

=> SELECT current_timestamp, current_user FROM dual;
  ?column? | current_user
-----+-----
2010-03-08 12:57:32.065841-05 | release
(1 row)

mydb=> CREATE TABLE t1(col1 VARCHAR(20), col2 VARCHAR(2));
mydb=> INSERT INTO T1(SELECT 'hello' AS col1, 1 AS col2 FROM dual);)
mydb=> SELECT * FROM t1;
  col1 | col2
-----+-----
hello | 1
(1 row)
```

Because DUAL is a system table, you cannot create projections for it, nor can you use it in pre-join projections for normal tables. For example, the following is not permitted:

```
=> CREATE TABLE foo (a varchar(20), b varchar(2));
=> CREATE PROJECTION t1_prejoin AS SELECT * FROM t1 JOIN dual ON t1.col1 =
dual.dummy;
      ERROR: Virtual tables are not allowed in FROM clause of projection
```

The following is also not permitted:

**Note:** Only the rows where the `PROJECTION_REFRESHES.IS_EXECUTING` column equals `false` are cleared.

```
=> CREATE PROJECTION dual_proj AS SELECT * FROM dual;
      ERROR: Virtual tables are not allowed in FROM clause of projection
```

## FOREIGN\_KEYS

Provides foreign key information.

Column Name	Data Type	Description
CONSTRAINT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the constraint.
CONSTRAINT_NAME	VARCHAR	The constraint name for which information is listed.
COLUMN_NAME	VARCHAR	The name of the column that is constrained.
ORDINAL_POSITION	VARCHAR	The position of the column relative to other columns in the table.
TABLE_NAME	VARCHAR	The table name for which information is listed.
REFERENCE_TABLE_NAME	VARCHAR	References the TABLE_NAME column in the PRIMARY_KEY table.
CONSTRAINT_TYPE	VARCHAR	The constraint type, f, for foreign key.
REFERENCE_COLUMN_NAME	VARCHAR	References the COLUMN_NAME column in the PRIMARY_KEY table.
TABLE_SCHEMA	VARCHAR	The schema name for which information is listed.
REFERENCE_TABLE_SCHEMA	VARCHAR	References the TABLE_SCHEMA column in the PRIMARY_KEY table.

### Example

```
mydb=> SELECT constraint_name, table_name, ordinal_position, reference_table_name
      FROM foreign_keys ORDER BY 3;
      constraint_name | table_name | ordinal_position | reference_table_name
-----+-----+-----+-----
fk_store_sales_date | store_sales_fact | 1 | date_dimension
fk_online_sales_saledate | online_sales_fact | 1 | date_dimension
fk_store_orders_product | store_orders_fact | 1 | product_dimension
fk_inventory_date | inventory_fact | 1 | date_dimension
fk_inventory_product | inventory_fact | 2 | product_dimension
fk_store_sales_product | store_sales_fact | 2 | product_dimension
fk_online_sales_shipdate | online_sales_fact | 2 | date_dimension
```

```

fk_store_orders_product | store_orders_fact | 2 | product_dimension
fk_inventory_product    | inventory_fact     | 3 | product_dimension
fk_store_sales_product  | store_sales_fact   | 3 | product_dimension
fk_online_sales_product | online_sales_fact  | 3 | product_dimension
fk_store_orders_store   | store_orders_fact  | 3 | store_dimension
fk_online_sales_product | online_sales_fact  | 4 | product_dimension
fk_inventory_warehouse  | inventory_fact     | 4 | warehouse_dimension
fk_store_orders_vendor  | store_orders_fact  | 4 | vendor_dimension
fk_store_sales_store    | store_sales_fact   | 4 | store_dimension
fk_store_orders_employee | store_orders_fact | 5 | employee_dimension
fk_store_sales_promotion | store_sales_fact   | 5 | promotion_dimension
fk_online_sales_customer | online_sales_fact  | 5 | customer_dimension
fk_store_sales_customer | store_sales_fact   | 6 | customer_dimension
fk_online_sales_cc      | online_sales_fact  | 6 | call_center_dimension
fk_store_sales_employee | store_sales_fact   | 7 | employee_dimension
fk_online_sales_op      | online_sales_fact  | 7 | online_page_dimension
fk_online_sales_shipping | online_sales_fact  | 8 | shipping_dimension
fk_online_sales_warehouse | online_sales_fact | 9 | warehouse_dimension
fk_online_sales_promotion | online_sales_fact | 10 | promotion_dimension
(26 rows)

```

## GRANTS

Provides information about privileges granted on various objects, the granting user and grantee user. The order of columns in the table corresponds to the order in which they appear in the GRANT command.

Column Name	Data Type	Description
GRANTOR	VARCHAR	The user granting permission.
PRIVILEGES_DESCRIPTION	VARCHAR	A readable description of the privileges being granted; for example INSERT, SELECT.
OBJECT_SCHEMA	VARCHAR	The name of the schema that is being granted privileges.
OBJECT_NAME	VARCHAR	The name of the object that is being granted privileges. Note that for schema privileges, the schemaname appears in the OBJECT_NAME column rather than the OBJECT_SCHEMA column.
GRANTEE	VARCHAR	The user being granted permission.

### Notes

The vsql commands \dp and \z both include the schema name in the output:

```

=> \dp
      Access privileges for database "vmartdb"
  Grantee | Grantor | Privileges | Schema |      Name
-----+-----+-----+-----+-----
          | release | USAGE     |        | public
          | release | USAGE     |        | v_internal
          | release | USAGE     |        | v_catalog
          | release | USAGE     |        | v_monitor
          | release | USAGE     |        | v_internal
          | release | USAGE     |        | v_catalog
          | release | USAGE     |        | v_monitor
          | release | USAGE     |        | v_internal

```

```

      | release | USAGE      |      | designer_system
(9 rows)
=> \z

```

```

      Access privileges for database "vmartdb"
Grantee | Grantor | Privileges | Schema | Name
-----+-----+-----+-----+-----
      | release | USAGE      |      | public
      | release | USAGE      |      | v_internal
      | release | USAGE      |      | v_catalog
      | release | USAGE      |      | v_monitor
      | release | USAGE      |      | v_internal
      | release | USAGE      |      | v_catalog
      | release | USAGE      |      | v_monitor
      | release | USAGE      |      | v_internal
      | release | USAGE      |      | designer_system
(9 rows)

```

The `vsql` command `\dp *.tablename` displays table names in all schemas. This command lets you distinguish the grants for same-named tables in different schemas:

```

=> \dp *.events;
      Access privileges for database "dbadmin"
Grantee | Grantor | Privileges | Schema | Name
-----+-----+-----+-----+-----
user2   | dbadmin | INSERT, SELECT, UPDATE, DELETE, REFERENCES | schema1 | events
user1   | dbadmin | SELECT      | schema1 | events
user2   | dbadmin | INSERT, SELECT, UPDATE, DELETE, REFERENCES | schema2 | events
user1   | dbadmin | INSERT, SELECT | schema2 | events
(4 rows)

```

The `vsql` command `\dp schemaname.*` displays all tables in the named schema:

```

=> \dp schema1.*
      Access privileges for database "dbadmin"
grantee | grantor | privileges_description | table_schema | table_name
-----+-----+-----+-----+-----
user2   | dbadmin | INSERT, SELECT, UPDATE, DELETE, REFERENCES | schema1      | events
user1   | dbadmin | SELECT                  | schema1      | events
(2 rows)

```

## Example

In the following example, `online_sales` is the schema that first gets privileges, and then inside that schema the anchor table gets `SELECT` privileges:

```

=> SELECT grantee, grantor, privileges_description, object_schema, object_name
      FROM grants WHERE grantee='u1' ORDER BY object_name;
grantee | grantor | privileges_description | object_schema | object_name
-----+-----+-----+-----+-----
u1      | release | CREATE                |               | online_sales
u1      | release | SELECT                 | online_sales  | online_sales_fact

```

## LICENSE\_AUDITS

Lists the results of Vertica's license automatic compliance audits. See [How Vertica Calculates Database Size in the Administrator's Guide](#).

Column Name	Data Type	Description
DATABASE_SIZE_BYTES	INTEGER	The estimated raw data size of the database
LICENSE_SIZE_BYTES	INTEGER	The licensed data allowance
USAGE_PERCENT	FLOAT	Percentage of the licensed allowance used
AUDIT_START_TIMESTAMP	TIMESTAMP Z	When the audit started
AUDIT_END_TIMESTAMP	TIMESTAMP Z	When the audit finished
CONFIDENCE_LEVEL_PERCENT	FLOAT	The confidence level of the size estimate
ERROR_TOLERANCE_PERCENT	FLOAT	The error tolerance used for the size estimate
USED_SAMPLING	BOOLEAN	Whether data was randomly sampled (if false, all of the data was analyzed)
CONFIDENCE_INTERVAL_LOWER_BOUND_BYTES	INTEGER	The lower bound of the data size estimate within the confidence level
CONFIDENCE_INTERVAL_UPPER_BOUND_BYTES	INTEGER	The upper bound of the data size estimate within the confidence level
SAMPLE_COUNT	INTEGER	The number of data samples used to generate the estimate
CELL_COUNT	INTEGER	The number of cells in the database

## NODES

Lists the nodes in the database and their current state.

Column Name	Date Type	Description
NODE_NAME	VARCHAR	The name of the node
NODE_OID	INT	Unique identifier for the node
NODE_STATE	VARCHAR	The node's current state (UP, DOWN, RECOVERING, etc.)

### Example

```
=> SELECT * FROM V_MONITOR.NODES;
      node_name          |      node_oid          | node_state
-----+-----+-----
v_ExampleDB_node0001 | 45035996273704970 | UP
v_ExampleDB_node0002 | 45035996273714018 | DOWN
v_ExampleDB_node0003 | 45035996273714022 | UP
(3 rows)
```

## PASSWORDS

Contains user passwords information. This table stores not only current passwords, but also past passwords if any profiles have `PASSWORD_REUSE_TIME` or `PASSWORD_REUSE_MAX` parameters set. See **CREATE PROFILE** (page 637) for details.

Column Name	Data Type	Description
<code>USER_ID</code>	INTEGER	The ID of the user who owns the password.
<code>USER_NAME</code>	VARCHAR	The name of the user who owns the password.
<code>PASSWORD</code>	VARCHAR	The encrypted password.
<code>PASSWORD_CREATE_TIME</code>	DATE TIME	The date and time when the password was created.
<code>IS_CURRENT_PASSWORD</code>	BOOLEAN	Denotes whether this is the user's current password. Non-current passwords are retained to enforce password reuse limitations.
<code>PROFILE_ID</code>	INTEGER	The ID number of the profile to which the user is assigned.
<code>PROFILE_NAME</code>	VARCHAR	The name of the profile to which the user is assigned.
<code>PASSWORD_REUSE_MAX</code>	VARCHAR	The number password changes that must take place before an old password can be reused.
<code>PASSWORD_REUSE_TIME</code>	VARCHAR	The amount of time that must pass before an old password can be reused.

## PRIMARY\_KEYS

Provides primary key information.

Column Name	Data Type	Description
<code>CONSTRAINT_ID</code>	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the constraint.
<code>CONSTRAINT_NAME</code>	VARCHAR	The constraint name for which information is listed.
<code>COLUMN_NAME</code>	VARCHAR	The column name for which information is listed.
<code>ORDINAL_POSITION</code>	VARCHAR	The position of the column relative to other columns in the table.
<code>TABLE_NAME</code>	VARCHAR	The table name for which information is listed.
<code>CONSTRAINT_TYPE</code>	VARCHAR	The constraint type, <code>P</code> , for primary key.
<code>TABLE_SCHEMA</code>	VARCHAR	The schema name for which information is listed.

### Example

Request specific columns from the `PRIMARY_KEYS` table:

```
=> SELECT constraint_name, table_name, ordinal_position, table_schema
      FROM primary_keys ORDER BY 3;
```

constraint_name	table_name	ordinal_position	table_schema
C_PRIMARY	customer_dimension	1	public
C_PRIMARY	product_dimension	1	public
C_PRIMARY	store_dimension	1	store
C_PRIMARY	promotion_dimension	1	public
C_PRIMARY	date_dimension	1	public
C_PRIMARY	vendor_dimension	1	public
C_PRIMARY	employee_dimension	1	public
C_PRIMARY	shipping_dimension	1	public
C_PRIMARY	warehouse_dimension	1	public
C_PRIMARY	online_page_dimension	1	online_sales
C_PRIMARY	call_center_dimension	1	online_sales
C_PRIMARY	product_dimension	2	public

(12 rows)

## PROFILE PARAMETERS

Defines what information is stored in profiles.

Column Name	Data Type	Description
PROFILE_ID	INTEGER	The ID of the profile to which this parameter belongs.
PROFILE_NAME	VARCHAR	The name of the profile to which this parameter belongs.
PARAMETER_TYPE	VARCHAR	The policy type of this parameter (password_complexity, password_security, etc.)
PARAMETER_NAME	VARCHAR	The name of the parameter.
PARAMETER_LIMIT	VARCHAR	The parameter's value.

## PROFILES

Provides information about profiles.

Column Name	Data Type	Description
PROFILE_ID	INTEGER	The unique identifier for the profile.
PROFILE_NAME	VARCHAR	The profile's name.
PASSWORD_LIFE_TIME	VARCHAR	The number of days before the user's password expires. After expiration, the user is forced to change passwords during login or warned that their password has expired if password_grace_time is set to a value other than zero or unlimited.
PASSWORD_GRACE_TIME	VARCHAR	The number of days users are allowed to log in after their passwords expire. During the grace time, users are warned about their

		expired passwords when they log in. After the grace period, the user is forced to change passwords if he or she hasn't already.
PASSWORD_REUSE_MAX	VARCHAR	The number of password changes that must occur before the current password can be reused.
PASSWORD_REUSE_TIME	VARCHAR	The number of days that must pass after setting a password before it can be used again.
FAILED_LOGIN_ATTEMPTS	VARCHAR	The number of consecutive failed login attempts that triggers Vertica to lock the account.
PASSWORD_LOCK_TIME	VARCHAR	The number of days an account is locked after being locked due to too many failed login attempts.
PASSWORD_MAX_LENGTH	VARCHAR	The maximum number of characters allowed in a password.
PASSWORD_MIN_LENGTH	VARCHAR	The minimum number of characters required in a password.
PASSWORD_MIN_LETTERS	VARCHAR	The minimum number of letters (either uppercase or lowercase) required in a password.
PASSWORD_MIN_LOWERCASE_LETTERS	VARCHAR	The minimum number of lowercase.
PASSWORD_MIN_UPPERCASE_LETTERS	VARCHAR	The minimum number of uppercase letters required in a password.
PASSWORD_MIN_DIGITS	VARCHAR	The minimum number of digits required in a password.
PASSWORD_MIN_SYMBOLS	VARCHAR	The minimum of symbols (for example, !, #, \$, etc.) required in a password.

### Notes

Non-superusers querying this table see only the information for the profile to which they are assigned.

## PROJECTION\_COLUMNS

Provides information about projection columns, such as encoding type, sort order, type of statistics, and the time at which columns statistics were last updated.

Column Name	Data Type	Description
PROJECTION_NAME	VARCHAR	The projection name for which information is listed.
PROJECTION_COLUMN_NAME	VARCHAR	The projection column name.
COLUMN_POSITION	INTEGER	The ordinal position of a projection's column used in the <b>CREATE PROJECTION</b> (page 640) statement.
SORT_POSITION	INTEGER	The projection's column sort specification, as specified in CREATE PROJECTION ORDER BY clause. SORT_POSITION is NULL if the column is not included in the sort order.
COLUMN_ID	INTEGER	A unique numeric object ID (OID) assigned by the Vertica catalog, COLUMN_ID is the OID of the associated projection column object. This field is helpful as a key to other system tables.
DATA_TYPE	VARCHAR	Matches the corresponding table column data type (see <b>V_CATALOG.COLUMNS</b> (page 795)). DATA_TYPE is provided as a complement to ENCODING_TYPE.
ENCODING_TYPE	VARCHAR	The encoding type defined on the projection column.
ACCESS_RANK	INTEGER	The access rank of the projection column. See the ACCESSRANK parameter in the <b>CREATE PROJECTION</b> (page 640) statement for more information.
GROUP_ID	INTEGER	A unique numeric ID (OID) assigned by the Vertica catalog that identifies the group.
TABLE_SCHEMA	VARCHAR	The schema name in which the projection resides.
TABLE_NAME	VARCHAR	The table name that contains the projection.
TABLE_COLUMN_NAME	VARCHAR	The projection's corresponding table column name.
STATISTICS_TYPE	VARCHAR	The type of statistics the column has. Is one of: <ul style="list-style-type: none"> <li>▪ NONE—No statistics</li> <li>▪ ROWCOUNT—Created from existing catalog metadata, which Vertica automatically and periodically updates</li> <li>▪ FULL—Created by running <b>ANALYZE_STATISTICS()</b> (page 385)</li> </ul>
STATISTICS_UPDATED_TIMESTAMP	TIMESTAMP	The time at which the columns statistics were last updated.  Querying this column, along with STATISTICS_TYPE and PROJECTION_COLUMN_NAME, lets you quickly identify any projection columns whose statistics

		need updating. See also <b>V_CATALOG.ROJECTIONS.HAS_STATISTICS</b> (page 811).
--	--	--

**Example**

On a single-node cluster, the following sample schema defines a table named `trades`, which groups the highly-correlated columns `bid` and `ask` and stores the `stock` column separately:

```
=> CREATE TABLE trades (stock CHAR(5), bid INT, ask INT);
=> CREATE PROJECTION trades_p (stock ENCODING RLE, GROUPED(bid ENCODING
    DELTAVAL, ask)) AS (SELECT * FROM trades) ORDER BY stock, bid;
=> INSERT INTO trades VALUES('acme', 10, 20);
=> COMMIT;
```

Query the `PROJECTION_COLUMNS` table for table `trades`:

```
=> \x
Expanded display is on.
=> SELECT * FROM PROJECTION_COLUMNS WHERE table_name = 'trades';
```

Notice that the `statistics_type` column returns `NONE` for all three columns in the `trades` table. Also, there is no value in the `statistics_updated_timestamp` field because statistics have not yet been run on this table.

```
-[ RECORD 1 ]-----+-----
projection_name      | trades_p
projection_column_name | stock
column_position     | 0
sort_position       | 0
column_id           | 45035996273743508
data_type           | char(5)
encoding_type       | RLE
access_rank         | 0
group_id            | 0
table_schema        | public
table_name           | trades
table_column_name   | stock
statistics_type      | NONE
statistics_updated_timestamp |
-[ RECORD 2 ]-----+-----
projection_name      | trades_p
projection_column_name | bid
column_position     | 1
sort_position       | 1
column_id           | 45035996273743510
data_type           | int
encoding_type       | DELTAVAL
access_rank         | 0
group_id            | 45035996273743512
table_schema        | public
table_name           | trades
table_column_name   | bid
statistics_type      | NONE
statistics_updated_timestamp |
-[ RECORD 3 ]-----+-----
projection_name      | trades_p
```

```

projection_column_name | ask
column_position       | 2
sort_position         |
column_id             | 45035996273743514
data_type             | int
encoding_type        | AUTO
access_rank          | 0
group_id             | 45035996273743512
table_schema         | public
table_name           | trades
table_column_name    | ask
statistics_type       | NONE
statistics_updated_timestamp |

```

Now run statistics on the `stock` column:

```
=> SELECT ANALYZE_STATISTICS('trades.stock');
```

The system returns 0 for success:

```

-[ RECORD 1 ]-----+--
ANALYZE_STATISTICS | 0

```

Now query `PROJECTION_COLUMNS` again:

```
=> SELECT * FROM PROJECTION_COLUMNS where table_name = 'trades';
```

This time, `statistics_type` changes to `FULL` for the `trades.stock` column (representing full statistics were run), and the `statistics_updated_timestamp` column returns the time the `stock` columns statistics were updated. Note that the timestamp for the `bid` and `ask` columns have not changed because statistics were not run on those columns. Also, the `bid` and `ask` columns changed from `NONE` to `ROWCOUNT`. This is because Vertica automatically updates `ROWCOUNT` statistics from time to time. The statistics are created by looking at existing catalog metadata.

```

-[ RECORD 1 ]-----+-----
projection_name       | trades_p
projection_column_name | stock
column_position       | 0
sort_position         | 0
column_id            | 45035996273743508
data_type            | char(5)
encoding_type        | RLE
access_rank          | 0
group_id            | 0
table_schema         | public
table_name           | trades
table_column_name    | stock
statistics_type       | FULL
statistics_updated_timestamp | 2011-03-31 13:32:49.083544-04
-[ RECORD 2 ]-----+-----
projection_name       | trades_p
projection_column_name | bid
column_position       | 1
sort_position         | 1
column_id            | 45035996273743510
data_type            | int

```

```

encoding_type          | DELTAVAL
access_rank           | 0
group_id              | 45035996273743512
table_schema          | public
table_name             | trades
table_column_name     | bid
statistics_type       | ROWCOUNT
statistics_updated_timestamp | 2011-03-31 13:31:50.017845-04
-[ RECORD 3 ]-----+-----
projection_name        | trades_p
projection_column_name | ask
column_position       | 2
sort_position         |
column_id             | 45035996273743514
data_type             | int
encoding_type         | AUTO
access_rank           | 0
group_id              | 45035996273743512
table_schema          | public
table_name             | trades
table_column_name     | ask
statistics_type       | ROWCOUNT
statistics_updated_timestamp | 2011-03-31 13:31:50.017872-04

```

If you run statistics on the bid column and then query this system table again, only RECORD 2 is updated:

```

=> SELECT ANALYZE_STATISTICS('trades.bid');
-[ RECORD 1 ]-----+---
ANALYZE_STATISTICS | 0
=> SELECT * FROM PROJECTION_COLUMNS where table_name = 'trades';

-[ RECORD 1 ]-----+-----
projection_name        | trades_p
projection_column_name | stock
column_position       | 0
sort_position         | 0
column_id             | 45035996273743508
data_type             | char(5)
encoding_type         | RLE
access_rank           | 0
group_id              | 0
table_schema          | public
table_name             | trades
table_column_name     | stock
statistics_type       | FULL
statistics_updated_timestamp | 2011-03-31 13:32:49.083544-04
-[ RECORD 2 ]-----+-----
projection_name        | trades_p
projection_column_name | bid
column_position       | 1
sort_position         | 1
column_id             | 45035996273743510
data_type             | int
encoding_type         | DELTAVAL
access_rank           | 0

```

```

group_id          | 45035996273743512
table_schema     | public
table_name       | trades
table_column_name | bid
statistics_type  | FULL
statistics_updated_timestamp | 2011-03-31 13:35:35.817222-04
-[ RECORD 3 ]-----+-----
projection_name  | trades_p
projection_column_name | ask
column_position | 2
sort_position   |
column_id       | 45035996273743514
data_type       | int
encoding_type   | AUTO
access_rank     | 0
group_id        | 45035996273743512
table_schema    | public
table_name      | trades
table_column_name | ask
statistics_type | ROWCOUNT
statistics_updated_timestamp | 2011-03-31 13:31:50.017872-04

```

You can quickly query just the timestamp column to see when the columns were updated:

```

=> \x
Expanded display is off.

```

```

=> SELECT ANALYZE_STATISTICS('trades');
ANALYZE_STATISTICS
-----
                                0
(1 row)
=> SELECT projection_column_name, statistics_type,
       statistics_updated_timestamp
       FROM PROJECTION_COLUMNS where table_name = 'trades';

```

```

projection_column_name | statistics_type | STATISTICS_UPDATED_TIMESTAMP
-----+-----
stock                  | FULL           | 2011-03-31 13:39:16.968177-04
bid                    | FULL           | 2011-03-31 13:39:16.96885-04
ask                    | FULL           | 2011-03-31 13:39:16.968883-04
(3 rows)

```

**See Also**

**V\_CATALOG.PROJECTIONS** (page 811)

**ANALYZE\_STATISTICS** (page 385)

**CREATE PROJECTION** (page 640)

Collecting Statistics in the Administrator's Guide

## PROJECTION\_DELETE\_CONCERNS

Lists projections whose design may cause performance issues when deleting data. This table is generated by calling the **EVALUATE\_DELETE\_PERFORMANCE** (page 417) function. See *Optimizing Deletes and Updates for Performance in the Administrator's Guide* for more information.

Column Name	Data Type	Description
PROJECTION_ID	INTEGER	The ID number of the projection
PROJECTION_SCHEMA	VARCHAR	The schema containing the projection
PROJECTION_NAME	VARCHAR	The projection's name
CREATION_TIME	TIMESTAMPZ	When the projection was created
LAST_MODIFIED_TIME	TIMESTAMPZ	When the projection was last modified
COMMENT	VARCHAR	A comment describing the potential delete performance issue.

## PROJECTIONS

Provides information about projections.

Column Name	Data Type	Description
PROJECTION_SCHEMA_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the specific schema that contains the projection.
PROJECTION_SCHEMA	VARCHAR	The name of the schema that contains the projection.
PROJECTION_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_NAME	VARCHAR	The projection name for which information is listed.
OWNER_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the projection owner.
OWNER_NAME	VARCHAR	The name of the projection's owner.
ANCHOR_TABLE_ID	INTEGER	The unique numeric identification (OID) of the anchor table for pre-join projections, or the OID of the table from which the projection was created if it is not a pre-join projection. <b>Note:</b> A projection has only one anchor (fact) table.
ANCHOR_TABLE_NAME	VARCHAR	The name of the anchor table for pre-join projections, or the name of the table from which the projection was created if it is not a pre-join

		projection.
NODE_ID	INTEGER	A unique numeric ID (OID) that identifies the node(s) that contain the projection.
NODE_NAME	VARCHAR	The name of the node(s) that contain the projection. <b>Note:</b> this column returns information for unsegmented projections only, not for segmented and pinned projections.
IS_PREJOIN	BOOLEAN	Indicates whether the projection is a pre-join projection, where <i>t</i> is true and <i>f</i> is false.
CREATED_EPOCH	INTEGER	The epoch in which the projection was created.
CREATE_TYPE	VARCHAR	The method in which the projection was created: <ul style="list-style-type: none"> <li>▪ CREATE PROJECTION—A custom projection created through the CREATE PROJECTION statement.</li> <li>▪ CREATE TABLE—A superprojection that was automatically created when its associated table was created through the CREATE TABLE statement.</li> <li>▪ DELAYED_CREATION—A superprojection that was automatically created when data was loaded into its associated table.</li> <li>▪ DESIGNER—A projection created through Database Designer</li> <li>▪ IMPLEMENT_TEMP_DESIGN—A temporary projection. Note that Vertica no longer requires temp designs.</li> <li>▪ SYSTEM—A projection that was automatically created for a system table.</li> </ul>
VERIFIED_FAULT_TOLERANCE	INTEGER	The K-safety value for the projection.
IS_UP_TO_DATE	BOOLEAN	Indicates whether the projection is up to date, where <i>t</i> is true and <i>f</i> is false. Projections must be up to date to be used in queries.
HAS_STATISTICS	BOOLEAN	Indicates whether there are statistics for any column in the projection, where <i>t</i> is true and <i>f</i> is false. <b>Notes:</b> <ul style="list-style-type: none"> <li>▪ This column returns true only when all non-epoch columns for a table have full statistics. Otherwise the column returns false. See <b>ANALYZE_STATISTICS()</b> (page 385).</li> <li>▪ Projections that have no data never have full statistics. Use the <b>PROJECTION_STORAGE</b> (page 865)</li> </ul>

		system table to see if your projection contains data.
--	--	---

### Example

The following example queries the PROJECTION\_COLUMNS table to see if the projections are up to date and are pre-join projections.

```
=> SELECT projection_name, anchor_table_name, is_prejoin, is_up_to_date
FROM projections;
```

projection_name	anchor_table_name	is_prejoin	is_up_to_date
customer_dimension_site01	customer_dimension	f	t
customer_dimension_site02	customer_dimension	f	t
customer_dimension_site03	customer_dimension	f	t
customer_dimension_site04	customer_dimension	f	t
product_dimension_site01	product_dimension	f	t
product_dimension_site02	product_dimension	f	t
product_dimension_site03	product_dimension	f	t
product_dimension_site04	product_dimension	f	t
store_sales_fact_pl	store_sales_fact	t	t
store_sales_fact_pl_b1	store_sales_fact	t	t
store_orders_fact_pl	store_orders_fact	t	t
store_orders_fact_pl_b1	store_orders_fact	t	t
online_sales_fact_pl	online_sales_fact	t	t
online_sales_fact_pl_b1	online_sales_fact	t	t
promotion_dimension_site01	promotion_dimension	f	t
promotion_dimension_site02	promotion_dimension	f	t
promotion_dimension_site03	promotion_dimension	f	t
promotion_dimension_site04	promotion_dimension	f	t
date_dimension_site01	date_dimension	f	t
date_dimension_site02	date_dimension	f	t
date_dimension_site03	date_dimension	f	t
date_dimension_site04	date_dimension	f	t
vendor_dimension_site01	vendor_dimension	f	t
vendor_dimension_site02	vendor_dimension	f	t
vendor_dimension_site03	vendor_dimension	f	t
vendor_dimension_site04	vendor_dimension	f	t
employee_dimension_site01	employee_dimension	f	t
employee_dimension_site02	employee_dimension	f	t
employee_dimension_site03	employee_dimension	f	t
employee_dimension_site04	employee_dimension	f	t
shipping_dimension_site01	shipping_dimension	f	t
shipping_dimension_site02	shipping_dimension	f	t
shipping_dimension_site03	shipping_dimension	f	t
shipping_dimension_site04	shipping_dimension	f	t
warehouse_dimension_site01	warehouse_dimension	f	t
warehouse_dimension_site02	warehouse_dimension	f	t
warehouse_dimension_site03	warehouse_dimension	f	t
warehouse_dimension_site04	warehouse_dimension	f	t
inventory_fact_pl	inventory_fact	f	t
inventory_fact_pl_b1	inventory_fact	f	t
store_dimension_site01	store_dimension	f	t
store_dimension_site02	store_dimension	f	t
store_dimension_site03	store_dimension	f	t
store_dimension_site04	store_dimension	f	t
online_page_dimension_site01	online_page_dimension	f	t
online_page_dimension_site02	online_page_dimension	f	t
online_page_dimension_site03	online_page_dimension	f	t
online_page_dimension_site04	online_page_dimension	f	t
call_center_dimension_site01	call_center_dimension	f	t
call_center_dimension_site02	call_center_dimension	f	t
call_center_dimension_site03	call_center_dimension	f	t
call_center_dimension_site04	call_center_dimension	f	t

(52 rows)

**See Also****ANALYZE\_STATISTICS()** (page 385)**PROJECTION\_COLUMNS** (page 805)**PROJECTION\_STORAGE** (page 865)**RESOURCE\_POOLS**

Displays information about the parameters specified for the resource pool in the **CREATE RESOURCE POOL** (page 649) statement.

Column Name	Data Type	Description
NAME	VARCHAR	The name of the resource pool.
IS_INTERNAL	BOOLEAN	Denotes whether a pool is one of the <b>built-in pools</b> (page 652).
MEMORYSIZE	VARCHAR	Value of the amount of memory allocated to the resource pool
MAXMEMORYSIZE	VARCHAR	Value assigned as the maximum size the resource pool could grow by borrowing memory from the <b>GENERAL</b> pool.
PRIORITY	INTEGER	Value of <b>PRIORITY</b> parameter specified when defining the pool.
QUEUETIMEOUT	INTEGER	Value in seconds of <b>QUEUETIMEOUT</b> parameter specified when defining the pool. Represents the maximum amount of time the request is allowed to wait for resources to become available before being rejected.
PLANNEDCONCURRENCY	INTEGER	Value of <b>PLANNEDCONCURRENCY</b> parameter specified when defining the pool, which represents number of concurrent queries that are normally expected to be running against the resource pool.
MAXCONCURRENCY	INTEGER	Value of <b>MAXCONCURRENCY</b> parameter specified when defining the pool, which represents the maximum number of concurrent execution slots available to the resource pool.
SINGLEINITIATOR	BOOLEAN	Value that indicates whether all requests using this pool are issued against the same initiator node or whether multiple initiator nodes can be used; for instance in a round-robin configuration.

**Notes**

Column names in the **RESOURCE\_POOL** table mirror syntax in the **CREATE RESOURCE POOL** table; therefore, column names do not use underscores.

**Example**

```
=> SELECT * FROM V_CATALOG.RESOURCE_POOLS;
```

## SQL System Tables (Monitoring APIs)

```

name | is_internal | memorysize | maxmemorysize | priority | queue timeout | planned concurrency
| max concurrency | single initiator
-----+-----+-----+-----+-----+-----+-----
general | t | | Special: 95% | 0 | 300 | 4
| | f |
sysquery | t | 64M | | 110 | 300 | 4
| | f |
sysdata | t | 100M | 10% | | |
| | |
wosdata | t | 0% | 25% | | | 2
| | |
tm | t | 200M | | 105 | 300 | 2
| | |
| 3 | t |
refresh | t | 0% | | -10 | 300 | 4
| | t |
recovery | t | 0% | | 107 | 300 | 10
| | t |
| 5 | t |
dbd | t | 0% | | 0 | 0 | 4
| | t |
(8 rows)

```

### See Also

**CREATE RESOURCE POOL** (page 649)

Managing Workloads and Monitoring Resource Pools and Resource Usage by Queries in the Administrator's Guide for usage and examples.

## ROLES

Contains the names of all roles the user can access, along with any roles that have been assigned to those roles.

Column Name	Data Type	Description
NAME	VARCHAR	The name of a role that the user can access.
ASSIGNED_ROLES	VARCHAR	The names of any roles that have been granted to this role. By enabling the role, the user also has access to the privileges of these additional roles as well.

## SEQUENCES

Displays information about the parameters specified for a sequence using the **CREATE SEQUENCE** (page 658) statement.

Column Name	Data Type	Description
SEQUENCE_SCHEMA	VARCHAR	Schema in which the sequence was created.

SEQUENCE_NAME	VARCHAR	Name of the sequence defined in the CREATE SEQUENCE statement.
OWNER_NAME	VARCHAR	Name of the owner; for example, dbadmin.
IDENTITY_TABLE_NAME	VARCHAR	If created by an identity column, the name of the table to which it belongs. See <b>column constraints</b> (page 674) in the <b>CREATE TABLE</b> (page 664) statement.
SESSION_CACHE_COUNT	INTEGER	Count of values cached in a session.
ALLOW_CYCLE	BOOLEAN	Values allowed to cycle when max/min is reached. See CYCLE   NO CYCLE parameter in <b>CREATE SEQUENCE</b> (page 658).
OUTPUT_ORDERED	BOOLEAN	Values guaranteed to be ordered (always false).
INCREMENT_BY	INTEGER	Sequence values are incremented by this number (negative for reverse sequences).
MINIMUM	INTEGER	Minimum value the sequence can generate.
MAXIMUM	INTEGER	Maximum value the sequence can generate.
CURRENT_VALUE	INTEGER	Current value of the sequence.
SEQUENCE_SCHEMA_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the schema.
SEQUENCE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the sequence.
OWNER_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the user who created the sequence.
IDENTITY_TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the table to which the column belongs (if created by an identity column).

### Example

Create a simple sequence:

```
=> CREATE SEQUENCE my_seq MAXVALUE 5000 START 150;
CREATE SEQUENCE
```

Return information about the sequence you just created:

```
=> \x
Expanded display is on.
=> SELECT * FROM sequences;
-[ RECORD 1 ]-----+-----
sequence_schema      | public
sequence_name        | my_seq
owner_name           | dbadmin
identity_table_name  |
session_cache_count  | 250000
allow_cycle           | f
output_ordered       | f
```

```

increment_by      | 1
minimum          | 1
maximum          | 5000
current_value    | 149
sequence_schema_id | 45035996273704966
sequence_id      | 45035996273844996
owner_id         | 45035996273704962
identity_table_id | 0

```

You can also issue the `vsq` command `\ds` to return a list of sequences. The results below show the sequence created in the previous example. If more sequences existed, they would appear in this table.

```
=> \ds
```

```

                                List of Sequences
 Schema | Sequence | CurrentValue | IncrementBy | Minimum | Maximum | AllowCycle
-----+-----+-----+-----+-----+-----+-----
---
 public | my_seq   |          149 |           1 |         1 |       5000 | f
(1 row)

```

### See Also

**CREATE SEQUENCE** (page 658)

The `\d [ PATTERN ]` meta-commands in the Programmer's Guide

Using Sequences in the Administrator's Guide

## SYSTEM\_TABLES

Returns a list of all system table names.

Column Name	Data Type	Description
TABLE_SCHEMA	VARCHAR	The schema name in which the system table resides; for example, <code>V_CATALOG</code> (page 795) or <code>V_MONITOR</code> (page 828).
TABLE_NAME	VARCHAR	The name of the system table.
TABLE_DESCRIPTION	VARCHAR	A description of the system table's purpose.

### Example

Call all the system tables and order them by schema:

```
=> SELECT * FROM system_tables ORDER BY 1, 2;
```

```

table_schema | table_name | table_description
-----+-----+-----
-----
 v_catalog   | columns   | Table column information
 v_catalog   | dual      | Oracle(TM) compatibility DUAL table
 v_catalog   | foreign_keys | Foreign key information
 v_catalog   | grants    | Grant information
 v_catalog   | passwords | User password history and password reuse policy

```

v_catalog	primary_keys	Primary key information
v_catalog	profile_parameters	Profile Parameters information
v_catalog	profiles	Profile information
v_catalog	projection_columns	Projection columns information
v_catalog	projections	Projection information
v_catalog	resource_pools	Information about defined resource pools
v_catalog	system_tables	Displays a list of all non-internal system tables
v_catalog	table_constraints	Constraint information
v_catalog	tables	Table information
v_catalog	types	Information about supported data types
v_catalog	user_functions	User Defined Function information
v_catalog	user_procedures	User procedure information
v_catalog	users	User information
v_catalog	view_columns	View column information
v_catalog	views	View information
v_monitor	active_events	Displays all of the active events in the cluster
v_monitor	column_storage	Information on the amount of disk storage in use
v_monitor	configuration_parameters	Configuration Parameters information
v_monitor	current_session	Information on current Session
v_monitor	database_snapshots	Information on stored database snapshots
v_monitor	delete_vectors	Information on delete vectors
v_monitor	disk_resource_rejections	Disk Resource Rejection Summarizations
v_monitor	disk_storage	Disk usage information
v_monitor	event_configurations	Current Event configuration
v_monitor	execution_engine_profiles	Per EE operator profiling information
v_monitor	host_resources	Per host profiling information
v_monitor	load_streams	Load metrics for each load stream on each node.
v_monitor	locks	Lock grants and requests for all nodes
v_monitor	node_resources	Per node profiling information
v_monitor	partitions	Partition metadata
v_monitor	projection_refreshes	Refresh information on each Projection
v_monitor	projection_storage	Storage information on each Projection
v_monitor	query_metrics	Summarized query information
v_monitor	query_profiles	Query Profiling
v_monitor	resource_acquisitions	Resources in use by queries
v_monitor	resource_acquisitions_history	Resources used by completed queries
v_monitor	resource_pool_status	Resource pool usage Information
v_monitor	resource_queues	Queries waiting to acquire resources
v_monitor	resource_rejections	Resource Rejection Summarizations
v_monitor	resource_usage	Resource usage Information
v_monitor	session_profiles	Per session profiling information
v_monitor	sessions	Information on each Session
v_monitor	storage_containers	Information on each storage container
v_monitor	strata	Information of strata used in Tuple Mover
v_monitor	strata_structures	Information of strata structures used in Tuple Mover
v_monitor	system	System level information
v_monitor	tuple_mover_operations	Information about (automatic) Tuple Mover
v_monitor	wos_container_storage	Storage information on WOS allocator

(53 rows)

## TABLE\_CONSTRAINTS

Provides information about table constraints.

Column Name	Data Type	Description
CONSTRAINT_ID	VARCHAR	A unique numeric ID assigned by the Vertica catalog, which identifies the constraint.
CONSTRAINT_NAME	VARCHAR	The name of the constraint, if specified as UNIQUE, FOREIGN KEY, NOT NULL, or PRIMARY KEY.
CONSTRAINT_SCHEMA_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the schema containing the constraint.



- | p  
(33 rows)

## See Also

***ANALYZE CONSTRAINTS*** (page 377)

Adding Constraints in the Administrator's Guide

## TABLES

Provides information about all tables in the database.

Column Name	Data Type	Description
TABLE_SCHEMA_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the schema.
TABLE_SCHEMA	VARCHAR	The schema name for which information is listed.
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the table.
TABLE_NAME	VARCHAR	The table name for which information is listed.
OWNER_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the owner.
OWNER_NAME	VARCHAR	The name of the user who created the table.
IS_SYSTEM_TABLE	BOOLEAN	Indicates whether table is a system table, where <i>t</i> is true and <i>f</i> is false.
SYSTEM_TABLE_CREATOR	VARCHAR	The name of the process that created the table, such as Designer.
PARTITION_EXPRESSION	VARCHAR	The partition expression for the table.

## Notes

The `TABLE_SCHEMA` and `TABLE_NAME` columns are case sensitive when you run queries that contain the equality (=) predicate. Use the `ILIKE` predicate instead:

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema ILIKE 'schema1';
```

## Example

The following command returns information on all tables in the Vmart schema:

```
vmartdb=> SELECT table_schema, table_name, owner_name, is_system_table FROM TABLES;
```

```
table_schema | table_name | owner_name | is_system_table
-----+-----+-----+-----
public      | customer_dimension | release | f
public      | product_dimension | release | f
public      | promotion_dimension | release | f
public      | date_dimension | release | f
public      | vendor_dimension | release | f
```

```

public      | employee_dimension | release   | f
public      | shipping_dimension | release  | f
public      | warehouse_dimension | release  | f
public      | inventory_fact     | release  | f
store       | store_dimension   | release  | f
store       | store_sales_fact   | release  | f
store       | store_orders_fact  | release  | f
online_sales | online_page_dimension | release | f
online_sales | call_center_dimension | release | f
online_sales | online_sales_fact   | release | f
(15 rows)

```

## TYPES

Provides information about supported data types.

Column Name	Data Type	Description
TYPE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the specific data type.
TYPE_NAME	VARCHAR	The data type name associated with a particular data type ID.

### Example

```

=> SELECT * FROM types;
  type_id | type_name
-----+-----
      5 | Boolean
      6 | Integer
      7 | Float
      8 | Char
      9 | Varchar
     10 | Date
     11 | Time
     12 | Timestamp
     13 | TimestampTz
     14 | Interval
     15 | TimeTz
     16 | Numeric
     17 | Varbinary
    117 | Binary
(14 rows)

```

## USER\_AUDITS

Lists the results of database and object size audits generated by users calling the **AUDIT** (page 390) function. See *Monitoring Database Size for License Compliance* in the Administrator's Guide for more information.

Column Name	Data Type	Description
SIZE_BYTES	INTEGER	The estimated raw data size of the database
USER_ID	INTEGER	The ID of the user who generated the audit

USER_NAME	VARCHAR	The name of the user who generated the audit
OBJECT_ID	INTEGER	The ID of the object being audited
OBJECT_TYPE	VARCHAR	The type of object being audited (table, schema, etc.)
OBJECT_SCHEMA	VARCHAR	The schema containing the object being audited
OBJECT_NAME	VARCHAR	The name of the object being audited
AUDIT_START_TIMESTAMP	TIMESTAMP Z	When the audit started
AUDIT_END_TIMESTAMP	TIMESTAMP Z	When the audit finished
CONFIDENCE_LEVEL_PERCENT	FLOAT	The confidence level of the size estimate
ERROR_TOLERANCE_PERCENT	FLOAT	The error tolerance used for the size estimate
USED_SAMPLING	BOOLEAN	Whether data was randomly sampled (if false, all of the data was analyzed)
CONFIDENCE_INTERVAL_LOWER_BOUND_BYTES	INTEGER	The lower bound of the data size estimate within the confidence level
CONFIDENCE_INTERVAL_UPPER_BOUND_BYTES	INTEGER	The upper bound of the data size estimate within the confidence level
SAMPLE_COUNT	INTEGER	The number of data samples used to generate the estimate
CELL_COUNT	INTEGER	The number of cells in the database

## USER\_FUNCTIONS

Returns metadata about user-defined SQL Macros (which store commonly used SQL expressions as a function in the Vertica catalog) and User Defined Scalar Functions (UDSF).

Column Name	Data Type	Description
FUNCTION_NAME	VARCHAR	The name assigned by the user to the SQL Macro or UDSF.
FUNCTION_RETURN_TYPE	VARCHAR	The data type name that the SQL Macro returns.
FUNCTION_ARGUMENT_TYPE	VARCHAR	The number and data types of parameters for the function.
FUNCTION_DEFINITION	VARCHAR	The SQL expression that the user defined in the SQL Macro's function body.
VOLATILITY	VARCHAR	The SQL Macro's volatility (whether a function returns the same output given the same input). Can be immutable, volatile, or stable.

IS_STRICT	BOOLEAN	Indicates whether the SQL Macro is strict, where <i>t</i> is true and <i>f</i> is false.
-----------	---------	--

## Notes

- The volatility and strictness of a SQL Macro are automatically inferred from the function definition in order that Vertica perform constant folding optimization, when possible, and determine the correctness of usage, such as where an immutable function is expected but a volatile function is provided.
- The volatility and strictness of UDSF is set by the superuser when creating the function. See **CREATE FUNCTION (UDF)** (page 631) for details.

## Example

Create a SQL Macro called `zeroifnull` in the public schema:

```
=> CREATE FUNCTION zeroifnull(x INT) RETURN INT
  AS BEGIN
    RETURN (CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END);
  END;
```

Now query the `USER_FUNCTIONS` table. The query returns just the `zeroifnull` macro because it is the only one created in this schema:

```
=> SELECT * FROM user_functions;
-[ RECORD 1 ]-----+-----
schema_name          | public
function_name        | zeroifnull
function_return_type | Integer
function_argument_type | x Integer
function_definition  | RETURN CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END
volatility            | immutable
is_strict             | f
```

## See Also

**CREATE FUNCTION** (page 629)

**ALTER FUNCTION** (page 578)

**DROP FUNCTION** (page 700)

**GRANT (Function)** (page 721)

**REVOKE (Function)** (page 735)

See also Using SQL Macros in the Programmer's Guide

## USER\_PROCEDURES

Provides information about external procedures that have been defined for Vertica. User see only the procedures they can execute.

Column Name	Data Type	Description
PROCEDURE_NAME	VARCHAR	The name given to the external procedure through the CREATE PROCEDURE statement.
PROCEDURE_ARGUMENTS	VARCHAR	Lists arguments for the external procedure.
SCHEMA_NAME	VARCHAR	Indicates the schema in which the external procedure is defined.

### Example

```
=> SELECT * FROM user_procedures;
```

```

procedure_name | procedure_arguments | schema_name
-----+-----+-----
helloplanet   | arg1 Varchar       | public
(1 row)

```

## USER\_TRANSFORMS

Lists the currently-defined User Defined Transform Functions (UDTFs).

Column Name	Data Type	Description
SCHEMA_NAME	VARCHAR	The name of the schema containing the UDTF.
FUNCTION_NAME	VARCHAR	The SQL function name assigned by the user.
FUNCTION_RETURN_TYPE	VARCHAR	The data types of the columns the UDTF returns.
FUNCTION_ARGUMENT_TYPE	VARCHAR	The data types of the columns that make up the input row.
FUNCTION_DEFINITION	VARCHAR	A string containing the name of the factory class for the UDTF, and the name of the library that contains it.

## USERS

Provides information about all users in the database.

Column Name	Data Type	Description
USER_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the user.
USER_NAME	VARCHAR	The user name for which information is listed.
IS_SUPER_USER	BOOLEAN	Indicates whether the current user is superuser, where <i>t</i> is true and <i>f</i> is false.

PROFILE_NAME	VARCHAR	The name of the profile to which the user is assigned. The profile controls the user's password policy.
IS_LOCKED	BOOLEAN	Whether the user's account is locked. A locked user cannot log into the system.
LOCK_TIME	DATE TIME	When the user's account was locked. Used to determine when to automatically unlock the account, if the user's profile has a PASSWORD_LOCK_TIME parameter set.
RESOURCE_POOL	VARCHAR	The resource pool to which the user is assigned.
MEMORY_CAP_KB	VARCHAR	The maximum amount of memory a query run by the user can consume, in kilobytes.
TEMP_SPACE_CAP_KB	VARCHAR	The maximum amount of temporary disk space a query run by the user can consume, in kilobytes.
RUN_TIME_CAP	VARCHAR	The maximum amount of time any of the user's queries is allowed to run.

### Example

```
=> \x
Expanded display is on.
=> SELECT * FROM users;
-[ RECORD 1 ]-----+-----
user_id          | 45035996273704962
user_name        | dbadmin
is_super_user    | t
profile_name     | default
is_locked        | f
lock_time        |
resource_pool    | general
memory_cap_kb    | unlimited
temp_space_cap_kb | unlimited
run_time_cap     | unlimited
-[ RECORD 2 ]-----+-----
user_id          | 45035996273708334
user_name        | exampleuser
is_super_user    | f
profile_name     | default
is_locked        | f
lock_time        |
resource_pool    | general
memory_cap_kb    | unlimited
temp_space_cap_kb | unlimited
run_time_cap     | unlimited
```

## VIEW\_COLUMNS

Provides view attribute information.

Column Name	Data Type	Description
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the view of the table.
TABLE_SCHEMA	VARCHAR	The schema name for which information is listed.
TABLE_NAME	VARCHAR	The table name for which information is listed.
COLUMN_NAME	VARCHAR	The column name for which information is listed.
DATA_TYPE	VARCHAR	The data type of the column for which information is listed; for example, VARCHAR(128).
DATA_TYPE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the data type.
DATA_TYPE_LENGTH	INTEGER	The maximum allowable length for the data type.
CHARACTER_MAXIMUM_LENGTH	INTEGER	The maximum allowable length for the column, valid for character types.
NUMERIC_PRECISION	INTEGER	The number of significant decimal digits.
NUMERIC_SCALE	INTEGER	The number of fractional digits.
DATETIME_PRECISION	INTEGER	For TIMESTAMP data type, returns the declared precision; returns null if no precision was declared.
INTERVAL_PRECISION	INTEGER	The number of fractional digits retained in the seconds field.
ORDINAL_POSITION	VARCHAR	The position of the column relative to other columns.

## Notes

A warning like the following means only that view <t> had its associated table dropped. The view is not returned by the `SELECT * FROM view_columns` command, and the warning is returned merely to notify users about an orphaned view.

```
WARNING: invalid view v: relation "public.t" does not exist
```

## Example

```
=>\pset expanded
Expanded display is on.
=> SELECT * FROM view_columns;
-[ RECORD 3 ]-----+-----
table_id          | 45035996273881226
table_schema      | public
table_name        | t_cpp_v
column_name       | c3
data_type         | Char
data_type_id      | 8
data_type_length  | 10
character_maximum_length | 10
numeric_precision |
```

```

numeric_scale          |
datetime_precision    |
interval_precision    |
ordinal_position      | 3
-[ RECORD 4 ]-----+-----
table_id               | 45035996273881226
table_schema           | public
table_name             | t_cpp_v
column_name            | c4
data_type              | Date
data_type_id           | 10
data_type_length       | 8
character_maximum_length |
numeric_precision      |
numeric_scale          |
datetime_precision    |
interval_precision    |
ordinal_position      | 4

```

NULL fields in the above results indicate that those columns were not defined. For example, given the following table, the result for the `datetime_precision` column is NULL because no precision was declared:

```

=> CREATE TABLE c (c TIMESTAMP);
CREATE TABLE
=> SELECT table_name, column_name, datetime_precision FROM columns WHERE
table_name = 'c';
 table_name | column_name | datetime_precision
-----+-----+-----
c          | c          |
(1 row)

```

In this example, the `datetime_precision` column returns 4 because the precision was declared as 4 in the `CREATE TABLE` statement:

```

=> DROP TABLE c;
=> CREATE TABLE c (c TIMESTAMP(4));
CREATE TABLE
=> SELECT table_name, column_name, datetime_precision FROM columns WHERE
table_name = 'c';
 table_name | column_name | datetime_precision
-----+-----+-----
c          | c          | 4
(1 row)

```

## See Also

**VIEWS** (page 827)

## VIEWS

Provides information about all views within the system. See [Implementing Views](#) for more information.

Column Name	Data Type	Description
TABLE_SCHEMA_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the table schema.
TABLE_SCHEMA	VARCHAR	The name of the schema that contains the view.
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the table.
TABLE_NAME	VARCHAR	The table name for which information is listed.
OWNER_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the owner.
OWNER_NAME	VARCHAR	The name of the view owner.
VIEW_DEFINITION	VARCHAR	The query used to define the view.
IS_SYSTEM_VIEW	BOOLEAN	Indicates whether the table is a system view, where <i>t</i> is true and <i>f</i> is false.
SYSTEM_VIEW_CREATOR	VARCHAR	The user name who created the view.

### Example

Query the VIEWS table:

```
=>\pset expanded
Expanded display is on.
=> SELECT * FROM VIEWS;
-[ RECORD 1 ]-----+-----
table_schema_id      | 45035996273704963
table_schema         | public
table_id             | 45035996273823130
table_name           | temp
owner_id             | 45035996273704961
owner_name           | release
view_definition      | SELECT to_date('F'::character varying, 'dd mm yyyy'::character
                    | varying) AS to_date FROM public.customer_dimension
is_system_view       | f
system_view_creator  |
```

### See Also

Implementing Views

**VIEW\_COLUMNS** (page 825)

## V\_MONITOR Schema

The system tables in this section reside in the `v_monitor` schema. These tables provide information about the health of the Vertica database.

### ACTIVE\_EVENTS

Returns all active events in the cluster. See Monitoring Events.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name where the event occurred.
EVENT_CODE	INTEGER	A numeric ID that indicates the type of event. See Event Types for a list of event type codes.
EVENT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the specific event.
EVENT_SEVERITY	VARCHAR	The severity of the event from highest to lowest. These events are based on standard syslog severity types. <ul style="list-style-type: none"> <li>▪ 0—Emergency</li> <li>▪ 1—Alert</li> <li>▪ 2—Critical</li> <li>▪ 3—Error</li> <li>▪ 4—Warning</li> <li>▪ 5—Notice</li> <li>▪ 6—Informational</li> <li>▪ 7—Debug</li> </ul>
EVENT_POSTED_TIMESTAMP	TIMESTAMP	The year, month, day, and time the event was reported. The time is posted in military time.
EVENT_EXPIRATION	VARCHAR	The year, month, day, and time the event expire. The time is posted in military time. If the cause of the event is still active, the event is posted again.
EVENT_CODE_DESCRIPTION	VARCHAR	A brief description of the event and details pertinent to the specific situation.
EVENT_PROBLEM_DESCRIPTION	VARCHAR	A generic description of the event.
REPORTING_NODE	VARCHAR	The name of the node within the cluster that reported the event.
EVENT_SENT_TO_CHANNELS	VARCHAR	The event logging mechanisms that are configured for Vertica. These can include <code>vertica.log</code> , (configured by default) <code>syslog</code> , and <code>SNMP</code> .
EVENT_POSTED_COUNT	INTEGER	Tracks the number of times an event occurs. Rather than posting the same event multiple times, Vertica posts the event once and then counts the number of additional instances in which the event occurs.

### Example

Query the `ACTIVE_EVENTS` table:

```
=>\pset expanded
Expanded display is on.
=> SELECT * FROM active_events;
```

```
-[ RECORD 1 ]-----+-----
```

```

node_name           | site01
event_code          | 6
event_id            | 6
event_severity      | Informational
is_event_posted     | 2009-08-11 09:38:39.008458
event_expiration    | 2077-08-29 11:52:46.008458
event_code_description | Node State Change
event_problem_description | Changing node site01 startup state to UP
reporting_node      | site01
event_sent_to_channels | Vertica Log
event_posted_count  | 1
-[ RECORD 2 ]-----+-----
node_name           | site02
event_code          | 6
event_id            | 6
event_severity      | Informational
is_event_posted     | 2009-08-11 09:38:39.018172
event_expiration    | 2077-08-29 11:52:46.018172
event_code_description | Node State Change
event_problem_description | Changing node site02 startup state to UP
reporting_node      | site02
event_sent_to_channels | Vertica Log
event_posted_count  | 1
-[ RECORD 3 ]-----+-----
current_timestamp   | 2009-08-11 14:38:48.859987
node_name           | site03
event_code          | 6
event_id            | 6
event_severity      | Informational
is_event_posted     | 2009-08-11 09:38:39.027258
event_expiration    | 2077-08-29 11:52:46.027258
event_code_description | Node State Change
event_problem_description | Changing node site03 startup state to UP
reporting_node      | site03
event_sent_to_channels | Vertica Log
event_posted_count  | 1
-[ RECORD 4 ]-----+-----
node_name           | site04
event_code          | 6
event_id            | 6
event_severity      | Informational
is_event_posted     | 2009-08-11 09:38:39.008288
event_expiration    | 2077-08-29 11:52:46.008288
event_code_description | Node State Change
event_problem_description | Changing node site04 startup state to UP
reporting_node      | site04
event_sent_to_channels | Vertica Log
event_posted_count  | 1
...

```

## COLUMN\_STORAGE

Returns the amount of disk storage used by each column of each projection on each node.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
COLUMN_NAME	VARCHAR	The column name for which information is listed.
ROW_COUNT	INTEGER	The number of rows in the column.
USED_BYTES	INTEGER	The disk storage allocation of the column in bytes.
ENCODINGS	VARCHAR	The encoding type for the column.
COMPRESSION	VARCHAR	The compression type for the column.
WOS_ROW_COUNT	INTEGER	The number of WOS rows in the column.
ROS_ROW_COUNT	INTEGER	The number of ROS rows in the column.
ROS_USED_BYTES	INTEGER	The number of ROS bytes in the column.
ROS_COUNT	INTEGER	The number of ROS containers.
PROJECTION_NAME	VARCHAR	The associated projection name for the column.
PROJECTION_SCHEMA	VARCHAR	The name of the schema associated with the projection.
ANCHOR_TABLE_NAME	VARCHAR	The associated table name.
ANCHOR_TABLE_SCHEMA	VARCHAR	The associated table's schema name.

## Notes

- WOS data is stored by row, so per-column byte counts are not available.
- The `ENCODINGS` and `COMPRESSION` columns let you comparing the affect of different encoding types on column storage, when optimizing for compression.

## Example

Query the `COLUMN_STORAGE` table:

```
=> \pset expanded
Expanded display is on.
=> SELECT * FROM COLUMN_STORAGE;
-[ RECORD 1 ]-----+-----
node_name          | node0001
column_name        | bincol
row_count          | 2
used_bytes         | 0
encodings          | String
compressions       | lzo
wos_row_count      | 0
ros_row_count      | 2
ros_used_bytes     | 0
ros_count          | 1
projection_name    | allTypes_super
projection_schema  | public
anchor_table_name  | allTypes
anchor_table_schema | public
```

```

-[ RECORD 12 ]-----+-----
-
node_name          | node0001
column_name        | boolcol
row_count          | 2
used_bytes         | 0
encodings          | Uncompressed
compressions       | lzo
wos_row_count      | 0
ros_row_count      | 2
ros_used_bytes     | 0
ros_count          | 1
projection_name    | allTypes_super
projection_schema  | public
anchor_table_name  | allTypes
anchor_table_schema | public
-[ RECORD 13 ]-----+-----
-
node_name          | node0001
column_name        | charcol
row_count          | 2
used_bytes         | 0
encodings          | String
compressions       | lzo
wos_row_count      | 0
ros_row_count      | 2
ros_used_bytes     | 0
ros_count          | 1
projection_name    | allTypes_super
projection_schema  | public
anchor_table_name  | allTypes
anchor_table_schema | public
-[ RECORD 4 ]-----+-----
...

```

**Call specific columns from the COLUMN\_STORAGE table:**

```

SELECT column_name, row_count, projection_name, anchor_table_name
FROM COLUMN_STORAGE
WHERE node_name = 'site02' AND row_count = 1000;

```

column_name	row_count	projection_name	anchor_table_name
end_date	1000	online_page_dimension_site02	online_page_dimension
epoch	1000	online_page_dimension_site02	online_page_dimension
online_page_key	1000	online_page_dimension_site02	online_page_dimension
page_description	1000	online_page_dimension_site02	online_page_dimension
page_number	1000	online_page_dimension_site02	online_page_dimension
page_type	1000	online_page_dimension_site02	online_page_dimension
start_date	1000	online_page_dimension_site02	online_page_dimension
ad_media_name	1000	promotion_dimension_site02	promotion_dimension
ad_type	1000	promotion_dimension_site02	promotion_dimension
coupon_type	1000	promotion_dimension_site02	promotion_dimension
display_provider	1000	promotion_dimension_site02	promotion_dimension
display_type	1000	promotion_dimension_site02	promotion_dimension
epoch	1000	promotion_dimension_site02	promotion_dimension
price_reduction_type	1000	promotion_dimension_site02	promotion_dimension
promotion_begin_date	1000	promotion_dimension_site02	promotion_dimension
promotion_cost	1000	promotion_dimension_site02	promotion_dimension
promotion_end_date	1000	promotion_dimension_site02	promotion_dimension

```

promotion_key          |          1000 | promotion_dimension_site02 | promotion_dimension
promotion_media_type   |          1000 | promotion_dimension_site02 | promotion_dimension
promotion_name         |          1000 | promotion_dimension_site02 | promotion_dimension
20 rows)

```

## CONFIGURATION\_PARAMETERS

Provides information about configuration parameters currently in use by the system.

**Caution:** Contact *Technical Support* (on page 1) before changing any of the parameters that are not explicitly included in the documentation.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node names on the cluster for which information is listed.
PARAMETER_NAME	VARCHAR	The name of the configurable parameter. See Configuration Parameters in the Administrator's Guide for a detailed list of supported parameters.
CURRENT_VALUE	INTEGER	The value of the current setting for the parameter.
DEFAULT_VALUE	INTEGER	The default value for the parameter.
CHANGE_UNDER_SUPPORT_GUIDANCE	BOOLEAN	A <i>t</i> (true) setting indicates that changes to configuration parameters require guidance from Vertica <i>Technical Support</i> (on page 1).
CHANGE_REQUIRES_RESTART	BOOLEAN	Indicates whether the configuration change requires a restart, where <i>t</i> is true and <i>f</i> is false.
DESCRIPTION	VARCHAR	A description of the parameter's purpose.

### Notes

The CONFIGURATION\_PARAMETERS function returns the following error in non-default locales:

```
ERROR: ORDER BY is not supported with UNION/INTERSECT/EXCEPT in non-default locales
```

```
HINT: Please move the UNION to a FROM clause subquery.
```

See the **SET LOCALE** command for details.

### Example

The following command returns all current configuration parameters in Vertica:

```
=> SELECT * FROM CONFIGURATION_PARAMETERS;
```

### See Also

Configuration Parameters in the Administrator's Guide.

## CRITICAL\_NODES

Lists the critical nodes whose failure would cause the database to become unsafe and force a shutdown.

Column Name	Date Type	Description
NODE_NAME	VARCHAR	The name of a critical node

### Example

```
=> SELECT * FROM V_MONITOR.CRITICAL_NODES;
      node_name
-----
v_ExampleDB_node0001
v_ExampleDB_node0003
(2 rows)
```

## CURRENT\_SESSION

Returns information about the current active session. You can use this table to find out the current session's sessionID and get the duration of the previously-run query.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
USER_NAME	VARCHAR	The name used to log into the database or NULL if the session is internal.
CLIENT_HOSTNAME	VARCHAR	The host name and port of the TCP socket from which the client connection was made; NULL if the session is internal
CLIENT_PID	INTEGER	The process identifier of the client process that issued this connection. <b>Note:</b> Remember that the client process could be on a different machine than the server.
LOGIN_TIMESTAMP	TIMESTAMP	The date and time the user logged into the database or when the internal session was created. This column can be useful for identifying sessions that have been left open and could be idle.
SESSION_ID	VARCHAR	The identifier required to close or interrupt a session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
CLIENT_LABEL	VARCHAR	A user-specified label for the client connection that can be set when using ODBC. See <code>SessionLabel</code> in DSN Parameters in Programmer's Guide.
TRANSACTION_START	TIMESTAMP	The date/time the current transaction started or

	P	NULL if no transaction is running.
TRANSACTION_ID	VARCHAR	A string containing the hexadecimal representation of the transaction ID, if any; otherwise NULL.
TRANSACTION_DESCRIPTION	VARCHAR	A description of the current transaction.
STATEMENT_START	TIMESTAMP	The date/time the current statement started execution, or NULL if no statement is running.
STATEMENT_ID	VARCHAR	An ID for the currently-running statement. NULL indicates that no statement is currently being processed.
LAST_STATEMENT_DURATION_US	INTEGER	The duration of the last completed statement in microseconds.
CURRENT_STATEMENT	VARCHAR	The currently-running statement, if any. NULL indicates that no statement is currently being processed.
LAST_STATEMENT	VARCHAR	NULL if the user has just logged in; otherwise the currently running statement or the most recently completed statement.
EXECUTION_ENGINE_PROFILING_CONFIGURATION	VARCHAR	Returns a value that indicates whether profiling is turned on. Results are: <ul style="list-style-type: none"> <li>▪ Empty when no profiling</li> <li>▪ 'Local' when profiling on for this session</li> <li>▪ 'Global' when on by default for all sessions</li> <li>▪ 'Local, Global' when on by default for all sessions and on for current session</li> </ul>
QUERY_PROFILING_CONFIGURATION	VARCHAR	Returns a value that indicates whether profiling is turned on. Results are: <ul style="list-style-type: none"> <li>▪ Empty when no profiling</li> <li>▪ 'Local' when profiling on for this session</li> <li>▪ 'Global' when on by default for all sessions</li> <li>▪ 'Local, Global' when on by default for all sessions and on for current session</li> </ul>
SESSION_PROFILING_CONFIGURATION	VARCHAR	Returns a value that indicates whether profiling is turned on. Results are: <ul style="list-style-type: none"> <li>▪ Empty when no profiling</li> <li>▪ 'Local' when profiling on for this session</li> <li>▪ 'Global' when on by default for all sessions</li> <li>▪ 'Local, Global' when on by default for all sessions and on for current session</li> </ul>

**Notes**

- The default for profiling is ON ('1') for all sessions. Each session can turn profiling ON or OFF.
- Profiling parameters (such as `GlobalEEProfiling` in the examples below) are set in the Vertica configuration file (`vertica.conf`). To turn profiling off, set the parameter to '0'. To turn profiling on, set the parameter to '1'.

**Examples**

Query the `CURRENT_SESSION` table:

```
=> SELECT * FROM CURRENT_SESSION;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node01
user_name          | release
client_hostname    | xxx.x.x.x:xxxxx
client_pid         | 18082
login_timestamp    | 2010-10-07 10:10:03.114863-04
session_id         | myhost-17956:0x1d
client_label       |
transaction_start  | 2010-10-07 11:52:32.43386
transaction_id     | 45035996273727909
transaction_description | user release (select * from passwords;)
statement_start    | 2010-10-07 12:30:42.444459
statement_id       | 11
last_statement_duration_us | 85241
current_statement  | SELECT * FROM CURRENT_SESSION;
last_statement     | SELECT * FROM CONFIGURATION_PARAMETERS;
execution_engine_profiling_configuration | Local
query_profiling_configuration |
session_profiling_configuration |
```

Request specific columns from the table:

```
=> SELECT node_name, session_id, execution_engine_profiling_configuration
       FROM CURRENT_SESSION;
 node_name | session_id | execution_engine_profiling_configuration
-----+-----+-----
 site01    | myhost-17956:0x1d | Global
(1 row)
```

The sequence of commands in this example shows the use of disabling and enabling profiling for local and global sessions.

This command disables EE profiling for query execution runs:

```
=> SELECT disable_profiling('EE');
disable_profiling
-----
EE Profiling Disabled
(1 row)
```

The following command sets the `GlobalEEProfiling` configuration parameter to 0, which turns off profiling:

```
=> SELECT set_config_parameter('GlobalEEProfiling', '0');
set_config_parameter
-----
Parameter set successfully
```

```
(1 row)
```

The following command tells you whether profiling is set to 'Local' or 'Global' or none:

```
=> SELECT execution_engine_profiling_configuration FROM CURRENT_SESSION;
ee_profiling_config
-----
```

```
(1 row)
```

**Note:** The result set is empty because profiling was turned off in the preceding example.

This command now enables EE profiling for query execution runs:

```
=> SELECT enable_profiling('EE');
enable_profiling
-----
EE Profiling Enabled
(1 row)
```

Now when you run a select on the CURRENT\_SESSION table, you can see profiling is ON for the local session:

```
=> SELECT execution_engine_profiling_configuration FROM CURRENT_SESSION;
ee_profiling_config
-----
Local
(1 row)
```

Now turn profiling on for all sessions by setting the GlobalEEProfiling configuration parameter to 1:

```
=> SELECT set_config_parameter('GlobalEEProfiling', '1');
set_config_parameter
-----
Parameter set successfully
(1 row)
```

Now when you run a select on the CURRENT\_SESSION table, you can see profiling is ON for the local sessions, as well as for all sessions:

```
=> SELECT execution_engine_profiling_configuration FROM CURRENT_SESSION;
ee_profiling_config
-----
Local, Global
(1 row)
```

## See Also

**CLOSE\_SESSION** (page 396), **CLOSE\_ALL\_SESSIONS** (page 399), **EXECUTION\_ENGINE\_PROFILES** (page 849), **QUERY\_PROFILES** (page 870), **SESSION\_PROFILES** (page 886), and **SESSIONS** (page 888)

Managing Sessions and Configuration Parameters in the Administrator's Guide

## DATA\_COLLECTOR

Shows the data collector components, their current retention policies, and statistics about how much data is retained and how much has been discarded for various reasons. This table also calculates the approximate collection rate, to aid in sizing calculations.

Querying the DATA\_COLLECTOR table is the most convenient way to get a list of all supported and enabled collection components.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name on which information is retained.
COMPONENT	VARCHAR	The name of the component and its policy.
TABLE_NAME	VARCHAR	The data collector (dc) table name for which information is listed.
DESCRIPTION	VARCHAR	A short description about the component.
IN_DB_LOG	BOOLEAN	Denotes if monitoring information is retained in the db.log file.
IN_VERTICA_LOG	BOOLEAN	Denotes if monitoring information is retained in the vertica.log file.
MEMORY_BUFFER_SIZE_KB	INTEGER	The number of retired files.
DISK_SIZE_KB	INTEGER	The size of the memory buffer in kilobytes.
RECORD_TOO_BIG_ERRORS	INTEGER	A number that increments by one each time an error is thrown because data did not fit in memory (based on the data collector retention policy).
LOST_BUFFERS	INTEGER	The number of buffers lost.
LOST_RECORDS	INTEGER	The number of records lost.
RETIRED_FILES	INTEGER	The number of retired files.
RETIRED_RECORDS	INTEGER	The number of retired records.
CURRENT_MEMORY_RECORDS	INTEGER	The current number of rows in memory.
CURRENT_DISK_RECORDS	INTEGER	The current number of rows stored on disk.
CURRENT_MEMORY_BYTES	INTEGER	Total current memory used in kilobytes.
CURRENT_DISK_BYTES	INTEGER	Total current disk space used in kilobytes.
FIRST_TIME	TIMESTAMP	Timestamp of the first record.
LAST_TIME	TIMESTAMP	Timestamp of the last record
KB_PER_DAY	INTEGER	Total kilobytes used per day.

## Notes

- Data Collector is on by default, but database administrators can disable it, under **Technical Support** (on page 1) guidance, if they notice performance issues.
- The monitoring information retention policy is configurable. See **Data Collector Functions** (page 490) in this guide and Configuring Data Retention Policies in the Troubleshooting Guide

## Examples

The following command, which queries all columns in the DATA\_COLLECTOR system table, is truncated for brevity:

```
=> \x
=> SELECT * FROM data_collector;

-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0003
component          | AllocationPoolStatistics
table_name         | dc_allocation_pool_statistics
description        | Information about global memory pools, ...
in_db_log          | f
in_vertica_log     | f
memory_buffer_size_kb | 64
disk_size_kb       | 256
record_too_big_errors | 0
lost_buffers       | 0
lost_records       | 0
retired_files      | 150
retired_records    | 66318
current_memory_records | 0
current_disk_records | 1582
current_memory_bytes | 0
current_disk_bytes | 234536
first_time         | 2011-05-26 13:19:01.006121-04
last_time          | 2011-05-26 13:25:36.004994-04
kb_per_day         | 50014.1761771333
-[ RECORD 2 ]-----+-----
node_name          | v_vmartdb_node0003
component          | AllocationPoolStatisticsBySecond
table_name         | dc_allocation_pool_statistics_by_second
description        | Information about global memory pools, ...
in_db_log          | f
in_vertica_log     | f
memory_buffer_size_kb | 64
disk_size_kb       | 256
record_too_big_errors | 0
lost_buffers       | 0
lost_records       | 0
retired_files      | 648
retired_records    | 66180
current_memory_records | 0
current_disk_records | 349
current_memory_bytes | 0
current_disk_bytes | 222742
first_time         | 2011-05-26 13:24:09.002271-04
last_time          | 2011-05-26 13:25:36.005041-04
kb_per_day         | 214367.571703045
-[ RECORD 3 ]-----+-----
...
```

The following command returns all component names and their descriptions. This is a useful query if you want to change the retention policy for a particular component and don't remember its name:

```
=> SELECT DISTINCT component, description FROM data_collector ORDER BY 1 ASC;
```

component	description
AllocationPoolStatistics	Information about global memory pools ...
AllocationPoolStatisticsByDay	Information about global memory pools, ... (historical, by day)
AllocationPoolStatisticsByHour	Information about global memory pools, ... (historical, by hour)
AllocationPoolStatisticsByMinute	Information about global memory pools, ... (historical, by minute)
AllocationPoolStatisticsBySecond	Information about global memory pools, ... (historical, by second)
AnalyzeStatistics	History of statistics collection

Backups	Monitoring successful backups
CatalogInfo	Catalog statistics and history
CatalogInfoByDay	Catalog statistics and history (historical, by day)
CatalogInfoByHour	Catalog statistics and history (historical, by hour)
CatalogInfoByMinute	Catalog statistics and history (historical, by minute)
CatalogInfoBySecond	Catalog statistics and history (historical, by second)
ClientServerMessages	Client-Server Messages (Front End to Back End Protocol) sent
ConfigurationChanges	Changes to configuration parameters (vertica.conf)
CpuAggregate	Aggregate CPU information
CpuAggregateByDay	Aggregate CPU information (historical, by day)
CpuAggregateByHour	Aggregate CPU information (historical, by hour)
CpuAggregateByMinute	Aggregate CPU information (historical, by minute)
CpuAggregateBySecond	Aggregate CPU information (historical, by second)
CpuInfo	CPU information
CpuInfoByDay	CPU information (historical, by day)
CpuInfoByHour	CPU information (historical, by hour)
CpuInfoByMinute	CPU information (historical, by minute)
CpuInfoBySecond	CPU information (historical, by second)
DeploymentsCompleted	History of designs deployed
DesignsCompleted	History of designs executed
DiskResourceRejections	Disk Resource Rejection Records
Errors	History of all errors+warnings encountered
ExecutionEngineEvents	History of important events during local planning and execution
ExecutionEngineProfiles	History of EE profiles
IoInfo	Information about device IOs
IoInfoByDay	Information about device IOs (historical, by day)
IoInfoByHour	Information about device IOs (historical, by hour)
IoInfoByMinute	Information about device IOs (historical, by minute)
IoInfoBySecond	Information about device IOs (historical, by second)
LockAttempts	History of lock attempts (resolved requests)
LockReleases	History of lock releases
LockRequests	History of lock requests
LoginFailures	Failed login attempts
MemoryInfo	Information about node memory allocation, at the OS level
MemoryInfoByDay	Information about node memory allocation, ... (historical, by day)
MemoryInfoByHour	Information about node memory allocation, ... (historical, by
hour)	
MemoryInfoByMinute	Information about node memory allocation, ... (historical, by
minute)	
MemoryInfoBySecond	Information about node memory allocation, ... (historical, by
second)	
MonitoringEventsCleared	Monitoring events cleared
MonitoringEventsPosted	Monitoring events posted
NetworkInfo	Network interface information and statistics
NetworkInfoByDay	Network interface information and statistics (historical, by day)
NetworkInfoByHour	Network interface information and statistics (historical, by
hour)	
NetworkInfoByMinute	Network interface information and statistics (historical, by
minute)	
NetworkInfoBySecond	Network interface information and statistics (historical, by
second)	
NodeState	History of all node state changes
OptimizerEvents	History of important events during optimizer planning
OptimizerStats	History of optimizer runtime statistics
ProcessInfo	Information about vertica process memory, handles and system
limits	
ProcessInfoByDay	Information about vertica process memory, ... (historical, by day)
ProcessInfoByHour	Information about vertica process memory, ... (historical, by
hour)	
ProcessInfoByMinute	Information about vertica process memory, ... (historical, by
minute)	
ProcessInfoBySecond	Information about vertica process memory, ... (historical, by
second)	
ProjectionRecoveries	Monitoring completed projection recoveries
ProjectionRefreshesCompleted	History of refreshed projections
ProjectionsUsed	Projections used in each SQL request issued
RebalancedSegments	History of all segments rebalanced (EC)
RequestsCompleted	History of all SQL requests completed

RequestsIssued	History of all SQL requests issued
RequestsRetried	History of all SQL requests issued that were retried
ResourceAcquisitions	History of all resource acquisitions
ResourceRejections	Resource Rejection Records
ResourceReleases	History of all resource acquisition releases
RosesCreated	History of all ROS and DVROS created
RosesDestroyed	History of all ROS destroyed
SessionEnds	Sessions ended
SessionStarts	Sessions started
Signals	History of process signals received
Startups	History of all node startup events
StorageInfo	Storage information (Used and Free space)
StorageInfoByDay	Storage information (Used and Free space) (historical, by day)
StorageInfoByHour	Storage information (Used and Free space) (historical, by hour)
StorageInfoByMinute	Storage information (Used and Free space) (historical, by minute)
StorageInfoBySecond	Storage information (Used and Free space) (historical, by second)
StorageLayerStatistics	Statistics and history of storage and caching layer
StorageLayerStatisticsByDay	Statistics and history of storage and caching layer (historical, by day)
StorageLayerStatisticsByHour	Statistics and history of storage and caching layer (historical, by hour)
StorageLayerStatisticsByMinute	Statistics and history of storage and caching layer (historical, by minute)
StorageLayerStatisticsBySecond	Statistics and history of storage and caching layer (historical, by second)
Test	For data collector infrastructure testing
TransactionEnds	History of end transactions (commit or rollback)
TransactionStarts	History of begin transactions
TuningAnalysis	Tuning analysis history in Workload Analyzer
TuningRecommendations	Tuning Recommendations in Workload Analyzer
TupleMoverEvents	History of Tuple Mover activities
Upgrades	Monitoring catalog upgrades
UserAudits	History of user audits
(93 rows)	

**See Also**

**Data Collector Functions** (page 490)

Retaining Monitoring Information in the Troubleshooting Guide

**DATABASE\_CONNECTIONS**

Lists the connections that have been established to other databases for importing and exporting data. See Importing and Exporting Data in the Administrator's Guide.

Column Name	Data Type	Description
DATABASE	VARCHAR	The name of the connected database
USERNAME	VARCHAR	The username used to create the connection
HOST	VARCHAR	The host name used to create the connection
PORT	VARCHAR	The port number used to create the connection
ISVALID	BOOLEAN	Whether the connection is still open and usable or not

**Example**

```
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD '' ON '10.10.20.150',5433;
CONNECT
=> SELECT * FROM DATABASE_CONNECTIONS;
  database | username |      host      | port | isvalid
-----+-----+-----+-----+-----
  vmart    | dbadmin  | 10.10.20.150  | 5433 | t
(1 row)
```

**DATABASE\_SNAPSHOTS**

Monitors the space consumed by stored database snapshots.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The name of the node storing the snapshot
SNAPSHOT_NAME	VARCHAR	The name of the snapshot
IS_DURABLE_SNAPSHOT	BOOLEAN	Indicates whether the snapshot is non-transient
TOTAL_SIZE_BYTES	INTEGER	The size of the snapshot if it were copied off of the node
STORAGE_COST_BYTES	INTEGER	The actual amount of disk space used by the snapshot
ACQUISITION_TIMESTAMP	TIMESTAMP	The time the snapshot was created

**Example**

The following example produces a durable (non-transient) snapshot under Catalog/Snapshots/2011-05-17-Snapshot/ and Catalog/Snapshots/2011-05-17-Snapshot.ctlg.

```
=> SELECT DATABASE_SNAPSHOT('mysnapshot',true);
          DATABASE_SNAPSHOT
```

```
-----
v_vmartdb_node01,v_vmartdb_node02,v_vmartdb_node03,v_vmartdb_node04
(1 row)
```

Now you can query the DATABASE\_SNAPSHOTS table to look at the space the snapshot consumes:

```
=> SELECT * FROM database_snapshots;
```

```
vmartdb=> SELECT * FROM database_snapshots;
  node_name      | snapshot_name | is_durable_snapshot | total_size_bytes | storage_cost_bytes |
acquisition_timestamp
-----+-----+-----+-----+-----+-----
v_vmartdb_node0001 | mysnapshot    | t                    | 116108615        | 5054638            |
2010-10-07 12:39:22-04
v_vmartdb_node0002 | mysnapshot    | t                    | 116385001        | 5066175            |
```

```

2010-10-07 12:39:35-04
v_vmartdb_node0003 | mysnapshot | t | 116379703 | 5054692 |
2010-10-07 12:38:00-04
v_vmartdb_node0004 | mysnapshot | t | 116354638 | 5043155 |
2010-10-07 12:36:10-04
(4 rows)

```

## DELETE\_VECTORS

Holds information on deleted rows to speed up the delete process.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The name of the node storing the deleted rows.
SCHEMA_NAME	VARCHAR	The name of the schema where the deleted rows are located.
PROJECTION_NAME	VARCHAR	The name of the projection where the deleted rows are located.
STORAGE_TYPE	VARCHAR	The type of storage containing the delete vector (WOS or ROS).
DV_OID	INTEGER	The unique numeric ID (OID) that identifies this delete vector.
STORAGE_OID	INTEGER	The unique numeric ID (OID) that identifies the storage container that holds the delete vector.
DELETED_ROW_COUNT	INTEGER	The number of rows deleted.
USED_BYTES	INTEGER	The number of bytes used to store the deletion.
START_EPOCH	INTEGER	The start epoch of the data in the delete vector.
END_EPOCH	INTEGER	The end epoch of the data in the delete vector.

## DISK\_RESOURCE\_REJECTIONS

Returns requests for resources that are rejected due to disk space shortages.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
RESOURCE_TYPE	VARCHAR	The resource request requester (example: Temp files).
REJECTED_REASON	VARCHAR	One of 'Insufficient disk space' or 'Failed volume'.
REJECTED_COUNT	INTEGER	Number of times this REJECTED_REASON has been given for this RESOURCE_TYPE.

FIRST_REJECTED_TIMESTAMP	TIMESTAMP	The time of the first rejection for this REJECTED_REASON and RESOURCE_TYPE.
LAST_REJECTED_TIMESTAMP	TIMESTAMP	The time of the most recent rejection for this REJECTED_REASON and RESOURCE_TYPE.
LAST_REJECTED_VALUE	INTEGER	The value of the most recent rejection for this REJECTED_REASON and RESOURCE_TYPE.

**Notes**

Output is aggregated by both RESOURCE\_TYPE and REJECTED\_REASON to provide more comprehensive information.

**Example**

```
=>\pset expanded
Expanded display on.
=> SELECT * FROM disk_resource_rejections;
-[ RECORD 1 ]-----+-----
node_name          | e0
resource_type      | Table Data
rejected_reason    | Insufficient disk space
rejected_count     | 2
first_rejected_timestamp | 2009-10-16 15:55:16.336246
last_rejected_timestamp | 2009-10-16 15:55:16.336391
last_rejected_value | 1048576
-[ RECORD 2 ]-----+-----
node_name          | e1
resource_type      | Table Data
rejected_reason    | Insufficient disk space
rejected_count     | 2
first_rejected_timestamp | 2009-10-16 15:55:16.37908
last_rejected_timestamp | 2009-10-16 15:55:16.379207
last_rejected_value | 1048576
```

**See Also**

**RESOURCE\_REJECTIONS** (page 882)

**CLEAR\_RESOURCE\_REJECTIONS** (page 396)

Managing Workloads and Managing System Resource Usage in the Administrator's Guide

**DISK\_STORAGE**

Returns the amount of disk storage used by the database on each node.

Column Name	Date Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
STORAGE_PATH	VARCHAR	The path where the storage location is mounted.

STORAGE_USAGE	VARCHAR	The type of information stored in the location: <ul style="list-style-type: none"> <li>▪ DATA: Only data is stored in the location.</li> <li>▪ TEMP: Only temporary files that are created during loads or queries are stored in the location.</li> <li>▪ DATA,TEMP: Both types of files are stored in the location.</li> <li>▪ CATALOG</li> </ul>
RANK	INTEGER	The rank assigned to the storage location based on its performance. Ranks are used to create a tiered disk architecture in which projections, columns, and partitions are stored on different disks based on predicted or measured access patterns. See <i>Creating and Configuring Storage Locations</i> in the <i>Administrator's Guide</i> .
THROUGHPUT	INTEGER	The measure of a storage location's performance in MB/sec. 1/throughput is the time taken to read 1MB of data.
LATENCY	INTEGER	The measure of a storage location's performance in seeks/sec. 1/latency is the time taken to seek to the data.
STORAGE_STATUS	VARCHAR	The status of the storage location: active or retired.
DISK_BLOCK_SIZE_BYTES	INTEGER	The block size of the disk in bytes.
DISK_SPACE_USED_BLOCKS	INTEGER	The number of disk blocks in use.
DISK_SPACE_USED_MB	INTEGER	The number of megabytes of disk storage in use.
DISK_SPACE_FREE_BLOCKS	INTEGER	The number of free disk blocks available.
DISK_SPACE_FREE_MB	INTEGER	The number of megabytes of free storage available.
DISK_SPACE_FREE_PERCENT	INTEGER	The percentage of free disk space remaining.

## Notes

- All values returned are in the context of the operating system's filesystems and are not specific to Vertica-specific space.
- The storage usage annotation called CATALOG indicates the location is used to store the catalog. However, CATALOG location can only be specified when creating a new database and no new locations can be added as CATALOG locations using **ADD\_LOCATION** (page 374). Existing CATALOG annotations cannot be removed.
- A storage location's performance is measured in throughput in MB/sec and latency in seeks/sec. These two values are converted to single number(Speed) with the following formula:

ReadTime (time to read 1MB) = 1/throughput + 1 / latency

- 1/throughput is the time taken to read 1MB of data
- 1/latency is the time taken to seek to the data.
- ReadTime is the time taken to read 1MB of data.

- A disk is faster than another disk if its ReadTime is less.
- There can be multiple storage locations per node, and these locations can be on different disks with different free/used space, block size, etc. This information is useful in letting you know where the data files reside.

### Example

Query the DISK\_STORAGE table:

```
=>\pset expanded
```

```
Expanded display is on.
```

```
=> SELECT * FROM DISK_STORAGE;
```

```
-[ RECORD 1 ]-----+-----  
current_timestamp      | 2009-08-11 14:48:35.932541  
node_name              | site01  
storage_path           | /mydb/node01_catalog/Catalog  
storage_usage          | CATALOG  
rank                   | 0  
throughput             | 0  
latency                | 0  
storage_status         | Active  
disk_block_size_bytes | 4096  
disk_space_used_blocks | 34708721  
disk_space_used_mb     | 135581  
disk_space_free_blocks | 178816678  
disk_space_free_mb     | 698502  
disk_space_free_percent | 83%  
-----+-----  
-[ RECORD 2 ]-----+-----  
current_timestamp      | 2009-08-11 14:48:53.884255  
node_name              | site01  
storage_path           | /mydb/node01_data  
storage_usage          | DATA,TEMP  
rank                   | 0  
throughput             | 0  
latency                | 0  
storage_status         | Active  
disk_block_size_bytes | 4096  
disk_space_used_blocks | 34708721  
disk_space_used_mb     | 135581  
disk_space_free_blocks | 178816678  
disk_space_free_mb     | 698502  
disk_space_free_percent | 83%  
-----+-----  
-[ RECORD 3 ]-----+-----  
current_timestamp      | 2009-08-11 14:49:08.299012  
node_name              | site02  
storage_path           | /mydb/node02_catalog/Catalog  
storage_usage          | CATALOG  
rank                   | 0  
throughput             | 0  
latency                | 0  
storage_status         | Active  
disk_block_size_bytes | 4096  
disk_space_used_blocks | 19968349
```

SQL System Tables (Monitoring APIs)

```

disk_space_used_mb      | 78001
disk_space_free_blocks | 193557050
disk_space_free_mb     | 756082
disk_space_free_percent | 90%
-[ RECORD 4 ]-----+-----
current_timestamp      | 2009-08-11 14:49:22.696772
node_name              | site02
storage_path          | /mydb/node02_data
storage_usage         | DATA,TEMP
rank                  | 0
throughput            | 0
latency               | 0
storage_status        | Active
disk_block_size_bytes | 4096
disk_space_used_blocks | 19968349
disk_space_used_mb    | 78001
disk_space_free_blocks | 193557050
disk_space_free_mb    | 756082
disk_space_free_percent | 90%
-[ RECORD 5 ]-----+-----
current_timestamp      | 2009-08-11 14:50:03.960157
node_name              | site03
storage_path          | /mydb/node03_catalog/Catalog
storage_usage         | CATALOG
rank                  | 0
throughput            | 0
latency               | 0
storage_status        | Active
disk_block_size_bytes | 4096
disk_space_used_blocks | 19902595
disk_space_used_mb    | 77744
disk_space_free_blocks | 193622804
disk_space_free_mb    | 756339
disk_space_free_percent | 90%
-[ RECORD 6 ]-----+-----
current_timestamp      | 2009-08-11 14:50:27.415735
node_name              | site03
storage_path          | /mydb/node03_data
storage_usage         | DATA,TEMP
rank                  | 0
throughput            | 0
latency               | 0
storage_status        | Active
disk_block_size_bytes | 4096
disk_space_used_blocks | 19902595
disk_space_used_mb    | 77744
disk_space_free_blocks | 193622804
disk_space_free_mb    | 756339
disk_space_free_percent | 90%
-[ RECORD 7 ]-----+-----
current_timestamp      | 2009-08-11 14:50:39.398879
node_name              | site04
storage_path          | /mydb/node04_catalog/Catalog
storage_usage         | CATALOG
rank                  | 0

```

```
throughput          | 0
latency             | 0
storage_status      | Active
disk_block_size_bytes | 4096
disk_space_used_blocks | 19972309
disk_space_used_mb   | 78017
disk_space_free_blocks | 193553090
disk_space_free_mb   | 756066
disk_space_free_percent | 90%
-[ RECORD 8 ]-----+-----
current_timestamp   | 2009-08-11 14:50:57.879302
node_name           | site04
storage_path        | /mydb/node04_data
storage_usage       | DATA,TEMP
rank                | 0
throughput          | 0
latency             | 0
storage_status      | Active
disk_block_size_bytes | 4096
disk_space_used_blocks | 19972309
disk_space_used_mb   | 78017
disk_space_free_blocks | 193553090
disk_space_free_mb   | 756066
disk_space_free_percent | 90%
```

Request only specific columns from the table:

```
=> SELECT node_name, storage_path, storage_status, disk_space_free_percent FROM disk_storage;
```

```
node_name | storage_path | storage_status | disk_space_free_percent
-----+-----+-----+-----
site01    | /mydb/node01_catalog/Catalog | Active | 83%
site01    | /mydb/node01_data | Active | 83%
site02    | /mydb/node02_catalog/Catalog | Active | 90%
site02    | /mydb/node02_data | Active | 90%
site03    | /mydb/node03_catalog/Catalog | Active | 90%
site03    | /mydb/node03_data | Active | 90%
site04    | /mydb/node04_catalog/Catalog | Active | 90%
site04    | /mydb/node04_data | Active | 90%
(8 rows)
```

## EVENT\_CONFIGURATIONS

Monitors the configuration of events.

Column Name	Date Type	Description
EVENT_ID	VARCHAR	The name of the event.
EVENT_DELIVERY_CHANNELS	VARCHAR	The delivery channel on which the event occurred.

### Example

```
=> SELECT * FROM event_configurations;
```

```
event_id | event_delivery_channels
-----+-----
```

Low Disk Space	Vertica Log, SNMP Trap
Read Only File System	Vertica Log, SNMP Trap
Loss Of K Safety	Vertica Log, SNMP Trap
Current Fault Tolerance at Critical Level	Vertica Log, SNMP Trap
Too Many ROS Containers	Vertica Log, SNMP Trap
WOS Over Flow	Vertica Log, SNMP Trap
Node State Change	Vertica Log, SNMP Trap
Recovery Failure	Vertica Log, SNMP Trap
Recovery Error	Vertica Log
Recovery Lock Error	Vertica Log
Recovery Projection Retrieval Error	Vertica Log
Refresh Error	Vertica Log
Refresh Lock Error	Vertica Log
Tuple Mover Error	Vertica Log
Timer Service Task Error	Vertica Log
Stale Checkpoint	Vertica Log, SNMP Trap

(16 rows)

## EXECUTION\_ENGINE\_PROFILES

Provides information regarding query execution runs.

For additional details about profiling and debugging, see Profiling Database Performance in the Troubleshooting Guide.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
SESSION_ID	VARCHAR	The identification of the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
TRANSACTION_ID	INTEGER	An identifier for the transaction within the session if any; otherwise NULL.
STATEMENT_ID	INTEGER	A unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed.
USER_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the user.
USER_NAME	VARCHAR	The user name for which query profile information is listed.
OPERATOR_NAME	VARCHAR	The name of the Execution Engine component; for example, NetworkSend.
OPERATOR_ID	INTEGER	The ID of the Execution Engine component.
BASEPLAN_ID	INTEGER	The ID in the original plan by the optimizer (EXPLAIN plan).
PATH_ID	INTEGER	An ID that correlates EXPLAIN plans with profiling counters and join error messages.

LOCALPLAN_ID	INTEGER	The ID in the plan that was actually executed (EXPLAIN LOCAL plan).
COUNTER_NAME	VARCHAR	The name of the counter. See the "COUNTER_NAME Values" section below this table.
COUNTER_TAG	VARCHAR	A string that uniquely identifies the counter for operators that might need to distinguish between different instances. For example, COUNTER_TAG is used to identify to which of the node bytes are being sent to or received from for the NetworkSend operator.
COUNTER_VALUE	INTEGER	The value of the counter.
IS_EXECUTING	VARCHAR	Distinguishes between active and completed profiles.

### COUNTER\_NAME Values

The value of COUNTER\_NAME can be any of the following:

COUNTER_NAME	Description
buffers spilled	[NetworkSend] Buffers spilled to disk by NetworkSend.
bytes received	[NetworkRecv] The number of bytes received over the network for query execution.
bytes sent	[NetworkSend] Size of data after encoding and compression sent over the network (actual network bytes).
bytes spilled	[NetworkSend] Bytes spilled to disk by NetworkSend.
bytes total	Only relevant to SendFiles operator (that is, recover-by-container plan) total number of bytes to send / receive.
clock time (µs)	Real-time clock in microseconds. <b>Note:</b> This counter was called execution time (us) in previous Vertica releases.
completed merge phases	Number of merge phases already completed by an LSort or DataTarget operator. Compare to the total merge phases. Variants on this value include "join inner completed merge phases."
cumulative size of raw temp data (bytes)	Compare to cumulative size of temp files (bytes) to understand impact of encoding and compression in an externalizing operator. Variants on this value include join inner cumulative size of raw temp files (bytes).
cumulative size of temp files (bytes)	For externalizing operators only, the total number of bytes the operator has written to temp files. A sort operator might go through multiple merge phases, where at each pass sorted chunks of data are merged into fewer chunks. This counter remembers the cumulative size of all temp files past and present. Variants on this value include join inner cumulative size of temp files (bytes).

current size of temp files (bytes)	For externalizing operators only, the current size of the operator's temp files in bytes. Variants on this value include <code>join inner current size of temp files (bytes)</code> .
encoded bytes received	[NetworkRecv] Size of received data after decompressed (but still encoded) received over the network.
encoded bytes sent	[NetworkSend] Size of data sent over the network after encoding.
executable time (ms)	Thread CPU clock time in milliseconds.
files completed	Relevant only to <code>SendFiles/RecvFiles</code> operators (that is, recover-by-container plan) number of files sent / received.
file handles	Number of file handles used by the operator.
files total	Relevant only to <code>SendFiles/RecvFiles</code> operators (that is, recover-by-container plan) total number of files to send / receive.
input queue wait ( $\mu$ s)	Time in microseconds that an operator spends waiting for upstream operators.
input size (bytes)	Total number of bytes of the <code>Load</code> operator's input source, where NULL is unknown (read from FIFO).
memory allocated (bytes)	Actual memory in bytes that the operator allocated at run time.
memory reserved (bytes)	Memory reserved by the operator in the <code>ResourceManager</code> operator. <b>Note:</b> An allocation slightly more than the reservation (a few MB) is not a cause for concern and is built into <code>ResourceManager</code> calculations.
network wait ( $\mu$ s)	[NetworkSend, NetworkRecv] Time in microseconds spent waiting on the network.
output queue wait ( $\mu$ s)	Time in microseconds that an operator spends waiting for the output buffer to be consumed by a downstream operator.
phj sort cumulative size of raw temp files (bytes)	Less common counter for partitioned hash join. <b>Note:</b> Used only for joins of dynamically-redistributed data.
phj inner cumulative size of raw temp files (bytes)	
phj outer cumulative size of raw temp files (bytes)	
phj sort cumulative size of temp files (bytes)	
phj inner cumulative size of temp files (bytes)	
phj outer cumulative size of temp files (bytes)	
phj sort current size of temp files (bytes)	
phj inner current size of	

temp files (bytes)	
phj outer current size of temp files (bytes)	
phj total partitions	Total number of partitions created. This counter is null until all input is consumed and the number of partitions is known.
phj completed partitions	Number of partitions that have been processed.
producer stall (µs)	[NetworkSend] Time in microseconds spent by NetworkSend when stalled waiting for network buffers to clear.
producer wait (µs)	[NetworkSend] Time in microseconds spent by the input operator making rows to send.
rle rows produced	Number of physical tuples produced by an operator. Complements the rows produced counter, which shows the number of logical rows produced by an operator. For example, if a value occurs 1000 rows consecutively and is RLE encoded, it counts as 1000 rows produced not only 1 rle rows produced.
read (bytes)	Number of bytes read from the input source by the Load operator.
receive time (µs)	Time in microseconds that a Recv operator spends reading data from its socket.
rows produced	Number of logical rows produced by an operator. See also the rle rows produced counter.
rows received	[NetworkRecv] Number of received sent over the network.
rows rejected	The number of rows rejected by the Load operator.
rows sent	[NetworkSend] Number of rows sent over the network.
send time (µs)	Time in microseconds that a Send operator spends writing data to its socket.
total merge phases	Number of merge phases an LSort or DataTarget operator must complete to finish sorting its data. NULL until the operator can compute this value (all data must first be ingested by the operator). Variants on this value include join inner total merge phases.
wait clock time (µs)	StorageUnion wait time in microseconds.
wait execution time (µs)	
WOS bytes acquired	Number of bytes acquired from the WOS by a DataTarget operator. <b>Note:</b> This is usually more but can be less than WOS bytes written if an earlier statement in the transaction acquired some WOS memory.
WOS bytes written	Number of bytes written to the WOS by a DataTarget operator.

### Example

```
=> SELECT operator_name, operator_id, counter_name, counter_value
       FROM EXECUTION_ENGINE_PROFILES WHERE operator_name = 'Scan'
       ORDER BY counter_value DESC;
```

operator_name	operator_id	counter_name	counter_value
Scan	6	memory allocated (bytes)	15688
Scan	6	estimated rows produced	9999
Scan	6	estimated rows produced	9999
Scan	6	memory allocated (bytes)	2152
Scan	6	clock time (us)	572
Scan	6	execution time (us)	187
Scan	6	clock time (us)	0
Scan	6	memory reserved (bytes)	0
Scan	6	file handles	0
Scan	6	rows produced	0
Scan	6	memory reserved (bytes)	0
Scan	6	rows produced	0
Scan	6	file handles	0
Scan	6	execution time (us)	0

(14 rows)

```
=> SELECT DISTINCT counter_name FROM execution_engine_profiles;
       counter_name
```

```
-----
file handles
estimated rows produced
memory reserved (bytes)
clock time (us)
output queue wait (us)
input queue wait (us)
wait clock time (us)
wait execution time (us)
execution time (us)
memory allocated (bytes)
rows produced
```

(11 rows)

The notable thing about the following query is the `path_id` column, which correlates EXPLAIN plan textual output with profiling counters and join error messages.

```
=> SELECT operator_name, path_id, counter_name,
       counter_value FROM execution_engine_profiles;
operator_name | path_id | counter_name | counter_value
-----+-----+-----+-----
Join          | 1      | estimated rows produced | 10000
Join          | 1      | file handles           | 0
Join          | 1      | memory allocated (bytes) | 2405824
Join          | 1      | memory reserved (bytes) | 1769472
Join          | 1      | rle rows produced      | 3
Join          | 1      | rows produced          | 3
Join          | 1      | clock time (us)        | 24105
Join          | 1      | execution time (us)    | 235
```

**See Also**

Profiling Database Performance in the Troubleshooting Guide, particularly Viewing Profiling Data  
 Correlating EXPLAIN Plan Output with Error Messages in the Troubleshooting Guide

**HOST\_RESOURCES**

Provides a snapshot of the node. This is useful for regularly polling the node with automated tools or scripts.

Column Name	Data Type	Description
HOST_NAME	VARCHAR	The host name for which information is listed.
OPEN_FILES_LIMIT	INTEGER	The maximum number of files that can be open at one time on the node.
THREADS_LIMIT	INTEGER	The maximum number of threads that can coexist on the node.
CORE_FILE_LIMIT_MAX_SIZE_BYTES	INTEGER	The maximum core file size allowed on the node.
PROCESSOR_COUNT	INTEGER	The number of system processors.
PROCESSOR_CORE_COUNT	INTEGER	The number of processor cores in the system.
PROCESSOR_DESCRIPTION	VARCHAR	A description of the processor. For example: Inter(R) Core(TM)2 Duo CPU T8100 @2.10GHz (1 row)
OPENED_FILE_COUNT	INTEGER	The total number of open files on the node.
OPENED_SOCKET_COUNT	INTEGER	The total number of open sockets on the node.
OPENED_NONFILE_NONSOCKET_COUNT	INTEGER	The total number of <i>other</i> file descriptions open in which 'other' could be a directory or FIFO. It is not an open file or socket.
TOTAL_MEMORY_BYTES	INTEGER	The total amount of physical RAM, in bytes, available on the system.
TOTAL_MEMORY_FREE_BYTES	INTEGER	The amount of physical RAM, in bytes, left unused by the system.
TOTAL_BUFFER_MEMORY_BYTES	INTEGER	The amount of physical RAM, in bytes, used for file buffers on the system
TOTAL_MEMORY_CACHE_BYTES	INTEGER	The amount of physical RAM, in bytes, used as cache memory on the system.
TOTAL_SWAP_MEMORY_BYTES	INTEGER	The total amount of swap memory available, in bytes, on the system.
TOTAL_SWAP_MEMORY_FREE_BYTES	INTEGER	The total amount of swap memory free, in bytes, on the system.
DISK_SPACE_FREE_MB	INTEGER	The free disk space available, in megabytes, for all storage location file systems (data

		directories).
DISK_SPACE_USED_MB	INTEGER	The disk space used, in megabytes, for all storage location file systems.
DISK_SPACE_TOTAL_MB	INTEGER	The total free disk space available, in megabytes, for all storage location file systems.

## Examples

Query the `HOST_RESOURCES` table:

```
=>\pset expanded
```

```
Expanded display is on.
```

```
=> SELECT * FROM HOST_RESOURCES;
```

```

-[ RECORD 1 ]-----+-----
host_name          | myhost-s1
open_files_limit   | 65536
threads_limit      | 15914
core_file_limit_max_size_bytes | 1649680384
processor_count    | 2
processor_core_count | 8
processor_description | Intel(R) Xeon(R) CPU E5504 @ 2.00GHz
opened_file_count  | 5
opened_socket_count | 4
opened_nonfile_nonssocket_count | 3
total_memory_bytes | 16687161344
total_memory_free_bytes | 4492627968
total_buffer_memory_bytes | 1613922304
total_memory_cache_bytes | 9349111808
total_swap_memory_bytes | 36502126592
total_swap_memory_free_bytes | 36411580416
disk_space_free_mb | 121972
disk_space_used_mb | 329235
disk_space_total_mb | 451207
-[ RECORD 2 ]-----+-----
host_name          | myhost-s2
open_files_limit   | 65536
threads_limit      | 15914
core_file_limit_max_size_bytes | 3772891136
processor_count    | 2
processor_core_count | 4
processor_description | Intel(R) Xeon(R) CPU E5504 @ 2.00GHz
opened_file_count  | 5
opened_socket_count | 3
opened_nonfile_nonssocket_count | 3
total_memory_bytes | 16687161344
total_memory_free_bytes | 9525706752
total_buffer_memory_bytes | 2840420352
total_memory_cache_bytes | 3060588544
total_swap_memory_bytes | 34330370048
total_swap_memory_free_bytes | 34184642560
disk_space_free_mb | 822190
disk_space_used_mb | 84255
disk_space_total_mb | 906445
-[ RECORD 3 ]-----+-----
host_name          | myhost-s3
open_files_limit   | 65536
threads_limit      | 15914
core_file_limit_max_size_bytes | 3758211072
processor_count    | 2
processor_core_count | 4
processor_description | Intel(R) Xeon(R) CPU E5504 @ 2.00GHz

```

```

opened_file_count          | 5
opened_socket_count       | 3
opened_nonfile_nonsocket_count | 3
total_memory_bytes        | 16687161344
total_memory_free_bytes   | 9718706176
total_buffer_memory_bytes | 2928369664
total_memory_cache_bytes  | 2757115904
total_swap_memory_bytes   | 34315689984
total_swap_memory_free_bytes | 34205523968
disk_space_free_mb        | 820789
disk_space_used_mb        | 85640
disk_space_total_mb       | 906429
-[ RECORD 4 ]-----+-----
host_name                  | myhost-s4
open_files_limit           | 65536
threads_limit              | 15914
core_file_limit_max_size_bytes | 3799433216
processor_count            | 2
processor_core_count       | 8
processor_description      | Intel(R) Xeon(R) CPU E5504 @ 2.00GHz
opened_file_count          | 5
opened_socket_count       | 3
opened_nonfile_nonsocket_count | 3
total_memory_bytes        | 16687161344
total_memory_free_bytes   | 8772620288
total_buffer_memory_bytes | 3792273408
total_memory_cache_bytes  | 2831040512
total_swap_memory_bytes   | 34356912128
total_swap_memory_free_bytes | 34282590208
disk_space_free_mb        | 818896
disk_space_used_mb        | 55291
disk_space_total_mb       | 874187

```

## LOAD\_STREAMS

Monitors load metrics for each load stream on each node.

Column Name	Date Type	Description
SESSION_ID	INTEGER	The identification of the session for which load streams information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
TRANSACTION_ID	INTEGER	An identifier for the transaction within the session if any; otherwise NULL.
STATEMENT_ID	INTEGER	A unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed.
STREAM_NAME	VARCHAR	The optional identifier that names a stream, if specified.
SCHEMA_NAME	VARCHAR	The schema name for which load stream information is listed.
TABLE_NAME	VARCHAR	The name of the table being loaded.
LOAD_START	VARCHAR	The Linux system time when the load started.

ACCEPTED_ROW_COUNT	INTEGER	The number of rows loaded.
REJECTED_ROW_COUNT	INTEGER	The number of rows rejected.
READ_BYTES	INTEGER	The number of bytes read from the input file.
INPUT_FILE_SIZE_BYTES	INTEGER	The size of the input file in bytes. <b>Note:</b> When using STDIN as input size of input file size is zero (0).
PARSE_COMPLETE_PERCENT	INTEGER	The percent of the rows in the input file that have been loaded.
UNSORTED_ROW_COUNT	INTEGER	The cumulative number rows not sorted across all projections. <b>Note:</b> UNSORTED_ROW_COUNT could be greater than ACCEPTED_ROW_COUNT because data is copied and sorted for every projection in the target table.
SORTED_ROW_COUNT	INTEGER	The cumulative number of rows sorted across all projections.
SORT_COMPLETE_PERCENT	INTEGER	The percent of the rows in the input file that have been sorted.

## Notes

If a COPY ... DIRECT operation is in progress, the ACCEPTED\_ROW\_COUNT field could increase up to the maximum number of rows in the input file as the rows are being parsed. If COPY reads from many named pipes, PARSE\_COMPLETE\_PERCENT shows 0 until it receives an EOF from *all* named pipes. This can take a significant amount of time, and it is easy to mistake this state as a hang. Check your system CPU and disk accesses to determine if any activity is in progress before canceling COPY or reporting a hang.

In a typical load, you might notice PARSE\_COMPLETE\_PERCENT creep up to 100% or jump to 100% if loading from named pipes or STDIN, while SORT\_COMPLETE\_PERCENT is at 0. Once PARSE\_COMPLETE\_PERCENT reaches 100%, SORT\_COMPLETE\_PERCENT creeps up to 100%. Depending on the data sizes, there could be significant lag between the time PARSE\_COMPLETE\_PERCENT reaches 100% and the time SORT\_COMPLETE\_PERCENT begins to increase.

## Example

```
=> \pset expanded
Expanded display is on.
```

```
=> SELECT stream_name, table_name, load_start, accepted_row_count,
        rejected_row_count, read_bytes, unsorted_row_count, sorted_row_count,
        sort_complete_percent FROM load_streams;
```

```
-[ RECORD 1 ]-----+-----
stream_name      | fact-13
table_name       | fact
load_start       | 2010-12-28 15:07:41.132053
accepted_row_count | 900
```

```

rejected_row_count | 100
read_bytes         | 11975
input_file_size_bytes | 0
parse_complete_percent | 0
unsorted_row_count | 3600
sorted_row_count   | 3600
sort_complete_percent | 100

```

## LOCKS

Monitors lock grants and requests for all nodes.

Column Name	Date Type	Description
NODE_NAMES	VARCHAR	The nodes on which lock interaction occurs. <b>Note on node rollup:</b> If a transaction has the same lock in the same mode in the same scope on multiple nodes, it gets one (1) line in the table. NODE_NAMES are separated by commas.
OBJECT_NAME	VARCHAR	Name of object being locked; can be a table or an internal structure (projection, global catalog, or local catalog).
OBJECT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the object being locked.
TRANSACTION_DESCRIPTION	VARCHAR	ID of transaction and associated description, typically the query that caused the transaction's creation.
LOCK_MODE	VARCHAR	Describes the intended operations of the transaction: <ul style="list-style-type: none"> <li>▪ S — Share lock needed for select operations</li> <li>▪ I — Insert lock needed for insert operations</li> <li>▪ X — Exclusive lock is always needed for delete operations. Xlock is also the result of lock promotion (see Table 2)</li> <li>▪ T — Tuple Mover lock used by the Tuple Mover and also used for COPY into pre-join projections</li> <li>▪ U — Usage lock needed for moveout and mergeout operations in the first phase; they then upgrade their U lock to a T lock for the second phase. U locks conflicts with no other locks but O.</li> <li>▪ O — Owner lock needed for DROP_PARTITION, TRUNCATE TABLE, and ADD COLUMN. O locks conflict with all locks. O locks never promote.</li> </ul>
LOCK_SCOPE	VARCHAR	The expected duration of the lock once it is granted. Before the lock is granted, the scope is listed as

		<p>REQUESTED.</p> <p>Once a lock has been granted, the following scopes are possible:</p> <ul style="list-style-type: none"> <li>▪ STATEMENT_LOCALPLAN</li> <li>▪ STATEMENT_COMPILE</li> <li>▪ STATEMENT_EXECUTE</li> <li>▪ TRANSACTION_POSTCOMMIT</li> <li>▪ TRANSACTION</li> </ul> <p>All scopes, other than TRANSACTION, are transient and are used only as part of normal query processing.</p>
REQUEST_TIMESTAMP	TIMESTAMP	Indicates the time when the transaction started waiting on the lock.
GRANT_TIMESTAMP	TIMESTAMP	<p>Indicates the time the transaction acquired or upgraded the lock.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>▪ Return values are NULL until the grant occurs.</li> <li>▪ Values could be the same as REQUEST_TIMESTAMP if the grant occurs immediately.</li> </ul>

### Notes

- Lock acquisition and wait times are collected and exposed via the START\_TIMESTAMP column.
- Locks acquired on tables that were subsequently dropped by another transaction can result in the message, `Unknown or deleted object`, appearing in the output's OBJECT column.
- If a `SELECT..FROM LOCKS` query times out after five minutes, it is possible the cluster has failed. Run the Diagnostics Utility and contact **Technical Support** (on page 1).
- A table call with no results indicates that no locks are in use.

The following two tables are from *Transaction Processing: Concepts and Techniques* [http://www.amazon.com/gp/product/1558601902/ref=s9sdps\\_c1\\_14\\_at1-rfc\\_p-frt\\_p-3237\\_g1\\_si1?pf\\_rd\\_m=ATVPDKIKX0DER&pf\\_rd\\_s=center-1&pf\\_rd\\_r=1QHH6V589JEV0DR3DQ1D&pf\\_rd\\_t=101&pf\\_rd\\_p=463383351&pf\\_rd\\_i=507846](http://www.amazon.com/gp/product/1558601902/ref=s9sdps_c1_14_at1-rfc_p-frt_p-3237_g1_si1?pf_rd_m=ATVPDKIKX0DER&pf_rd_s=center-1&pf_rd_r=1QHH6V589JEV0DR3DQ1D&pf_rd_t=101&pf_rd_p=463383351&pf_rd_i=507846) by Jim Gray (Figure 7.11, p. 408 and Figure 8.6, p. 467).

**Table 1: Compatibility matrix for granular locks**

This table is for compatibility with other users. The table is symmetric.

Requested Mode	Granted Mode					
	S	I	X	T	U	O
S	Yes	No	No	Yes	Yes	No
I	<b>No</b>	<b>Yes</b>	No	Yes	Yes	No

<b>X</b>	No	No	No	No	Yes	No
<b>T</b>	Yes	Yes	No	Yes	Yes	No
<b>U</b>	Yes	Yes	Yes	Yes	Yes	No
<b>O</b>	No	No	No	No	No	No

The following two examples refer to Table 1 above:

- **Example 1:** If someone else has an S lock, you cannot get an I lock.
- **Example 2:** If someone has an I lock, you can get an I lock.

**Table 2: Lock conversion matrix**

This table is used for upgrading locks you already have. For example, If you have an S lock and you want an I lock, you request an **X** lock. If you have an S lock and you want an S lock, no lock requests is required.

Requested Mode	Granted Mode					
	S	I	X	T	U	O
<b>S</b>	S	X	X	S	S	0
<b>I</b>	<b>X</b>	I	X	I	I	0
<b>X</b>	X	X	X	X	X	0
<b>T</b>	S	I	X	T	T	0
<b>U</b>	S	I	X	T	U	O
<b>O</b>	O	O	O	O	O	O

**Example**

This example shows an INSERT lock in use:

```
=>\pset expanded
Expanded display is on.
=> SELECT * FROM LOCKS;
-[ RECORD 1
]-----+-----
node_names          | node01,node02,node03,node04
object_name         | Table:fact
object_id           | 45035996273772278
transaction_description | Txn: a000000000112b 'COPY fact FROM '/data_dg/fact.dat'
                    | DELIMITER '|' NULL '\\N';'
lock_mode           | I
lock_scope          | TRANSACTION
request_timestamp   | 2011-04-17 14:01:07.662325-04
grant_timestamp     | 2011-04-17 14:01:07.662325-04
```

**See Also****DUMP\_LOCKTABLE** (page 414)**PROJECTION\_REFRESHES** (page 863)**SELECT** (page 747) FOR UPDATE clause**SESSION\_PROFILES** (page 886)**NODE\_RESOURCES**

Provides a snapshot of the node. This is useful for regularly polling the node with automated tools or scripts.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
HOST_NAME	VARCHAR	The hostname associated with a particular node.
PROCESS_SIZE_BYTES	INTEGER	The total size of the program.
PROCESS_RESIDENT_SET_SIZE_BYTES	INTEGER	The total number of pages that the process has in memory.
PROCESS_SHARED_MEMORY_SIZE_BYTES	INTEGER	The amount of shared memory used.
PROCESS_TEXT_MEMORY_SIZE_BYTES	INTEGER	The total number of text pages that the process has in physical memory. This does not include any shared libraries.
PROCESS_DATA_MEMORY_SIZE_BYTES	INTEGER	The amount of physical memory, in pages, used for performing processes. This does not include the executable code.
PROCESS_LIBRARY_MEMORY_SIZE_BYTES	INTEGER	The total number of library pages that the process has in physical memory.
PROCESS_DIRTY_MEMORY_SIZE_BYTES	INTEGER	The number of pages that have been modified since they were last written to disk.

**Example**

Query the NODE\_RESOURCES table:

```
=>\pset expanded
```

```
Expanded display is on.
```

```
=> SELECT * FROM NODE_RESOURCES;
```

```
-[ RECORD 1 ]-----+-----
node_name           | v_vmartdb_node01
host_name           | myhost-s1
process_size_bytes  | 2001829888
process_resident_set_size_bytes | 40964096
process_shared_memory_size_bytes | 16543744
```

```

process_text_memory_size_bytes | 46649344
process_data_memory_size_bytes | 0
process_library_memory_size_bytes | 1885351936
process_dirty_memory_size_bytes | 0
-[ RECORD 2 ]-----+-----
node_name | v_vmartdb_node02
host_name | myhost-s2
process_size_bytes | 399822848
process_resident_set_size_bytes | 31453184
process_shared_memory_size_bytes | 10862592
process_text_memory_size_bytes | 46649344
process_data_memory_size_bytes | 0
process_library_memory_size_bytes | 299356160
process_dirty_memory_size_bytes | 0
-[ RECORD 3 ]-----+-----
node_name | v_vmartdb_node03
host_name | myhost-s3
process_size_bytes | 399822848
process_resident_set_size_bytes | 31100928
process_shared_memory_size_bytes | 10735616
process_text_memory_size_bytes | 46649344
process_data_memory_size_bytes | 0
process_library_memory_size_bytes | 299356160
process_dirty_memory_size_bytes | 0
-[ RECORD 4 ]-----+-----
node_name | v_vmartdb_node04
host_name | myhost-s4
process_size_bytes | 466923520
process_resident_set_size_bytes | 31309824
process_shared_memory_size_bytes | 10735616
process_text_memory_size_bytes | 46649344
process_data_memory_size_bytes | 0
process_library_memory_size_bytes | 366456832
process_dirty_memory_size_bytes | 0

```

## PARTITIONS

Displays partition metadata, one row per partition key, per ROS container.

Column Name	Data Type	Description
PARTITION_KEY	VARCHAR	The partition value(s).
TABLE_SCHEMA	VARCHAR	The schema name for which information is listed.
PROJECTION_NAME	VARCHAR	The projection name for which information is listed.
ROS_ID	VARCHAR	A unique numeric ID assigned by the Vertica catalog, which identifies the ROS container.
ROS_SIZE_BYTES	INTEGER	The ROS container size in bytes.
ROS_ROW_COUNT	INTEGER	Number of rows in the ROS container.
NODE_NAME	VARCHAR	Node where the ROS container resides.

## Notes

- A many-to-many relationship exists between partitions and ROS containers. PARTITIONS displays information in a denormalized fashion.
- To find the number of ROS containers having data of a specific partition, aggregate PARTITIONS over the `partition_key` column.
- To find the number of partitions stored in a ROS container, aggregate PARTITIONS over the `ros_id` column.

## Example

Given a projection named `p1` with three ROS containers (RC1, RC2 and RC3), the values are defined as follow:

COLUMN NAME	RC1	RC2	RC3
PARTITION_KEY	(20, 30, 40)	(20)	(30, 60)
ROS_ID	45035986273705000	45035986273705001	45035986273705002
SIZE	1000	20000	30000
ROW_ROW_COUNT	100	200	300
NODE_NAME	node01	node01	node01

The PARTITIONS function returns six rows:

```
=> SELECT PARTITION_KEY, PROJECTION_NAME, ROS_ID, ROS_SIZE_BYTES, ROS_ROW_COUNT, NODE_NAME
FROM PARTITIONS;
```

PARTITION_KEY	PROJECTION_NAME	ROS_ID	ROS_SIZE_BYTES	ROS_ROW_COUNT	NODE_NAME
20	p1	45035986273705000	10000	100	node01
30	p1	45035986273705000	10000	100	node01
40	p1	45035986273705000	10000	100	node01
20	p1	45035986273705001	20000	200	node01
30	p1	45035986273705002	30000	300	node01
60	p1	45035986273705002	30000	300	node01

## PROJECTION\_REFRESHES

Provides information about refresh operations for projections.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node where the refresh was initiated.
PROJECTION_SCHEMA	VARCHAR	The name of the schema associated with the projection.
PROJECTION_NAME	VARCHAR	The name of the projection that is targeted for refresh.
ANCHOR_TABLE_NAME	VARCHAR	The name of the projection's associated anchor table.
REFRESH_STATUS	VARCHAR	The status of the projection: <ul style="list-style-type: none"> <li>▪ Queued — Indicates that a projection is queued for refresh.</li> <li>▪ Refreshing — Indicates that a refresh for a projection is in process.</li> <li>▪ Refreshed — Indicates that a refresh for a projection has successfully completed.</li> <li>▪ Failed — Indicates that a refresh for a projection</li> </ul>

		did not successfully complete.
REFRESH_PHASE	VARCHAR	<p>Indicates how far the refresh has progressed:</p> <ul style="list-style-type: none"> <li>▪ Historical – Indicates that the refresh has reached the first phase and is refreshing data from historical data. This refresh phase requires the most amount of time.</li> <li>▪ Current – Indicates that the refresh has reached the final phase and is attempting to refresh data from the current epoch. To complete this phase, refresh must be able to obtain a lock on the table. If the table is locked by some other transaction, refresh is put on hold until that transaction completes.</li> <li>▪ The <b>LOCKS</b> (page 858) system table is useful for determining if a refresh has been blocked on a table lock. To determine if a refresh has been blocked, locate the term "refresh" in the transaction description. A refresh has been blocked when the scope for the refresh is REQUESTED and one or more other transactions have acquired a lock on the table.</li> </ul> <p><b>Note:</b> The REFRESH_PHASE field is NULL until the projection starts to refresh and is NULL after the refresh completes.</p>
REFRESH_METHOD	VARCHAR	<p>The method used to refresh the projection:</p> <ul style="list-style-type: none"> <li>▪ Buddy – Uses the contents of a buddy to refresh the projection. This method maintains historical data. This enables the projection to be used for historical queries.</li> <li>▪ Scratch – Refreshes the projection without using a buddy. This method does not generate historical data. This means that the projection cannot participate in historical queries from any point before the projection was refreshed.</li> </ul>
REFRESH_FAILURE_COUNT	INTEGER	The number of times a refresh failed for the projection. FAILURE_COUNT does not indicate whether the projection was eventually refreshed. See REFRESH_STATUS to determine how the refresh operation is progressing.
SESSION_ID	VARCHAR	A unique numeric ID assigned by the Vertica catalog, which identifies the refresh session.
REFRESH_START	TIMESTAMP	The time the projection refresh started (provided as a timestamp).
REFRESH_DURATION_SEC	INTEGER	The length of time that the projection refresh ran in seconds.

IS_EXECUTING	BOOLEAN	Indicates if the refresh is currently running (t) or if the refresh occurred in the past (f).
--------------	---------	---

### Notes

- Information about a refresh operation—whether successful or unsuccessful—is maintained in the PROJECTION\_REFRESHES system table until either the **CLEAR\_PROJECTION\_REFRESHES()** (page 395) function is executed or the storage quota for the table is exceeded.
- Tables and projections can be dropped while a query runs against them. The query continues to run, even after the drop occurs. Only when the query finishes does it notice the drop, which could cause a rollback. The same is true for refresh queries. PROJECTION\_REFRESHES, therefore, could report that a projection failed to be refreshed before the refresh query completes. In this case, the REFRESH\_DURATION\_SEC column continues to increase until the refresh query completes.

### Example

Query the PROJECTION\_REFRESHES table:

```
=>\pset expanded
Expanded display on.
=> SELECT * FROM projection_refreshes;
-[ RECORD 1 ]-----+-----
node_name          | node02
projection_schema  | public
projection_name     | fact_pl_b1
anchor_table_name  | fact
refresh_status     | refreshed
refresh_phase      |
refresh_method     | buddy
refresh_failure_count | 0
session_id         | myhost.verticacorp-15750:0x6b38
refresh_start      | 2010-09-28 15:09:12.411551
refresh_duration_sec | 2
is_executing       | f
```

The following command purges projection refresh history from the PROJECTION\_REFRESHES table:

```
=> SELECT clear_projection_refreshes();
clear_projection_refreshes
-----
CLEAR
(1 row)
```

Only the rows where the IS\_EXECUTING column equals false are cleared.

### See Also

**CLEAR\_PROJECTION\_REFRESHES** (page 395)

Clearing PROJECTION\_REFRESHES History in the Administrator's Guide

## PROJECTION\_STORAGE

Monitors the amount of disk storage used by each projection on each node.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
PROJECTION_NAME	VARCHAR	The projection name for which information is listed.
PROJECTION_SCHEMA	VARCHAR	The name of the schema associated with the projection.
PROJECTION_COLUMN_COUNT	INTEGER	The number of columns in the projection.
ROW_COUNT	INTEGER	The number of rows in the table's projections, excluding any rows marked for deletion.
USED_BYTES	INTEGER	The number of bytes of disk storage used by the projection.
WOS_ROW_COUNT	INTEGER	The number of WOS rows in the projection.
WOS_USED_BYTES	INTEGER	The number of WOS bytes in the projection.
ROS_ROW_COUNT	INTEGER	The number of ROS rows in the projection.
ROS_USED_BYTES	INTEGER	The number of ROS bytes in the projection.
ROS_COUNT	INTEGER	The number of ROS containers in the projection.
ANCHOR_TABLE_NAME	VARCHAR	The associated table name for which information is listed.
ANCHOR_TABLE_SCHEMA	VARCHAR	The associated table schema for which information is listed.

### Notes

Projections that have no data never have full statistics. Querying this system table lets you see if your projection contains data.

### Example

```
=> SELECT projection_name, row_count, ros_used_bytes, used_bytes
   FROM PROJECTION_STORAGE WHERE projection_schema = 'store' ORDER BY used_bytes;
      projection_name          | row_count | ros_used_bytes |
-----+-----+-----+-----
store_dimension_DBD_4_seg_vmartdb_design_vmartdb_design |          53 |          2791 |
store_dimension_DBD_29_seg_vmartdb_design_vmartdb_design |          53 |          2791 |
store_dimension_DBD_29_seg_vmartdb_design_vmartdb_design |          56 |          2936 |
store_dimension_DBD_4_seg_vmartdb_design_vmartdb_design |          56 |          2936 |
store_dimension_DBD_4_seg_vmartdb_design_vmartdb_design |          68 |          3360 |
store_dimension_DBD_29_seg_vmartdb_design_vmartdb_design |          68 |          3360 |
```

SQL System Tables (Monitoring APIs)

store_dimension_DBD_29_seg_vmartdb_design_vmartdb_design	73	3579
3579		
store_dimension_DBD_4_seg_vmartdb_design_vmartdb_design	73	3579
3579		
store_orders_fact_DBD_31_seg_vmartdb_design_vmartdb_design	53974	1047782
1047782		
store_orders_fact_DBD_6_seg_vmartdb_design_vmartdb_design	53974	1047782
1047782		
store_orders_fact_DBD_6_seg_vmartdb_design_vmartdb_design	66246	1285786
1285786		
store_orders_fact_DBD_31_seg_vmartdb_design_vmartdb_design	66246	1285786
1285786		
store_orders_fact_DBD_31_seg_vmartdb_design_vmartdb_design	71909	1395258
1395258		
store_orders_fact_DBD_6_seg_vmartdb_design_vmartdb_design	71909	1395258
1395258		
store_orders_fact_DBD_6_seg_vmartdb_design_vmartdb_design	107871	2090941
2090941		
store_orders_fact_DBD_31_seg_vmartdb_design_vmartdb_design	107871	2090941
2090941		
store_sales_fact_DBD_5_seg_vmartdb_design_vmartdb_design	1235825	24285740
24285740		
store_sales_fact_DBD_30_seg_vmartdb_design_vmartdb_design	1235825	24285740
24285740		
store_sales_fact_DBD_30_seg_vmartdb_design_vmartdb_design	1245865	24480819
24480819		
store_sales_fact_DBD_5_seg_vmartdb_design_vmartdb_design	1245865	24480819
24480819		
store_sales_fact_DBD_5_seg_vmartdb_design_vmartdb_design	1249547	24551817
24551817		
store_sales_fact_DBD_30_seg_vmartdb_design_vmartdb_design	1249547	24551817
24551817		
store_sales_fact_DBD_30_seg_vmartdb_design_vmartdb_design	1268763	24930549
24930549		
store_sales_fact_DBD_5_seg_vmartdb_design_vmartdb_design	1268763	24930549
24930549		

(24 rows)

The following command returns the per-node storage used by a segmented table:

```
SELECT anchor_table_name, node_name,
       SUM(ros_used_bytes)/1024/1024 AS MB,
       SUM(ros_row_count) AS Rows
FROM projection_storage GROUP BY 1,2 ORDER BY 1,2;
```

anchor_table_name	node_name	MB	Rows
call_center_dimension	v_vmartdb_node0001	0	98
call_center_dimension	v_vmartdb_node0002	0	101
call_center_dimension	v_vmartdb_node0003	0	102
call_center_dimension	v_vmartdb_node0004	0	99
customer_dimension	v_vmartdb_node0001	0	24030
customer_dimension	v_vmartdb_node0002	0	1648
customer_dimension	v_vmartdb_node0003	0	25970
customer_dimension	v_vmartdb_node0004	1	48352
date_dimension	v_vmartdb_node0001	0	910
date_dimension	v_vmartdb_node0002	0	913
date_dimension	v_vmartdb_node0003	0	916
date_dimension	v_vmartdb_node0004	0	913
employee_dimension	v_vmartdb_node0001	0	4998
employee_dimension	v_vmartdb_node0002	0	4999
employee_dimension	v_vmartdb_node0003	0	5002
employee_dimension	v_vmartdb_node0004	0	5001
inventory_fact	v_vmartdb_node0001	0	150414
inventory_fact	v_vmartdb_node0002	0	149736
inventory_fact	v_vmartdb_node0003	0	149586
inventory_fact	v_vmartdb_node0004	0	150264
online_page_dimension	v_vmartdb_node0001	0	499
online_page_dimension	v_vmartdb_node0002	0	500

```

online_page_dimension | v_vmartdb_node0003 | 0 | 501
online_page_dimension | v_vmartdb_node0004 | 0 | 500
online_sales_fact | v_vmartdb_node0001 | 59 | 4941898
online_sales_fact | v_vmartdb_node0002 | 59 | 5024898
online_sales_fact | v_vmartdb_node0003 | 59 | 5058102
online_sales_fact | v_vmartdb_node0004 | 59 | 4975102
product_dimension | v_vmartdb_node0001 | 1 | 103185
product_dimension | v_vmartdb_node0002 | 1 | 100428
product_dimension | v_vmartdb_node0003 | 1 | 76815
product_dimension | v_vmartdb_node0004 | 1 | 79572
promotion_dimension | v_vmartdb_node0001 | 0 | 499
promotion_dimension | v_vmartdb_node0002 | 0 | 500
promotion_dimension | v_vmartdb_node0003 | 0 | 501
promotion_dimension | v_vmartdb_node0004 | 0 | 500
rostab | v_vmartdb_node0001 | 0 | 0
rostab | v_vmartdb_node0002 | 0 | 0
rostab | v_vmartdb_node0003 | 0 | 0
rostab | v_vmartdb_node0004 | 0 | 0
shipping_dimension | v_vmartdb_node0001 | 0 | 49
shipping_dimension | v_vmartdb_node0002 | 0 | 50
shipping_dimension | v_vmartdb_node0003 | 0 | 51
shipping_dimension | v_vmartdb_node0004 | 0 | 50
store_dimension | v_vmartdb_node0001 | 0 | 123
store_dimension | v_vmartdb_node0002 | 0 | 125
store_dimension | v_vmartdb_node0003 | 0 | 127
store_dimension | v_vmartdb_node0004 | 0 | 125
store_orders_fact | v_vmartdb_node0001 | 2 | 147768
store_orders_fact | v_vmartdb_node0002 | 2 | 149759
store_orders_fact | v_vmartdb_node0003 | 2 | 152232
store_orders_fact | v_vmartdb_node0004 | 2 | 150241
store_sales_fact | v_vmartdb_node0001 | 46 | 2501318
store_sales_fact | v_vmartdb_node0002 | 47 | 2512691
store_sales_fact | v_vmartdb_node0003 | 46 | 2498682
store_sales_fact | v_vmartdb_node0004 | 46 | 2487309
vendor_dimension | v_vmartdb_node0001 | 0 | 48
vendor_dimension | v_vmartdb_node0002 | 0 | 48
vendor_dimension | v_vmartdb_node0003 | 0 | 52
vendor_dimension | v_vmartdb_node0004 | 0 | 52
warehouse_dimension | v_vmartdb_node0001 | 0 | 49
warehouse_dimension | v_vmartdb_node0002 | 0 | 50
warehouse_dimension | v_vmartdb_node0003 | 0 | 51
warehouse_dimension | v_vmartdb_node0004 | 0 | 50
(64 rows)

```

**See Also**

**PROJECTIONS** (page 811)

**ANALYZE\_STATISTICS** (page 385)

**QUERY\_METRICS**

Monitors the sessions and queries running on each node.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
ACTIVE_USER_SESSION_COUNT	INTEGER	The number of active user sessions (connections).
ACTIVE_SYSTEM_SESSION_COUNT	INTEGER	The number of active system sessions.
TOTAL_USER_SESSION_COUNT	INTEGER	The total number of user sessions.

TOTAL_SYSTEM_SESSION_COUNT	INTEGER	The total number of system sessions.
TOTAL_ACTIVE_SESSION_COUNT	INTEGER	The total number of active user and system sessions.
TOTAL_SESSION_COUNT	INTEGER	The total number of user and system sessions.
RUNNING_QUERY_COUNT	INTEGER	The number of queries currently running.
EXECUTED_QUERY_COUNT	INTEGER	The total number of queries that ran.

## Notes

Totals get reset each time you restart the database.

## Example

```
=>\pset expanded
```

```
Expanded display is on.
```

```
=> SELECT * FROM QUERY_METRICS;
```

```
-[ RECORD 1 ]-----+-----
node_name           | v_vmartdb_node01
active_user_session_count | 1
active_system_session_count | 2
total_user_session_count  | 2
total_system_session_count | 6248
total_active_session_count | 3
total_session_count      | 6250
running_query_count      | 1
executed_query_count     | 42
-[ RECORD 2 ]-----+-----
node_name           | v_vmartdb_node02
active_user_session_count | 1
active_system_session_count | 2
total_user_session_count  | 2
total_system_session_count | 6487
total_active_session_count | 3
total_session_count      | 6489
running_query_count      | 0
executed_query_count     | 0
-[ RECORD 3 ]-----+-----
node_name           | v_vmartdb_node03
active_user_session_count | 1
active_system_session_count | 2
total_user_session_count  | 2
total_system_session_count | 6489
total_active_session_count | 3
total_session_count      | 6491
running_query_count      | 0
executed_query_count     | 0
-[ RECORD 4 ]-----+-----
node_name           | v_vmartdb_node04
active_user_session_count | 1
active_system_session_count | 2
total_user_session_count  | 2
```

```

total_system_session_count | 6489
total_active_session_count | 3
total_session_count       | 6491
running_query_count       | 0
executed_query_count      | 0

```

## QUERY\_PROFILES

Provides information about queries that have run. To obtain information about query profiling, see [Collecting Query Information in the Troubleshooting Guide](#).

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
SESSION_ID	VARCHAR	The identification of the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
TRANSACTION_ID	INTEGER	An identifier for the transaction within the session if any; otherwise NULL.
STATEMENT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the currently-executing statement. <b>Note:</b> NULL indicates that no statement is currently being processed.
IDENTIFIER	VARCHAR	A string to identify the query in virtual tables.
QUERY	VARCHAR	The query string used for the query.
QUERY_SEARCH_PATH	VARCHAR	A list of schemas in which to look for tables.
SCHEMA_NAME	VARCHAR	The schema name in which the query is being profiled.
TABLE_NAME	VARCHAR	The table name in the query being profiled.
PROJECTIONS_USED	VARCHAR	The projections used in the query.
QUERY_DURATION_US	INTEGER	The duration of the query in microseconds.
QUERY_START_EPOCH	VARCHAR	The epoch number at the start of the given query.
QUERY_START	VARCHAR	The Linux system time of query execution in a format that can be used as a DATE/TIME expression.
QUERY_TYPE	VARCHAR	Is one of INSERT, SELECT, UPDATE, DELETE, UTILITY, or UNKNOWN.
ERROR_CODE	INTEGER	The return error code for the query.
USER_NAME	VARCHAR	The name of the user who ran the query.
PROCESSED_ROW_COUNT	INTEGER	The number of rows returned by the query.
RESERVED_EXTRA_MEMORY	INTEGER	The amount of extra memory reserved for the query. Extra memory is the amount of memory reserved for the plan but not assigned to a particular operator. This is the

		memory from which unbounded operators pull first. If they acquire all of the extra memory, then the plan must go back to the Resource Manager for more memory. See <b>Notes</b> section below this table.
IS_EXECUTING	BOOLEAN	Displays information about actively running queries, regardless of whether profiling is enabled.

## Notes

- The total memory reserved by the query is available in `RESOURCE_ACQUISITIONS.MEMORY_INUSE_KB` (page 872). The difference between `RESOURCE_ACQUISITIONS.MEMORY_INUSE_KB` and `QUERY_PROFILES.EXTRA_MEMORY` is the "essential memory."
  - `RESOURCE_ACQUISITIONS.MEMORY_INUSE_KB` is the total memory acquired.
  - `QUERY_PROFILES.EXTRA_MEMORY` is the unused portion of the acquired memory.
  - The difference gives you the memory in use.
- If the query has finished executing, query the `RESOURCE_ACQUISITIONS_HISTORY` (page 874) table.

## Example

Query the `QUERY_PROFILES` table:

```
=>\pset expanded
Expanded display is on.
=> SELECT * FROM QUERY_PROFILES;

-[ RECORD 1 ]-----+-----
...
-[ RECORD 18 ]-----+-----
node_name          | v_vmartdb_node0001
session_id         | raster-s1-17956:0x1d
transaction_id     | 45035996273728061
statement_id       | 6
identifier         |
query              | SELECT * FROM event_configurations;
query_search_path  | "$user", public, v_catalog, v_monitor, v_internal
schema_name        |
table_name         |
projections used   | v_monitor.event_configurations_p
query_duration_us  | 9647
query_start_epoch  | 429
query_start        | 2010-10-07 12:46:24.370044-04
query_type         | SELECT
error_code         | 0
user_name          | release
processed_row_count | 16
reserved_extra_memory | 0
is_executing       | f
-[ RECORD ... ]-----+-----
...
```

## See Also

***RESOURCE\_ACQUISITIONS*** (page 872)

***RESOURCE\_ACQUISITIONS\_HISTORY*** (page 874)

Profiling Database Performance, Collecting Query Information, and Monitoring the QUERY\_REPO Table in the Troubleshooting Guide

Managing Workloads in the Administrator's Guide

## RESOURCE\_ACQUISITIONS

Provides information about resources (memory, open file handles, threads) acquired by each running request for each resource pool in the system.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
TRANSACTION_ID	INTEGER	Transaction identifier for this request.
STATEMENT_ID	INTEGER	Statement identifier for this request.
POOL_NAME	VARCHAR	The name of the resource pool.
THREAD_COUNT	INTEGER	Number of threads in use by this request.
OPEN_FILE_HANDLE_COUNT	INTEGER	Number of open file handles in use by this request.
MEMORY_INUSE_KB	INTEGER	Amount of memory in Kilobytes acquired by this request. See Notes section below this table.
QUEUE_ENTRY_TIMESTAMP	TIMESTAMP	Timestamp when the request was queued.
ACQUISITION_TIMESTAMP	TIMESTAMP	Timestamp when the request was admitted to run.

### Notes

- The total memory reserved by the query is available in RESOURCE\_ACQUISITIONS.MEMORY\_INUSE\_KB. The difference between RESOURCE\_ACQUISITIONS.MEMORY\_INUSE\_KB and QUERY\_PROFILES.EXTRA\_MEMORY (page 870) is the "essential memory."
  - RESOURCE\_ACQUISITIONS.MEMORY\_INUSE\_KB is the total memory acquired.
  - QUERY\_PROFILES.EXTRA\_MEMORY is the unused portion of the acquired memory.
  - The difference gives you the memory in use.
- If the query has finished executing, query the RESOURCE\_ACQUISITIONS\_HISTORY (page 874) table.

### Example

```
vmartdb=> \x
Expanded display is on.
```

```
vmartdb=> SELECT * FROM resource_acquisitions;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node01
transaction_id     | 45035996273744830
statement_number   | 3
pool_name          | sysquery
thread_count       | 4
```

SQL System Tables (Monitoring APIs)

```

open_file_handle_count | 0
memory_inuse_kb        | 4110
queue_entry_timestamp  | 2010-05-05 16:22:43.272117-04
acquisition_timestamp  | 2010-05-05 16:22:43.272123-04
-[ RECORD 2 ]-----+-----
node_name              | v_vmartdb_node01
transaction_id         | -1
statement_number      | 45035996273708350
pool_name              | sysdata
thread_count          | 0
open_file_handle_count | 0
memory_inuse_kb        | 4096
queue_entry_timestamp  | 2010-05-05 14:22:52.863803-04
acquisition_timestamp  | 2010-05-05 14:22:52.863828-04
-[ RECORD 3 ]-----+-----
node_name              | v_vmartdb_node01
transaction_id         | -1
statement_number      | 45035996273708352
pool_name              | wosdata
thread_count          | 0
open_file_handle_count | 0
memory_inuse_kb        | 0
queue_entry_timestamp  | 2010-05-05 14:32:46.997389-04
acquisition_timestamp  | 2010-05-05 15:21:38.447699-04
-[ RECORD 4 ]-----+-----
node_name              | v_vmartdb_node02
transaction_id         | -1
statement_number      | 45035996273708352
pool_name              | wosdata
thread_count          | 0
open_file_handle_count | 0
memory_inuse_kb        | 0
queue_entry_timestamp  | 2010-05-05 14:32:47.00394-04
acquisition_timestamp  | 2010-05-05 15:21:38.448964-04
-[ RECORD 5 ]-----+-----
node_name              | v_vmartdb_node03
transaction_id         | -1
statement_number      | 45035996273708352
pool_name              | wosdata
thread_count          | 0
open_file_handle_count | 0
memory_inuse_kb        | 0
queue_entry_timestamp  | 2010-05-05 14:32:47.005306-04
acquisition_timestamp  | 2010-05-05 15:21:38.454139-04
-[ RECORD 6 ]-----+-----
node_name              | v_vmartdb_node04
transaction_id         | -1
statement_number      | 45035996273708352
pool_name              | wosdata
thread_count          | 0
open_file_handle_count | 0
memory_inuse_kb        | 0
queue_entry_timestamp  | 2010-05-05 14:32:47.003121-04
acquisition_timestamp  | 2010-05-05 15:21:38.452377-04

```

**See Also****QUERY\_PROFILES** (page 870)**RESOURCE\_ACQUISITIONS\_HISTORY** (page 874)**RESOURCE\_POOL\_STATUS** (page 877)**RESOURCE\_POOLS** (page 814)**RESOURCE\_QUEUES** (page 882)**RESOURCE\_REJECTIONS** (page 882)

Managing Workloads and Scenario: Setting a Hard Limit on Concurrency For An Application in the Administrator's Guide

**RESOURCE\_ACQUISITIONS\_HISTORY**

Provides information about resources (memory, open file handles, threads) acquired by any profiled query for each resource pool in the system. The data in this table is retained as long as the equivalent data is stored in the **QUERY\_PROFILES** (page 870) table. This means that the data may be cleared from this table when the CLEAR\_PROFILING function is called, or when data from QUERY\_PROFILES is moved to the query repository.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
TRANSACTION_ID	INTEGER	Transaction identifier for this request.
STATEMENT_ID	INTEGER	Statement identifier for this request.
POOL_NAME	VARCHAR	The name of the resource pool.
THREAD_COUNT	INTEGER	Number of threads in use by this request.
OPEN_FILE_HANDLE_COUNT	INTEGER	Number of open file handles in use by this request.
MEMORY_INUSE_KB	INTEGER	Amount of memory in kilobytes acquired by this request.
QUEUE_ENTRY_TIMESTAMP	TIMESTAMP	TIMESTAMP when the request was queued.
ACQUISITION_TIMESTAMP	TIMESTAMP	TIMESTAMP when the request was admitted to run.

**Example**

```
vmartdb=> \x
Expanded display is on.
vmartdb=> SELECT * FROM resource_acquisitions_history;
```

```
-[ RECORD 1 ]-----+-----
node_name           | v_vmartdb_node01
transaction_id      | 45035996273734167
```

SQL System Tables (Monitoring APIs)

```

statement_id          | 1
pool_name             | sysquery
thread_count         | 4
open_file_handle_count | 0
memory_inuse_kb      | 4110
queue_entry_timestamp | 2010-10-19 17:57:32.830559-04
acquisition_timestamp | 2010-10-19 17:57:32.830566-04
-[ RECORD 2 ]-----+-----
node_name             | v_vmartdb_node01
transaction_id        | 45035996273733629
statement_id          | 1
pool_name             | sysquery
thread_count         | 4
open_file_handle_count | 0
memory_inuse_kb      | 4232
queue_entry_timestamp | 2010-10-19 15:06:38.568905-04
acquisition_timestamp | 2010-10-19 15:06:38.568914-04
-[ RECORD 3 ]-----+-----
node_name             | v_vmartdb_node01
transaction_id        | 45035996273733629
statement_id          | 2
pool_name             | sysquery
thread_count         | 4
open_file_handle_count | 0
memory_inuse_kb      | 4232
queue_entry_timestamp | 2010-10-19 15:12:23.296074-04
acquisition_timestamp | 2010-10-19 15:12:23.296082-04
-[ RECORD 4 ]-----+-----
node_name             | v_vmartdb_node01
transaction_id        | 45035996273733629
statement_id          | 3
pool_name             | sysquery
thread_count         | 4
open_file_handle_count | 0
memory_inuse_kb      | 4232
queue_entry_timestamp | 2010-10-19 15:51:44.131538-04
acquisition_timestamp | 2010-10-19 15:51:44.131545-04
-[ RECORD 5 ]-----+-----
node_name             | v_vmartdb_node01
transaction_id        | 45035996273733629
statement_id          | 4
pool_name             | sysquery
thread_count         | 4
open_file_handle_count | 0
memory_inuse_kb      | 4232
queue_entry_timestamp | 2010-10-19 15:52:06.91811-04
acquisition_timestamp | 2010-10-19 15:52:06.918117-04
-[ RECORD 6 ]-----+-----
node_name             | v_vmartdb_node01
transaction_id        | 45035996273733629
statement_id          | 5
pool_name             | sysquery
thread_count         | 4
open_file_handle_count | 0
memory_inuse_kb      | 4232

```

```

queue_entry_timestamp | 2010-10-19 15:52:56.427166-04
acquisition_timestamp | 2010-10-19 15:52:56.427174-04
-[ RECORD 7 ]-----+-----
node_name             | v_vmartdb_node01
transaction_id        | 45035996273733629
statement_id          | 6
pool_name             | sysquery
thread_count          | 4
open_file_handle_count | 0
memory_inuse_kb       | 4229
queue_entry_timestamp | 2010-10-19 15:54:04.529661-04
acquisition_timestamp | 2010-10-19 15:54:04.529668-04
-[ RECORD 8 ]-----+-----
node_name             | v_vmartdb_node01
transaction_id        | 45035996273733629
statement_id          | 7
pool_name             | sysquery
thread_count          | 4
open_file_handle_count | 0
memory_inuse_kb       | 4232
queue_entry_timestamp | 2010-10-19 16:00:34.279086-04
acquisition_timestamp | 2010-10-19 16:00:34.279093-04
-[ RECORD 9 ]-----+-----
node_name             | v_vmartdb_node01
transaction_id        | 45035996273733629
statement_id          | 8
pool_name             | sysquery
thread_count          | 4
open_file_handle_count | 0
memory_inuse_kb       | 4232
queue_entry_timestamp | 2010-10-19 16:01:01.112476-04
acquisition_timestamp | 2010-10-19 16:01:01.112484-04
-[ RECORD 10 ]-----+-----
node_name             | v_vmartdb_node01
transaction_id        | 45035996273733629
statement_id          | 9
pool_name             | sysquery
thread_count          | 4
open_file_handle_count | 0
memory_inuse_kb       | 4232
queue_entry_timestamp | 2010-10-19 16:02:38.263142-04
acquisition_timestamp | 2010-10-19 16:02:38.26315-04
-[ RECORD 11 ]-----+-----
node_name             | v_vmartdb_node01
transaction_id        | 45035996273733629
statement_id          | 10
pool_name             | sysquery
thread_count          | 4
open_file_handle_count | 0
memory_inuse_kb       | 4232
queue_entry_timestamp | 2010-10-19 16:02:44.278322-04
acquisition_timestamp | 2010-10-19 16:02:44.278329-04
-[ RECORD 12 ]-----+-----
node_name             | v_vmartdb_node01

```

```

transaction_id      | 45035996273733629
statement_id       | 11
pool_name          | sysquery
thread_count       | 4
open_file_handle_count | 0
memory_inuse_kb    | 4232
queue_entry_timestamp | 2010-10-19 16:04:05.409825-04
acquisition_timestamp | 2010-10-19 16:04:05.409832-04
-[ RECORD 13 ]-----+-----
node_name          | v_vmartdb_node01
transaction_id     | 0
statement_id       | 0
pool_name          |
thread_count       | 0
open_file_handle_count | 0
memory_inuse_kb    | 0
queue_entry_timestamp | 1999-12-31 19:00:00-05
acquisition_timestamp | 1999-12-31 19:00:00-05

```

**See Also****QUERY\_PROFILES** (page 870)**RESOURCE\_ACQUISITIONS** (page 872)**RESOURCE\_POOL\_STATUS** (page 877)**RESOURCE\_POOLS** (page 814)**RESOURCE\_QUEUES** (page 882)**RESOURCE\_REJECTIONS** (page 882)

Managing Workloads in the Administrator's Guide

**RESOURCE\_POOL\_STATUS**

Provides configuration settings of the various resource pools in the system, including internal pools.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The name of the node for which information is provided.
POOL_OID	INTEGER	A unique numeric ID assigned by the Vertica catalog that identifies the pool.
POOL_NAME	VARCHAR	The name of the resource pool.
IS_INTERNAL	BOOLEAN	Denotes whether a pool is one of the <b>built-in pools</b> (page 652).
MEMORY_SIZE_KB	INTEGER	Value of <code>MemorySize</code> setting of the pool in kilobytes
MEMORY_SIZE_ACTUAL_KB	INTEGER	Current amount of memory in kilobytes allocated

		to the pool by the resource manager. Note that the actual size can be less than specified in the DDL, if the pool has been recently altered in a running system and the request to shuffle memory is pending. See <b>ALTER RESOURCE POOL</b> (page 582).
MEMORY_INUSE_KB	INTEGER	Amount of memory in Kilobytes acquired by requests running against this pool.
GENERAL_MEMORY_BORROWED_KB	INTEGER	Amount of memory in Kilobytes borrowed from the General pool by requests running against this pool. The sum of MEMORY_INUSE_KB and GENERAL_MEMORY_BORROWED_KB should be less than MAX_MEMORY_SIZE_KB (see below).
QUEUEING_THRESHOLD_KB	INTEGER	Calculated as MAX_MEMORY_SIZE_KB * 75%. When the amount of memory used by all requests against this queue exceed the QUEUEING_THRESHOLD_KB (but less than MAX_MEMORY_SIZE_KB), new requests against the pool will be queued until memory becomes available.
MAX_MEMORY_SIZE_KB	INTEGER	Value of MAXMEMORYSIZE size parameter specified when defining the pool. Provides an upper limit on the amount of memory that can be taken up by requests running against this pool. Once this threshold is reached, new requests against this pool are rejected until memory becomes available.
RUNNING_QUERY_COUNT	INTEGER	Number of queries actually running using this pool.
PLANNED_CONCURRENCY	INTEGER	Value of PLANNEDCONCURRENCY parameter specified when defining the pool.
MAX_CONCURRENCY	INTEGER	Value of MAXCONCURRENCY parameter specified when defining the pool.
IS_STANDALONE	BOOLEAN	If the pool is configured to have MEMORYSIZE equal to MAXMEMORYSIZE, it does not borrow any memory from the General pool and hence said to be standalone.
QUEUE_TIMEOUT_SECONDS	INTEGER	Value of QUEUETIMEOUT parameter specified when defining the pool.
PRIORITY	INTEGER	Value of PRIORITY parameter specified when defining the pool.
SINGLE_INITIATOR	INTEGER	Value of SINGLEINITIATOR parameter specified when defining the pool.
QUERY_BUDGET_KB	INTEGER	The current amount of memory that queries are tuned to use.

**Example**

The following command finds all the configuration settings of the various resource pools on node01:

```
vmartdb=> SELECT * FROM RESOURCE_POOL_STATUS WHERE node_name ILIKE 'node02';
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node02
pool_oid           | 45035996273708346
pool_name          | general
is_internal        | t
memory_size_kb     | 7562919
memory_size_actual_kb | 7562919
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
queueing_threshold_kb | 5672189
max_memory_size_kb | 7562919
running_query_count | 0
planned_concurrency | 4
max_concurrency    |
is_standalone      | t
queue_timeout_in_seconds | 300
priority           | 0
single_initiator   | false
query_budget_kb    | 1398847
-[ RECORD 2 ]-----+-----
node_name          | v_vmartdb_node02
pool_oid           | 45035996273708348
pool_name          | sysquery
is_internal        | t
memory_size_kb     | 65536
memory_size_actual_kb | 65536
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
queueing_threshold_kb | 5721341
max_memory_size_kb | 7628455
running_query_count | 0
planned_concurrency | 4
max_concurrency    |
is_standalone      | f
queue_timeout_in_seconds | 300
priority           | 110
single_initiator   | false
query_budget_kb    | 16384
-[ RECORD 3 ]-----+-----
node_name          | v_vmartdb_node02
pool_oid           | 45035996273708350
pool_name          | sysdata
is_internal        | t
memory_size_kb     | 102400
memory_size_actual_kb | 102400
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
queueing_threshold_kb | 587493
```

```

max_memory_size_kb          | 783325
running_query_count         | 0
planned_concurrency         | 1
max_concurrency              | 0
is_standalone                | f
queue_timeout_in_seconds    | 0
priority                     | 0
single_initiator             | false
query_budget_kb              |
-[ RECORD 4 ]-----+-----
node_name                    | v_vmartdb_node02
pool_oid                     | 45035996273708352
pool_name                     | wosdata
is_internal                   | t
memory_size_kb                | 0
memory_size_actual_kb        | 0
memory_inuse_kb               | 0
general_memory_borrowed_kb   | 0
queueing_threshold_kb        | 1468734
max_memory_size_kb           | 1958313
running_query_count          | 0
planned_concurrency          | 2
max_concurrency              | 0
is_standalone                | f
queue_timeout_in_seconds     | 0
priority                     | 0
single_initiator             | false
query_budget_kb              |
-[ RECORD 5 ]-----+-----
node_name                    | v_vmartdb_node02
pool_oid                     | 45035996273708354
pool_name                     | tm
is_internal                   | t
memory_size_kb                | 102400
memory_size_actual_kb        | 102400
memory_inuse_kb               | 0
general_memory_borrowed_kb   | 0
queueing_threshold_kb        | 5748989
max_memory_size_kb           | 7665319
running_query_count          | 0
planned_concurrency          | 1
max_concurrency              | 2
is_standalone                | f
queue_timeout_in_seconds     | 300
priority                     | 105
single_initiator             | true
query_budget_kb              | 102400
-[ RECORD 6 ]-----+-----
node_name                    | v_vmartdb_node02
pool_oid                     | 45035996273708356
pool_name                     | refresh
is_internal                   | t
memory_size_kb                | 0
memory_size_actual_kb        | 0

```

```

memory_inuse_kb          | 0
general_memory_borrowed_kb | 0
queueing_threshold_kb   | 5672189
max_memory_size_kb      | 7562919
running_query_count     | 0
planned_concurrency     | 4
max_concurrency         |
is_standalone           | f
queue_timeout_in_seconds | 300
priority                 | -10
single_initiator        | true
query_budget_kb         | 1398847
-[ RECORD 7 ]-----+-----
node_name                | v_vmartdb_node02
pool_oid                 | 45035996273708358
pool_name                | recovery
is_internal              | t
memory_size_kb           | 0
memory_size_actual_kb    | 0
memory_inuse_kb          | 0
general_memory_borrowed_kb | 0
queueing_threshold_kb   | 5672189
max_memory_size_kb      | 7562919
running_query_count     | 0
planned_concurrency     | 3
max_concurrency         | 3
is_standalone           | f
queue_timeout_in_seconds | 300
priority                 | 107
single_initiator        | true
query_budget_kb         | 932564
-[ RECORD 8 ]-----+-----
node_name                | v_vmartdb_node02
pool_oid                 | 49539595901158692
pool_name                | dbd
is_internal              | t
memory_size_kb           | 0
memory_size_actual_kb    | 0
memory_inuse_kb          | 0
general_memory_borrowed_kb | 0
queueing_threshold_kb   | 5672189
max_memory_size_kb      | 7562919
running_query_count     | 0
planned_concurrency     | 4
max_concurrency         |
is_standalone           | f
queue_timeout_in_seconds | 0
priority                 | 0
single_initiator        | true
query_budget_kb         | 1398847

```

**See Also****RESOURCE\_ACQUISITIONS** (page 872)**RESOURCE\_ACQUISITIONS\_HISTORY** (page 874)

**RESOURCE\_POOLS** (page 814)

**RESOURCE\_QUEUES** (page 882)

**RESOURCE\_REJECTIONS** (page 882)

Managing Workloads, Monitoring Resource Pools and Resource Usage by Queries, Scenario:  
Restricting Resource Usage of Ad-hoc Query Application in the Administrator's Guide

## RESOURCE\_QUEUES

Provides information about requests pending for various resource pools.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The name of the node for which information is listed.
TRANSACTION_ID	INTEGER	Transaction identifier for this request
STATEMENT_ID	INTEGER	Statement identifier for this request
POOL_NAME	VARCHAR	The name of the resource pool
MEMORY_REQUESTED_KB	INTEGER	Amount of memory in kilobytes requested by this request
PRIORITY	INTEGER	Value of PRIORITY parameter specified when defining the pool.
POSITION_IN_QUEUE	INTEGER	Position of this request within the pool's queue
QUEUE_ENTRY_TIMESTAMP	TIMESTAMP	Timestamp when the request was queued

### See Also

**RESOURCE\_ACQUISITIONS** (page 872)

**RESOURCE\_ACQUISITIONS\_HISTORY** (page 874)

**RESOURCE\_POOLS** (page 814)

**RESOURCE\_REJECTIONS** (page 882)

Managing Workloads in the Administrator's Guide

## RESOURCE\_REJECTIONS

Monitors requests for resources that are rejected by the Resource Manager.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
POOL_NAME	VARCHAR	The name of the resource pool.

REASON	VARCHAR	The reason for rejecting this request; for example: <ul style="list-style-type: none"> <li>▪ Usage of single request exceeds high limit</li> <li>▪ Timed out waiting for resource reservation</li> <li>▪ Canceled waiting for resource reservation</li> </ul>
RESOURCE_TYPE	VARCHAR	Memory, threads, file handles or execution slots. The following list shows the resources that are limited by the resource manager. A query might need some amount of each resource, and if the amount needed is not available, the query is queued and could eventually time out of the queue and be rejected. <ul style="list-style-type: none"> <li>▪ Number of running plans</li> <li>▪ Number of running plans on initiator node (local)</li> <li>▪ Number of requested threads</li> <li>▪ Number of requested file handles</li> <li>▪ Number of requested KB of memory</li> <li>▪ Number of requested KB of address space</li> </ul> <p><b>Note:</b> Execution slots are determined by MAXCONCURRENCY parameter.</p>
REJECTION_COUNT	INTEGER	Number of requests rejected due to specified reason and RESOURCE_TYPE.
FIRST_REJECTED_TIMESTAMP	TIMESTAMP	The time of the first rejection for this pool
LAST_REJECTED_TIMESTAMP	TIMESTAMP	The time of the last rejection for this pool
LAST_REJECTED_VALUE	INTEGER	The amount of the specific resource requested by the last rejection

## Notes

Information is valid only as long as the node is up and the counters reset to 0 upon node restart.

## Example

The following command returns the type of queries currently running on the node:

```
=> SELECT resource_type FROM resource_rejections;
   request_type
-----
UPDATE_QUERY
UPDATE_QUERY
UPDATE_QUERY
(3 rows)
```

## See Also

**CLEAR\_RESOURCE\_REJECTIONS** (page 396)

**DISK\_RESOURCE\_REJECTIONS** (page 843)

Managing Workloads and Managing System Resource Usage in the Administrator's Guide

Managing and Viewing Query Repository in the Administrator's Guide

**RESOURCE\_USAGE**

Monitors system resource management on each node.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
REQUEST_COUNT	INTEGER	The cumulative number of requests for threads, file handles, and memory (in kilobytes).
LOCAL_REQUEST_COUNT	INTEGER	The cumulative number of local requests.
REQUEST_QUEUE_DEPTH	INTEGER	The current request queue depth.
ACTIVE_THREAD_COUNT	INTEGER	The current number of active threads.
OPEN_FILE_HANDLE_COUNT	INTEGER	The current number of open file handles.
MEMORY_REQUESTED_KB	INTEGER	The memory requested in kilobytes.
ADDRESS_SPACE_REQUESTED_KB	INTEGER	The address space requested in kilobytes.
WOS_USED_BYTES	INTEGER	The size of the WOS in bytes.
WOS_ROW_COUNT	INTEGER	The number of rows in the WOS.
ROS_USED_BYTES	INTEGER	The size of the ROS in bytes.
ROS_ROW_COUNT	INTEGER	The number of rows in the ROS.
TOTAL_USED_BYTES	INTEGER	The total size of storage (WOS + ROS) in bytes.
TOTAL_ROW_COUNT	INTEGER	The total number of rows in storage (WOS + ROS).
RESOURCE_REQUEST_REJECT_COUNT	INTEGER	The number of rejected plan requests.
RESOURCE_REQUEST_TIMEOUT_COUNT	INTEGER	The number of resource request timeouts.
RESOURCE_REQUEST_CANCEL_COUNT	INTEGER	The number of resource request cancelations.
DISK_SPACE_REQUEST_REJECT_COUNT	INTEGER	The number of rejected disk write requests.
FAILED_VOLUME_REJECT_COUNT	INTEGER	The number of rejections due to a failed volume.
TOKENS_USED	INTEGER	For internal use only.

TOKENS_AVAILABLE	INTEGER	For internal use only.
------------------	---------	------------------------

**Example**

```
=>\pset expanded
```

```
Expanded display is on.
```

```
=> SELECT * FROM RESOURCE_USAGE;
```

```

-[ RECORD 1 ]-----+-----
node_name          | node01
request_count      | 1
local_request_count | 1
request_queue_depth | 0
active_thread_count | 4
open_file_handle_count | 2
memory_requested_kb | 4352
address_space_requested_kb | 106752
wos_used_bytes     | 0
wos_row_count      | 0
ros_used_bytes     | 10390319
ros_row_count      | 324699
total_used_bytes   | 10390319
total_row_count    | 324699
resource_request_reject_count | 0
resource_request_timeout_count | 0
resource_request_cancel_count | 0
disk_space_request_reject_count | 0
failed_volume_reject_count | 0
tokens_used        | 1
tokens_available   | 7999999
-[ RECORD 2 ]-----+-----
node_name          | node02
request_count      | 0
local_request_count | 0
request_queue_depth | 0
active_thread_count | 0
open_file_handle_count | 0
memory_requested_kb | 0
address_space_requested_kb | 0
wos_used_bytes     | 0
wos_row_count      | 0
ros_used_bytes     | 10359489
ros_row_count      | 324182
total_used_bytes   | 10359489
total_row_count    | 324182
resource_request_reject_count | 0
resource_request_timeout_count | 0
resource_request_cancel_count | 0
disk_space_request_reject_count | 0
failed_volume_reject_count | 0
tokens_used        | 0
tokens_available   | 8000000
-[ RECORD 3 ]-----+-----
node_name          | node03

```

```

request_count          | 0
local_request_count   | 0
request_queue_depth   | 0
active_thread_count   | 0
open_file_handle_count | 0
memory_requested_kb   | 0
address_space_requested_kb | 0
wos_used_bytes        | 0
wos_row_count         | 0
ros_used_bytes        | 10355231
ros_row_count         | 324353
total_used_bytes      | 10355231
total_row_count       | 324353
resource_request_reject_count | 0
resource_request_timeout_count | 0
resource_request_cancel_count | 0
disk_space_request_reject_count | 0
failed_volume_reject_count | 0
tokens_used           | 0
tokens_available      | 8000000
-[ RECORD 4 ]-----+-----
node_name             | node04
request_count        | 0
local_request_count   | 0
request_queue_depth   | 0
active_thread_count   | 0
open_file_handle_count | 0
memory_requested_kb   | 0
address_space_requested_kb | 0
wos_used_bytes        | 0
wos_row_count         | 0
ros_used_bytes        | 10385744
ros_row_count         | 324870
total_used_bytes      | 10385744
total_row_count       | 324870
resource_request_reject_count | 0
resource_request_timeout_count | 0
resource_request_cancel_count | 0
disk_space_request_reject_count | 0
failed_volume_reject_count | 0
tokens_used           | 0
tokens_available      | 8000000

```

## SESSION\_PROFILES

Provides basic session parameters and lock time out data. To obtain information about sessions, see Profiling Database Performance.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.

USER_NAME	VARCHAR	The name used to log in to the database or NULL if the session is internal.
CLIENT_HOSTNAME	VARCHAR	The host name and port of the TCP socket from which the client connection was made; NULL if the session is internal.
LOGIN_TIMESTAMP	TIMESTAMP	The date and time the user logged into the database or when the internal session was created. This field is useful for identifying sessions that have been left open for a period of time and could be idle.
LOGOUT_TIMESTAMP	TIMESTAMP	The date and time the user logged out of the database or when the internal session was closed.
SESSION_ID	VARCHAR	A unique numeric ID assigned by the Vertica catalog, which identifies the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
EXECUTED_STATEMENT_SUCCESS_COUNT	INTEGER	The number of successfully run statements.
EXECUTED_STATEMENT_FAILURE_COUNT	INTEGER	The number of unsuccessfully run statements.
LOCK_GRANT_COUNT	INTEGER	The number of locks granted during the session.
DEADLOCK_COUNT	INTEGER	The number of deadlocks encountered during the session.
LOCK_TIMEOUT_COUNT	INTEGER	The number of times a lock timed out during the session.
LOCK_CANCELLATION_COUNT	INTEGER	The number of times a lock was canceled during the session.
LOCK_REJECTION_COUNT	INTEGER	The number of times a lock was rejected during a session.
LOCK_ERROR_COUNT	INTEGER	The number of lock errors encountered during the session.

### Example

Query the SESSION\_PROFILES table:

```
=> \pset expanded
Expanded display on.
=> SELECT * FROM SESSION_PROFILES;
-[ RECORD 1 ]-----+-----
node_name          | node04
user_name          | dbadmin
client_hostname    | 192.168.1.1:46816
login_timestamp    | 2009-09-28 11:40:34.01518
```

```

logout_timestamp          | 2009-09-28 11:41:01.811484
session_id                | myhost.verticacorp-20790:0x32f
executed_statement_success_count | 51
executed_statement_failure_count | 1
lock_grant_count          | 579
deadlock_count            | 0
lock_timeout_count        | 0
lock_cancellation_count   | 0
lock_rejection_count      | 0
lock_error_count          | 0
    
```

**See Also**

**LOCKS** (page 858)

**SESSIONS**

Monitors external sessions. You can use this table to:

- Identify users who are running long queries
- Identify users who are holding locks due to an idle but uncommitted transaction
- Disconnect users in order to shut down the database
- Determine the details behind the type of database security (Secure Socket Layer (SSL) or client authentication) used for a particular session.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
USER_NAME	VARCHAR	The name used to log into the database or NULL if the session is internal.
CLIENT_HOSTNAME	VARCHAR	The host name and port of the TCP socket from which the client connection was made; NULL if the session is internal.
CLIENT_PID	INTEGER	The process identifier of the client process that issued this connection. Remember that the client process could be on a different machine than the server.
LOGIN_TIMESTAMP	TIMESTAMP	The date and time the user logged into the database or when the internal session was created. This can be useful for identifying sessions that have been left open for a period of time and could be idle.
SESSION_ID	VARCHAR	The identifier required to close or interrupt a session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
CLIENT_LABEL	VARCHAR	A user-specified label for the client connection that can be set when using ODBC. See <code>SessionLabel</code> in DSN Parameters in

		Programmer's Guide.
TRANSACTION_START	DATE	The date/time the current transaction started or NULL if no transaction is running.
TRANSACTION_ID	INTEGER	A string containing the hexadecimal representation of the transaction ID, if any; otherwise NULL.
TRANSACTION_DESCRIPTION	VARCHAR	A description of the current transaction.
STATEMENT_START	TIMESTAMP	The timestamp the current statement started execution, or NULL if no statement is running.
STATEMENT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the currently-executing statement. <b>Note:</b> NULL indicates that no statement is currently being processed.
LAST_STATEMENT_DURATION_US	INTEGER	The duration of the last completed statement in microseconds.
CURRENT_STATEMENT	VARCHAR	The currently executing statement, if any. NULL indicates that no statement is currently being processed.
LAST_STATEMENT	VARCHAR	NULL if the user has just logged in; otherwise the currently running statement or the most recently completed statement.
SSL_STATE	VARCHAR	Indicates if Vertica used Secure Socket Layer (SSL) for a particular session. Possible values are: <ul style="list-style-type: none"> <li>▪ None – Vertica did not use SSL.</li> <li>▪ Server – Server authentication was used, so the client could authenticate the server.</li> <li>▪ Mutual – Both the server and the client authenticated one another through mutual authentication.</li> </ul> See Implementing Security and Implementing SSL.
AUTHENTICATION_METHOD	VARCHAR	The type of client authentication used for a particular session, if known. Possible values are: <ul style="list-style-type: none"> <li>▪ Unknown</li> <li>▪ Trust</li> <li>▪ Reject</li> <li>▪ Kerberos</li> <li>▪ Password</li> <li>▪ MD5</li> <li>▪ LDAP</li> <li>▪ Kerberos-GSS</li> </ul> See Implementing Security and Implementing

	Client Authentication.
--	------------------------

## Notes

- The superuser has unrestricted access to all session information, but users can only view information about their own, current sessions.
- During session initialization and termination, you might see sessions running only on nodes other than the node on which you ran the virtual table query. This is a temporary situation that corrects itself as soon as session initialization and termination completes.

## Example

```
=>\pset expanded
```

```
Expanded display is on.
```

```
=> SELECT * FROM SESSIONS;
```

```

-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node01
user_name          | dbadmin
client_hostname    | xxx.x.x.x:xxxxx
client_pid         | 18082
login_timestamp    | 2010-10-07 10:10:03.114863-04
session_id         | myhost-17956:0x1d
client_label       |
transaction_start  | 2010-10-07 12:40:15.243772
transaction_id     | 45035996273728061
transaction_description | user dbadmin(SELECT * FROM database_snapshots;)
statement_start    | 2010-10-07 13:33:56.542804
statement_id       | 34
last_statement_duration_us | 19194
current_statement  | SELECT * FROM SESSIONS;
last_statement     | select get_table_projections('transaction');
ssl_state          | None
authentication_method | Trust

```

## See Also

**CLOSE\_SESSION** (page 396) and **CLOSE\_ALL\_SESSIONS** (page 399)

Managing Sessions and Configuration Parameters in the Administrator's Guide

## STORAGE\_CONTAINERS

Monitors information about WOS and ROS storage containers in the database.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
SCHEMA_NAME	VARCHAR	The schema name for which information is listed.
PROJECTION_NAME	VARCHAR	The projection name for which information is listed on that node.

STORAGE_TYPE	VARCHAR	Type of storage container: ROS or WOS.
STORAGE_OID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the storage.
TOTAL_ROW_COUNT	VARCHAR	Total rows in the storage container listed for that projection.
DELETED_ROW_COUNT	INTEGER	Total rows in the storage container deleted for that projection.
USED_BYTES	INTEGER	Total bytes in the storage container listed for that projection.
START_EPOCH	INTEGER	The number of the start epoch in the storage container for which information is listed.
END_EPOCH	INTEGER	The number of the end epoch in the storage container for which information is listed.
GROUPING	VARCHAR	The group by which columns are stored: <ul style="list-style-type: none"> <li>▪ ALL – All columns are grouped</li> <li>▪ PROJECTION – Columns grouped according to projection definition</li> <li>▪ NONE – No columns grouped, despite grouping in the projection definition</li> <li>▪ OTHER – Some grouping but neither all nor according to projection (e.g., results from add column)</li> </ul>

### Example

The following command returns all the nodes on which a segmented projection has data on the TickStore database:

```
TickStore=> SELECT node_name, projection_name, total_row_count
FROM storage_containers ORDER BY projection_name;
```

```

node_name | projection_name | total_row_count
-----|-----|-----
v_tick_node0001 | Quotes_Fact_tmp_node0001 | 512
v_tick_node0001 | Quotes_Fact_tmp_node0001 | 480176
v_tick_node0002 | Quotes_Fact_tmp_node0002 | 512
v_tick_node0002 | Quotes_Fact_tmp_node0002 | 480176
v_tick_node0003 | Quotes_Fact_tmp_node0003 | 480176
v_tick_node0003 | Quotes_Fact_tmp_node0003 | 512
v_tick_node0004 | Quotes_Fact_tmp_node0004 | 480176
v_tick_node0004 | Quotes_Fact_tmp_node0004 | 512
v_tick_node0001 | Trades_Fact_tmp_node0001 | 512
v_tick_node0001 | Trades_Fact_tmp_node0001 | 500334
v_tick_node0002 | Trades_Fact_tmp_node0002 | 500334
v_tick_node0002 | Trades_Fact_tmp_node0002 | 512
v_tick_node0003 | Trades_Fact_tmp_node0003 | 500334
v_tick_node0003 | Trades_Fact_tmp_node0003 | 512
v_tick_node0004 | Trades_Fact_tmp_node0004 | 500334
v_tick_node0004 | Trades_Fact_tmp_node0004 | 512
(16 rows)
```

The following command returns information on inventory\_fact projections on all nodes on the Vmart schema:

```
=> SELECT * FROM storage_containers WHERE projection_name LIKE
'inventory_fact_p%';
```

```

-[ RECORD 1 ]-----+-----
node_name      | node01
schema_name    | public
projection_name | inventory_fact_p_node0001
storage_type   | WOS
storage_oid    | 45035996273720173
total_row_count | 3000
deleted_row_count | 100
used_bytes     | 196608
start_epoch    | 1
end_epoch      | 2
grouping       | ALL
-[ RECORD 2 ]-----+-----
node_name      | node01
schema_name    | public
projection_name | inventory_fact_p_node0001
storage_type   | ROS
storage_oid    | 45035996273722211
total_row_count | 500
deleted_row_count | 25
used_bytes     | 5838
start_epoch    | 1
end_epoch      | 1
grouping       | ALL
-[ RECORD 3 ]-----+-----
node_name      | node01
schema_name    | public
projection_name | inventory_fact_p_node0001
storage_type   | ROS
storage_oid    | 45035996273722283
total_row_count | 500
deleted_row_count | 25
used_bytes     | 5794
start_epoch    | 1
end_epoch      | 1
grouping       | ALL
-[ RECORD 4 ]-----+-----
node_name      | node01
schema_name    | public
projection_name | inventory_fact_p_node0001
storage_type   | ROS
storage_oid    | 45035996273723379
total_row_count | 500
deleted_row_count | 25
used_bytes     | 5838
start_epoch    | 1
end_epoch      | 1
grouping       | ALL
-[ RECORD 5 ]-----+-----
node_name      | node01
schema_name    | public
projection_name | inventory_fact_p_node0001
storage_type   | ROS
storage_oid    | 45035996273723451

```

```

total_row_count | 500
deleted_row_count | 25
used_bytes | 5794
start_epoch | 1
end_epoch | 1
grouping | ALL
-[ RECORD 6 ]-----+-----
node_name | node01
schema_name | public
projection_name | inventory_fact_p_node0001
storage_type | ROS
storage_oid | 45035996273724547
total_row_count | 500
deleted_row_count | 0
used_bytes | 5838
start_epoch | 2
end_epoch | 2
grouping | ALL
-[ RECORD 7 ]-----+-----
node_name | node01
schema_name | public
projection_name | inventory_fact_p_node0001
storage_type | ROS
storage_oid | 45035996273724619
total_row_count | 500
deleted_row_count | 0
used_bytes | 5794
start_epoch | 2
end_epoch | 2
grouping | ALL
-[ RECORD 8 ]-----+-----
...

```

**See Also**

Column Store Architecture with FlexStore in the Concepts Guide

**STRATA**

This table contains internal details of how the Tuple Mover combines ROSs in each projection, broken down by stratum. For a brief overview of how the Tuple Mover combines ROSs, see Tuple Mover topic in Administrator's Guide. The STRATA table contains detailed information on how the ROS containers are classified by size and partition. The related **STRATA\_STRUCTURES** (page 896) virtual table provides a summary of the strata values.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed
SCHEMA_NAME	VARCHAR	The schema name for which information is listed
PROJECTION_NAME	VARCHAR	The projection name for which information is listed on that node

PARTITION_KEY	VARCHAR	The data partition for which information is listed
STRATA_COUNT	INTEGER	The total number of strata for this projection partition
STRATUM_CAPACITY	INTEGER	The maximum number of ROS containers for the stratum before they must be merged.
STRATUM_HEIGHT	FLOAT	The size ratio between the smallest and largest ROS container in this stratum
STRATUM_NO	INTEGER	The stratum number. Strata are numbered starting at 0, for the stratum containing the smallest ROS containers
STRATUM_LOWER_SIZE	INTEGER	The smallest ROS container size allowed in this stratum
STRATUM_UPPER_SIZE	INTEGER	The largest ROS container size allowed in this stratum
ROS_CONTAINER_COUNT	INTEGER	The current number of ROS containers in the projection partition

**Example**

```

vmartdb=> \pset expanded
Expanded display is on.
vmartdb=> SELECT * from STRATA WHERE node_name ILIKE 'node01'
        AND stratum_upper_size < '15MB';
-[ RECORD 1
]-----+-----
node_name          | v_vmartdb_node01
schema_name        | online_sales
projection_name     |
call_center_dimension_DBD_32_seg_vmart_design_vmart_design
partition_key      |
strata_count        | 5
stratum_capacity    | 19
stratum_height      | 8.97589786696783
stratum_no          | 0
stratum_lower_size  | 0B
stratum_upper_size  | 13MB
ROS_container_count | 1
-[ RECORD 2
]-----+-----
node_name          | v_vmartdb_node01
schema_name        | online_sales
projection_name     |
call_center_dimension_DBD_8_seg_vmart_design_vmart_design
partition_key      |
strata_count        | 5
stratum_capacity    | 19
stratum_height      | 8.97589786696783
stratum_no          | 0
stratum_lower_size  | 0B
stratum_upper_size  | 13MB
ROS_container_count | 1
-[ RECORD 3
]-----+-----
node_name          | v_vmartdb_node01

```

SQL System Tables (Monitoring APIs)

```

schema_name          | online_sales
projection_name      | online_sales_fact_DBD_33_seg_vmart_design_vmart_design
partition_key       |
strata_count        | 5
stratum_capacity    | 13
stratum_height      | 8.16338338718601
stratum_no          | 1
stratum_lower_size  | 19MB
stratum_upper_size  | 155.104MB
ROS_container_count | 1
-[ RECORD 4
]-----+-----
node_name           | v_vmartdb_node01
schema_name         | online_sales
projection_name      | online_sales_fact_DBD_9_seg_vmart_design_vmart_design
partition_key       |
strata_count        | 5
stratum_capacity    | 13
stratum_height      | 8.16338338718601
stratum_no          | 1
stratum_lower_size  | 19MB
stratum_upper_size  | 155.104MB
ROS_container_count | 1
-[ RECORD 5
]-----+-----
node_name           | v_vmartdb_node01
schema_name         | public
projection_name      | promotion_dimension_DBD_16_seg_vmart_design_vmart_design
partition_key       |
strata_count        | 5
stratum_capacity    | 19
stratum_height      | 8.97589786696783
stratum_no          | 0
stratum_lower_size  | 0B
stratum_upper_size  | 13MB
ROS_container_count | 1
-[ RECORD 6
]-----+-----
node_name           | v_vmartdb_node01
schema_name         | public
projection_name      | promotion_dimension_DBD_17_seg_vmart_design_vmart_design
partition_key       |
strata_count        | 5
stratum_capacity    | 19
stratum_height      | 8.97589786696783
stratum_no          | 0
stratum_lower_size  | 0B
stratum_upper_size  | 13MB
ROS_container_count | 1
-[ RECORD 7
]-----+-----
node_name           | v_vmartdb_node01
schema_name         | store
projection_name      | store_sales_fact_DBD_29_seg_vmart_design_vmart_design
partition_key       |

```

```

strata_count          | 5
stratum_capacity     | 16
stratum_height       | 8.52187248329035
stratum_no           | 1
stratum_lower_size   | 16MB
stratum_upper_size   | 136.35MB
ROS_container_count  | 1
-[ RECORD 8
]-----+-----
node_name            | v_vmartdb_node01
schema_name          | store
projection_name      | store_sales_fact_DBD_5_seg_vmart_design_vmart_design
partition_key        |
strata_count         | 5
stratum_capacity     | 16
stratum_height       | 8.52187248329035
stratum_no           | 1
stratum_lower_size   | 16MB
stratum_upper_size   | 136.35MB
ROS_container_count  | 1

```

## STRATA STRUCTURES

This table provides an overview of the Tuple Mover's internal details. It summarizes how the ROS containers are classified by size. A more detailed view can be found in the **STRATA** (page 893) virtual table.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed
SCHEMA_NAME	VARCHAR	The schema name for which information is listed
PROJECTION_NAME	VARCHAR	The projection name for which information is listed on that node
PARTITION_KEY	VARCHAR	The data partition for which the information is listed
STRATA_COUNT	INTEGER	The total number of strata for this projection partition
STRATUM_CAPACITY	INTEGER	The maximum number of ROS containers that the strata can contained before it must merge them
STRATUM_HEIGHT	FLOAT	The size ratio between the smallest and largest ROS container in a stratum.
ACTIVE_STRATA_COUNT	INTEGER	The total number of strata that have ROS containers in them

### Example

```

vmartdb=> \pset expanded
Expanded display is on.
vmartdb=> SELECT * FROM strata_structures WHERE stratum_capacity > 60;
-[ RECORD 1 ]-----+-----
node_name            | v_vmartdb_node01
schema_name          | public

```

SQL System Tables (Monitoring APIs)

```

projection_name | shipping_dimension_DBD_22_seg_vmart_design_vmart_design
partition_key   |
strata_count    | 4
stratum_capacity | 62
stratum_height  | 25.6511590887058
active_strata_count | 1
-[ RECORD 2 ]-----+-----
node_name       | v_vmartdb_node01
schema_name     | public
projection_name | shipping_dimension_DBD_23_seg_vmart_design_vmart_design
partition_key   |
strata_count    | 4
stratum_capacity | 62
stratum_height  | 25.6511590887058
active_strata_count | 1
-[ RECORD 3 ]-----+-----
node_name       | v_vmartdb_node02
schema_name     | public
projection_name | shipping_dimension_DBD_22_seg_vmart_design_vmart_design
partition_key   |
strata_count    | 4
stratum_capacity | 62
stratum_height  | 25.6511590887058
active_strata_count | 1
-[ RECORD 4 ]-----+-----
node_name       | v_vmartdb_node02
schema_name     | public
projection_name | shipping_dimension_DBD_23_seg_vmart_design_vmart_design
partition_key   |
strata_count    | 4
stratum_capacity | 62
stratum_height  | 25.6511590887058
active_strata_count | 1
-[ RECORD 5 ]-----+-----
node_name       | v_vmartdb_node03
schema_name     | public
projection_name | shipping_dimension_DBD_22_seg_vmart_design_vmart_design
partition_key   |
strata_count    | 4
stratum_capacity | 62
stratum_height  | 25.6511590887058
active_strata_count | 1
-[ RECORD 6 ]-----+-----
node_name       | v_vmartdb_node03
schema_name     | public
projection_name | shipping_dimension_DBD_23_seg_vmart_design_vmart_design
partition_key   |
strata_count    | 4
stratum_capacity | 62
stratum_height  | 25.6511590887058
active_strata_count | 1
-[ RECORD 7 ]-----+-----
node_name       | v_vmartdb_node04
schema_name     | public
projection_name | shipping_dimension_DBD_22_seg_vmart_design_vmart_design

```

```

partition_key      |
strata_count       | 4
stratum_capacity   | 62
stratum_height     | 25.6511590887058
active_strata_count | 1
-[ RECORD 8 ]-----+-----
node_name          | v_vmartdb_node04
schema_name        | public
projection_name    | shipping_dimension_DBD_23_seg_vmart_design_vmart_design
partition_key      |
strata_count       | 4
stratum_capacity   | 62
stratum_height     | 25.6511590887058
active_strata_count | 1

```

## SYSTEM

Monitors the overall state of the database.

Column Name	Data Type	Description
CURRENT_EPOCH	INTEGER	The current epoch number.
AHM_EPOCH	INTEGER	The AHM epoch number.
LAST_GOOD_EPOCH	INTEGER	The smallest (min) of all the checkpoint epochs on the cluster.
REFRESH_EPOCH	INTEGER	The oldest of the refresh epochs of all the nodes in the cluster
DESIGNED_FAULT_TOLERANCE	INTEGER	The designed or intended K-safety level.
NODE_COUNT	INTEGER	The number of nodes in the cluster.
NODE_DOWN_COUNT	INTEGER	The number of nodes in the cluster that are currently down.
CURRENT_FAULT_TOLERANCE	INTEGER	The number of node failures the cluster can tolerate before it shuts down automatically.
CATALOG_REVISION_NUMBER	INTEGER	The catalog version number.
WOS_USED_BYTES	INTEGER	The WOS size in bytes (cluster-wide).
WOS_ROW_COUNT	INTEGER	The number of rows in WOS (cluster-wide).
ROS_USED_BYTES	INTEGER	The ROS size in bytes (cluster-wide).
ROS_ROW_COUNT	INTEGER	The number of rows in ROS (cluster-wide).
TOTAL_USED_BYTES	INTEGER	The total storage in bytes (WOS + ROS) (cluster-wide).
TOTAL_ROW_COUNT	INTEGER	The total number of rows (WOS + ROS) (cluster-wide).

### Example

Query the SYSTEM table:

```

=>\pset expanded
Expanded display is on.
=> SELECT * FROM SYSTEM;
-[ RECORD 1 ]-----+-----
current_epoch          | 429
ahm_epoch              | 428
last_good_epoch        | 428
refresh_epoch          | -1
designed_fault_tolerance | 1
node_count             | 4
node_down_count        | 0
current_fault_tolerance | 1
catalog_revision_number | 1590
wos_used_bytes         | 0
wos_row_count          | 0
ros_used_bytes         | 443131537
ros_row_count          | 21809072
total_used_bytes       | 443131537
total_row_count        | 21809072

```

If there are no projections in the system, `LAST_GOOD_EPOCH` returns the following:

```

=> SELECT get_last_good_epoch();
ERROR:  Last good epoch not set

```

And if there are projections in the system:

```

=> SELECT get_last_good_epoch();
   get_last_good_epoch
-----
                        428
(1 row)

```

## TUNING\_RECOMMENDATIONS

Returns the tuning recommendation results from the last `ANALYZE_WORKLOAD()` call. This is useful for letting you build filters on the Workload Analyzer result set.

See the Notes section below this table for example output for the parameters.

Column	Data type	Description
<code>observation_count</code>	INTEGER	An integer for the total number of events observed for this tuning recommendation.
<code>first_observation_time</code>	TIMESTAMP TZ	The timestamp for when the event first occurred. If null, the tuning recommendation is from the current status of the system instead of from any event.
<code>last_observation_time</code>	TIMESTAMP TZ	The timestamp for when the event last occurred. If null, the tuning recommendation is from the current status of the system instead of from any event.
<code>tuning_parameter</code>	VARCHAR	The objects on which tuning action will be

		applied.
<i>tuning_description</i>	VARCHAR	Returns the tuning recommendation from the Workload Analyzer.
<i>tuning_command</i>	VARCHAR	Returns a command string if tuning action is a SQL command.
<i>tuning_cost</i>	VARCHAR	The cost of performing the recommended tuning operation. Is one of: <ul style="list-style-type: none"> <li>▪ LOW</li> <li>▪ MEDIUM</li> <li>▪ HIGH.</li> </ul>

## Notes

- An example return value for `tuning_parameter` is `schema.table.column`; for example:  
`store.store_orders_fact.date_ordered`  
The above identifies the `date_ordered` column as the column that needs tuning. This is useful because it means the DBA can run `ANALYZE_STATISTICS` on a single column, rather than on an entire table.
- Example values returned for the `tuning_description` column are:
  - Run database designer on table `public.t`
  - Create replicated projection for table `public.vendor_dimension`
  - Consider query-specific design on query
  - Reset configuration parameter with `set_config_parameter()`
  - Re-segment projection `public.p_t` on high-cardinality column(s)
- Example return values for the `tuning_command` column are listed below. The first statement tells the user to update statistics on a particular `schema.table.column`:  
`SELECT ANALYZE_STATISTICS('public.tablename.column-name');`  
The next statement indicates a mismatched configuration parameter 'LockTimeout':  
`SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name = 'LockTimeout';`

## Examples

The following query returns the recommendation output from the last `ANALYZE_WORKLOAD()` call on table `t1`:

```
=> \x
Expanded display is on.

=> SELECT * FROM tuning_recommendations;

-[ RECORD 1 ]-----+-----
observation_count    | 1
first_observation_time |
last_observation_time |
tuning_parameter     | public.t1
```

```
tuning_description      | run database designer on table public.t1
tuning_command         |
tuning_cost            | HIGH
```

In the output above, an issue was found on table `t1`. The tuning recommendation is to run the Database Designer on table `t1`.

**Note:** Null observation times in the results means that the tuning recommendation is from the current status of the system instead of from any event.

You can also obtain recommendation output by invoking the `ANALYZE_WORKLOAD()` function. For example, the following statement instructs the Workload Analyzer to return all events for table `t1`:

```
=> \x
Expanded display is on.
=> SELECT ANALYZE_WORKLOAD('t1');

-[ RECORD 1 ]-----+-----
observation_count   | 1
first_observation_time |
last_observation_time |
tuning_parameter    | public.t1
tuning_description  | run database designer on table public.t1
tuning_command      |
tuning_cost         | HIGH
```

## See Also

***ANALYZE\_WORKLOAD*** (page 386)

Analyzing Workloads in the Administrator's Guide

## TUPLE\_MOVER\_OPERATIONS

Monitors the status of the Tuple Mover (ATM) on each node.

Column Name	Data Type	Description
<code>OPERATION_START_TIMESTAMP</code>	<code>TIMESTAMP</code>	The start time of a Tuple Mover operation.
<code>NODE_NAME</code>	<code>VARCHAR</code>	The node name for which information is listed.
<code>OPERATION_NAME</code>	<code>VARCHAR</code>	One of the following operations: Moveout Mergeout Analyze Statistics
<code>OPERATION_STATUS</code>	<code>VARCHAR</code>	Returns <code>Running</code> or an empty string to indicate 'not running.'
<code>TABLE_SCHEMA</code>	<code>VARCHAR</code>	The schema name for the specified projection.
<code>TABLE_NAME</code>	<code>VARCHAR</code>	The table name for the specified projection

PROJECTION_NAME	VARCHAR	The name of the projection being processed.
EARLIEST_CONTAINER_START_EPOCH	INTEGER	Populated for mergeout, purge and merge_partitions operations only. For an ATM-invoked mergeout, for example, the returned value represents the lowest epoch of containers involved in the mergeout.
LATEST_CONTAINER_END_EPOCH	INTEGER	Populated for mergeout, purge and merge_partitions operations only. For an ATM-invoked mergeout, for example, the returned value represents the highest epoch of containers involved in the mergeout.
ROS_COUNT	INTEGER	The number of ROS containers.
TOTAL_ROS_USED_BYTES	INTEGER	The size in bytes of all ROS containers in the mergeout operation. (Not applicable for other operations.)
PLAN_TYPE	VARCHAR	One of the following values: Moveout Mergeout Analyze Replay Delete

## Notes

Manual mergeouts are invoked using one of the following APIs:

- **DO\_TM\_TASK** (page 406)()
- **PURGE** (page 444)
- **MERGE\_PARTITIONS** (page 439)

**Note:** No output from `TUPLE_MOVER_OPERATIONS` means that the Tuple Mover is not performing an operation.

## Example

```
=> SELECT node_name, operation_status, projection_name, plan_type
      FROM TUPLE_MOVER_OPERATIONS;
node_name | operation_status | projection_name | plan_type
-----+-----+-----+-----
node0001  | Running         | p1_b2          | Mergeout
node0002  | Running         | p1             | Mergeout
node0001  | Running         | p1_b2          | Replay Delete
node0001  | Running         | p1_b2          | Mergeout
node0002  | Running         | p1_b2          | Mergeout
node0001  | Running         | p1_b2          | Replay Delete
node0002  | Running         | p1             | Mergeout
node0003  | Running         | p1_b2          | Replay Delete
node0001  | Running         | p1             | Mergeout
node0002  | Running         | p1_b1          | Mergeout
```

## See Also

**DO\_TM\_TASK** (page 406), **MERGE\_PARTITIONS** (page 439), and **PURGE** (page 444)

Understanding the Tuple Mover and Partitioning Tables in the Administrator's Guide

## USER\_LIBRARIES

Lists the user libraries that are currently loaded.

Column Name	Data Type	Description
SCHEMA_NAME	VARCHAR	The name of the schema containing the library
LIB_NAME	VARCHAR	The name of the library
LIB_OID	INTEGER	The object ID of the library
OWNER_ID	INTEGER	The object ID of the library's owner
LIB_FILE_NAME	VARCHAR	The name of the shared library file
MD5_SUM	VARCHAR	The MD5 checksum of the library file, used to ensure that the file was correctly copied to each node
SDK_VERSION	VARCHAR	The version of the Vertica SDK used to compile the library.
REVISION	VARCHAR	The revision of the Vertica SDK used to compile the library.

## USER\_LIBRARY\_MANIFEST

Lists the User Defined Functions contained in all of the loaded user libraries.

Column Name	Data Type	Description
SCHEMA_NAME	VARCHAR	The name of the schema containing the function.
LIB_NAME	VARCHAR	The name of the library containing the UDF.
LIB_OID	INTEGER	The object ID of the library containing the function.
OBJ_NAME	VARCHAR	The name of the constructor class in the library for a function.
OBJ_TYPE	VARCHAR	The type of user defined function (scalar function, transform function)
ARG_TYPES	VARCHAR	A comma-delimited list of data types of the function's parameters.
RETURN_TYPE	VARCHAR	A comma-delimited list of data types of the function's return values.

## WOS\_CONTAINER\_STORAGE

Monitors information about WOS storage, which is divided into regions. Each region allocates blocks of a specific size to store rows.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
WOS_TYPE	VARCHAR	Returns one of the following: <ul style="list-style-type: none"> <li>▪ <code>system</code> – for system table queries</li> <li>▪ <code>user</code> – for other user queries</li> </ul>
WOS_ALLOCATION_REGION	VARCHAR	The block size allocated by region in KB. The summary line sums the amount of memory used by all regions.
REGION_VIRTUAL_SIZE_KB	INTEGER	The amount of virtual memory in use by region in KB. Virtual size is greater than or equal to allocated size, which is greater than or equal to in-use size.
REGION_ALLOCATED_SIZE_KB	INTEGER	The amount of physical memory in use by a particular region in KB.
REGION_IN_USE_SIZE_KB	INTEGER	The actual number of bytes of data stored by the region in KB.
REGION_SMALL_RELEASE_COUNT	INTEGER	Internal use only.
REGION_BIG_RELEASE_COUNT	INTEGER	Internal use only.

### Notes

- The WOS allocator can use large amounts of virtual memory without assigning physical memory.
- To see the difference between virtual size and allocated size, look at the `REGION_IN_USE_SIZE` column to see if the WOS is full. The summary line tells you the amount of memory used by the WOS, which is typically capped at one quarter of physical memory per node.

### Examples

```
=>\pset expanded
Expanded display is on.
=> SELECT * FROM WOS_CONTAINER_STORAGE;

-[ RECORD 1 ]-----+-----
node_name          | host01
wos_type           | user
wos_allocation_region | 16 KB Region
region_virtual_size_kb | 2045408
region_allocated_size_kb | 0
region_in_use_size_kb | 0
```

```

region_small_release_count | 656
region_big_release_count  | 124
-[ RECORD 2 ]-----+-----
node_name                  | host01
wos_type                   | system
wos_allocation_region      | 16 KB Region
region_virtual_size_kb    | 1024
region_allocated_size_kb  | 960
region_in_use_size_kb     | 0
region_small_release_count | 78
region_big_release_count  | 9
-[ RECORD 3 ]-----+-----
node_name                  | host01
wos_type                   | system
wos_allocation_region      | 64 KB Region
region_virtual_size_kb    | 1024
region_allocated_size_kb  | 64
region_in_use_size_kb     | 0
region_small_release_count | 19
region_big_release_count  | 0
-[ RECORD 4 ]-----+-----
node_name                  | host01
wos_type                   | system
wos_allocation_region      | Summary
region_virtual_size_kb    | 2048
region_allocated_size_kb  | 1024
region_in_use_size_kb     | 0
region_small_release_count | 97
region_big_release_count  | 9
-[ RECORD 5 ]-----+-----
node_name                  | host01
wos_type                   | user
wos_allocation_region      | Summary
region_virtual_size_kb    | 2045408
region_allocated_size_kb  | 0
region_in_use_size_kb     | 0
region_small_release_count | 656
region_big_release_count  | 124
-[ RECORD 6 ]-----+-----
node_name                  | host02
wos_type                   | user
wos_allocation_region      | 16 KB Region
region_virtual_size_kb    | 2045408
region_allocated_size_kb  | 0
region_in_use_size_kb     | 0
region_small_release_count | 666
region_big_release_count  | 121
-[ RECORD 7 ]-----+-----
node_name                  | host02
wos_type                   | system
wos_allocation_region      | 16 KB Region
region_virtual_size_kb    | 1024
region_allocated_size_kb  | 960
region_in_use_size_kb     | 0
region_small_release_count | 38

```

```
region_big_release_count | 2
-[ RECORD 8 ]-----+-----
node_name                 | host02
wos_type                  | system
wos_allocation_region     | 64 KB Region
region_virtual_size_kb   | 1024
region_allocated_size_kb | 64
region_in_use_size_kb    | 0
region_small_release_count | 10
region_big_release_count  | 0
-[ RECORD 9 ]-----+-----
...
```

# Appendix: Compatibility with Other RDBMS

This section describes compatibility of Vertica with other relational database management systems.

Information in this appendix is intended to simplify database migration to Vertica.

## Data Type Mappings Between Vertica and Oracle

Oracle uses proprietary data types for all main data types (for example, VARCHAR, INTEGER, FLOAT, DATE), if you plan to migrate your database from Oracle to Vertica, Vertica strongly recommends that you convert the schema—a simple and important exercise that can minimize errors and time lost spent fixing erroneous data issues.

The following table compares the behavior of Oracle data types to Vertica data types.

Oracle	Vertica	Notes
NUMBER (no explicit precision)	INT, NUMERIC or FLOAT	<p>In Oracle, the NUMBER data type with no explicit precision stores each number N as an integer M, together with a scale S. The scale can range from -84 to 127, while the precision of M is limited to 38 digits. So <math>N = M * 10^S</math>.</p> <p>When precision is specified, precision/scale applies to all entries in the column. If omitted, the scale defaults to 0.</p> <p>For the common case where Oracle's NUMBER with no explicit precision data type is used to store only integer values, INT is the best suited and the fastest Vertica data type. However, INT (the same as BIGINT) is limited to a little less than 19 digits, with a scale of 0; if the Oracle column contains integer values outside of the range [-9223372036854775807, +9223372036854775807], use the Vertica data type NUMERIC(p,0) where p is the maximum number of digits required to represent the values of N.</p> <p>Even though no explicit scale is specified for an Oracle NUMBER column, Oracle allows non-integer values, each with its own scale. If the data stored in the column is approximate, Vertica recommends using the Vertica data type FLOAT, which is standard IEEE floating point, like ORACLE BINARY_DOUBLE. If the data is exact with fractional places, for example dollar amounts, Vertica recommends NUMERIC(p,s) where p is the precision (total number of digits) and s is the maximum scale (number of decimal places).</p> <p>Vertica conforms to standard SQL, which requires that <math>p \geq s</math> and <math>s \geq 0</math>. Vertica's NUMERIC data type is most effective for <math>p=18</math>, and increasingly expensive for <math>p=37, 58, 67</math>, etc., where <math>p \leq 1024</math>.</p> <p>Vertica recommends against using the data type NUMERIC(38,s) as a default "failsafe" mapping to guarantee no loss of precision. NUMERIC(18,s) is better, and INT or FLOAT are better yet, if one of these data types will do the job.</p>

NUMBER (P, 0), P <= 18	INT	In Oracle, when precision is specified the precision/scale applies to all entries in the column. If omitted the scale defaults to 0. For the Oracle NUMBER data type with 0 scale, and a precision less than or equal to 18, use INT in Vertica.
NUMBER (P, 0), P > 18	NUMERIC (p, 0)	An Oracle column precision greater than 18 is often more than an application really needs. If all values in the Oracle column are within the INT range [-9223372036854775807,+9223372036854775807], use INT for best performance. Otherwise, use the Vertica data type NUMERIC(p, 0), where p = P.
NUMBER (P, S) all cases other than previous 3 rows	NUMERIC (p, s) or FLOAT	When P >= S and S >= 0, use p = P and s = S, unless the data allows reducing P or using FLOAT as discussed above. If S > P, use p = S, s = S. If S < 0, use p = P - S, s = 0.
NUMERIC (P, S)	See notes -->	Rarely used in Oracle. See notes for the NUMBER type.
DECIMAL (P, S)	See notes -->	DECIMAL is a synonym for NUMERIC. See notes for the NUMBER type.
BINARY_FLOAT	FLOAT	Same as FLOAT(53) or DOUBLE PRECISION.
BINARY_DOUBLE	FLOAT	Same as FLOAT(53) or DOUBLE PRECISION.
RAW	VARBINARY (RAW )	The maximum size of RAW in Oracle is 2,000 bytes. The maximum size of CHAR/BINARY in Vertica is 65000 bytes. In Vertica, RAW is a synonym for VARBINARY.
LONG RAW	VARBINARY (RAW )	The maximum size of Oracle's LONG RAW is 2GB. The maximum size of Vertica's VARBINARY is 65000 bytes. Vertica user should exercise caution to avoid truncation during data migration from Oracle.
CHAR (n)	CHAR (n)	The maximum size of CHAR in Oracle is 2,000 bytes. The maximum size of CHAR in Vertica is 65000 bytes.
NCHAR (n)	CHAR (n*3)	Vertica supports national characters with CHAR(n) as variable-length UTF8-encoded UNICODE character string. UTF-8 represents ASCII in 1 byte, most European characters in 2 bytes, and most oriental and Middle Eastern characters in 3 bytes.
VARCHAR2 (n)	VARCHAR (n)	The maximum size of VARCHAR2 in Oracle is 4,000 bytes. The maximum size of VARCHAR in Vertica is 65000 . <b>Note:</b> The behavior of Oracle's VARCHAR2 and Vertica's VARCHAR is semantically different. Vertica's VARCHAR exhibits standard SQL behavior, whereas Oracle's VARCHAR2 is not completely consistent with standard behavior – it treats an empty string as NULL value and uses non-padded comparison if one operand is VARCHAR2.
NVARCHAR2 (n)	VARCHAR (n*3)	See notes for NCHAR().
DATE	TIMESTAMP or possibly DATE	Oracle's DATE is different from the SQL standard DATE data type implemented by Vertica. Oracle's DATE includes the time (no

		fractional seconds), while Vertica DATE type includes only date per SQL specification.
TIMESTAMP	TIMESTAMP	TIMESTAMP defaults to six places, that is, to microseconds
TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH TIME ZONE	TIME ZONE defaults to the currently SET or system time zone.
INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH	Per the SQL standard, INTERVAL can be qualified with YEAR TO MONTH sub-type in Vertica.
INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND	In Vertica, DAY TO SECOND is the default sub-type for INTERVAL.



# Index

---

## A

About the Documentation • 2  
ABS • 290  
ACOS • 290  
ACTIVE\_EVENTS • 855  
ADD\_LOCATION • 388, 391, 454, 478, 479, 485, 583, 585, 587, 588, 589, 872  
ADD\_MONTHS • 92, 192  
ADVANCE\_EPOCH • 390, 424, 518, 531  
AGE\_IN\_MONTHS • 88, 193, 196  
AGE\_IN\_YEARS • 88, 194, 195  
Aggregate Expressions • 45, 114, 773  
Aggregate Functions • 45, 114, 238, 630, 779, 781  
Alphabetical List of Vertica Functions • 388  
ALTER FUNCTION • 598, 654, 655, 725, 850  
ALTER PROFILE • 600  
ALTER PROFILE RENAME • 602  
ALTER PROJECTION RENAME • 390, 424, 444, 445, 518, 531, 545, 546, 547, 600  
ALTER RESOURCE POOL • 602, 604, 675, 729, 803, 804, 905  
ALTER ROLE RENAME • 605  
ALTER SCHEMA • 606, 682  
ALTER SEQUENCE • 322, 323, 325, 507, 607, 686, 733  
ALTER TABLE • 396, 502, 610, 698, 711, 712  
ALTER USER • 616, 720, 721, 747, 762, 798, 805, 806  
ALTER\_LOCATION\_USE • 389, 390, 454, 584, 587  
Analytic Functions • 114, 119, 121, 127, 128, 129, 132, 133, 142, 143, 144, 145, 146, 148, 149, 152, 155, 157, 159, 161, 162, 164, 165, 166, 168, 170, 172, 174, 176, 177, 178, 179, 181, 182, 183, 630, 773  
ANALYZE\_CONSTRAINTS • 391, 419, 462, 498, 505, 507, 641, 649, 651, 846  
ANALYZE\_HISTOGRAM • 396, 399, 576, 579  
ANALYZE\_STATISTICS • 396, 397, 398, 399, 426, 438, 449, 576, 577, 578, 579, 582, 583, 710, 832, 837, 839, 840, 895  
ANALYZE\_WORKLOAD • 397, 398, 399, 400, 577, 578, 579, 580, 594, 929  
AND operator • 38

## Appendix

Compatibility with Other RDBMS • 935

ASCII • 326, 331

ASIN • 291

ATAN • 291

ATAN2 • 292

AUDIT • 404, 523, 848

AUDIT\_LICENSE\_SIZE • 408, 482, 527, 529

AUDIT\_LICENSE\_TERM • 408, 527

AVG [Aggregate] • 114, 119, 130, 142

AVG [Analytic] • 115, 141, 146

## B

Backslashes • 10, 21, 28, 61, 258, 358, 629, 793, 806

Basic names • 11

BB\_WITHIN • 264

BEARING • 263, 265

BETWEEN-predicate • 53, 777

BIGINT • 12, 99, 104

Binary Data Types • 39, 68, 188, 190, 191, 632, 694

Binary Operators • 35, 69, 70, 71

BINARY\_VARYING • 68

BIT\_AND • 36, 70, 71, 187

BIT\_LENGTH • 327, 330, 348, 353

BIT\_OR • 37, 70, 71, 189

BIT\_XOR • 70, 71, 190

BITCOUNT • 72, 252, 328, 370

BITSTRING\_TO\_BINARY • 69, 72, 252, 329, 370

Boolean Data Type • 38, 39, 54, 72, 694

Boolean Functions • 187

Boolean Operators • 38, 54, 72, 777, 789

Boolean-predicate • 38, 52, 54, 72, 777, 789

BTRIM • 329, 351, 362, 373

Built-in Pool Configuration • 677, 820

Built-in Pools • 603, 604, 672, 674, 675, 729, 840, 905

BYTEA • 68

## C

CANCEL\_REFRESH • 408, 566

CASE Expressions • 38, 45, 47, 312, 313, 315, 318, 319, 332, 773, 782

CAST • 12, 14, 35, 39, 44, 68, 189, 190, 252, 255

Catalog Management Functions • 388, 490

CBRT • 292

CEILING (CEIL) • 293, 298

CHAR • 73

- Character Data Types • 39, 69, 73, 694
- Character string literals • 21, 358, 629
- CHARACTER VARYING • 12, 73
- CHARACTER\_LENGTH • 328, 330, 348, 353
- Characters in hexadecimal, entering • 21
- CHORD\_TO\_ARC • 263, 266
- CHR • 327, 331
- CLEAR\_PROJECTION\_REFRESHES • 409, 410, 462, 463, 487, 547, 548, 549, 892, 893
- CLEAR\_QUERY\_REPOSITORY • 409, 509
- CLEAR\_RESOURCE\_REJECTIONS • 410, 510, 871, 911
- CLOCK\_TIMESTAMP • 196, 233, 245
- CLOSE\_ALL\_SESSIONS • 413, 428, 486, 512, 517, 566, 572, 864, 918
- CLOSE\_SESSION • 410, 416, 428, 512, 569, 570, 864, 918
- Cluster Scaling Functions • 388, 496
- COALESCE • 311, 313, 314, 315, 317, 318, 320
- Coercion • 17, 39, 68, 108
- Column length limits
  - fixed length • 11
  - variable length • 11
- Column References • 48, 56, 60, 120, 121, 161, 164, 299, 302, 670, 671, 703, 704, 705, 715, 716, 717, 773, 779
- COLUMN\_STORAGE • 421, 424, 531, 593, 857
- column-constraint • 323, 324, 505, 686, 694, 698, 842
- column-definition (table) • 688, 690, 694
- column-definition (temp table) • 689, 706, 707, 709, 711
- column-name-list (table) • 688, 695
- column-name-list (temp table) • 706, 713
- COLUMNS • 821, 832
- Columns per table • 11
- column-value-predicate • 55, 777
- COMMENT ON COLUMN • 617
- COMMENT ON CONSTRAINT • 618
- COMMENT ON FUNCTION • 619
- COMMENT ON LIBRARY • 620
- COMMENT ON NODE • 621
- COMMENT ON PROJECTION • 622
- COMMENT ON SCHEMA • 623
- COMMENT ON SEQUENCE • 624
- COMMENT ON TABLE • 624
- COMMENT ON VIEW • 625
- Comments • 49
- COMMENTS • 618, 619, 620, 621, 622, 623, 624, 625, 626, 823
- COMMIT • 627, 642
- Commutative • 38
- Comparison Operators • 38, 55, 789
- Compound key • 614
- CONCAT • 331
- Concurrent connections
  - per cluster • 11
  - per node • 11
- CONDITIONAL\_CHANGE\_EVENT [Analytic] • 143, 144, 590
- CONDITIONAL\_TRUE\_EVENT [Analytic] • 143, 144, 590
- CONFIGURATION\_PARAMETERS • 413, 416, 515, 569, 572, 859
- CONNECT • 627, 649, 724, 743
- Constraint Management Functions • 388, 498
- COPY • 19, 24, 71, 73, 101, 102, 105, 383, 392, 396, 498, 502, 629, 649, 693, 700, 752, 756, 766
- COPY Formats • 641
- COPY FROM VERTICA • 627, 628, 649
- Copyright Notice • 949
- CORR • 115
- COS • 294
- COT • 294
- COUNT [Aggregate] • 115, 130, 146
- COUNT [Analytic] • 142, 145
- COVAR\_POP • 119
- COVAR\_SAMP • 120
- CREATE FUNCTION (SQL Macros) • 599, 652, 725, 850
- CREATE FUNCTION (UDF) • 655, 849
- CREATE LIBRARY • 655, 656, 658, 717, 718, 726
- CREATE PROCEDURE • 659, 727
- CREATE PROFILE • 660, 829
- CREATE PROJECTION • 663, 728, 832, 837
- CREATE RESOURCE POOL • 605, 672, 729, 803, 804, 840, 841
- CREATE ROLE • 681
- CREATE SCHEMA • 607, 681
- CREATE SEQUENCE • 322, 323, 324, 325, 505, 507, 608, 682, 702, 733, 842, 843
- CREATE TABLE • 324, 396, 424, 502, 505, 532, 642, 682, 688, 690, 709, 711, 755, 800, 842
- CREATE TEMPORARY TABLE • 690, 693, 706

- CREATE TRANSFORM FUNCTION • 717, 732  
 CREATE USER • 605, 617, 675, 719, 745, 759,  
 802, 803, 804, 805, 806  
 CREATE VIEW • 653, 721  
 CRITICAL\_NODES • 860  
 CUME\_DIST [Analytic] • 146, 168, 171  
 CURRENT\_DATABASE • 381  
 CURRENT\_DATE • 51, 194, 195, 196  
 CURRENT\_SCHEMA • 381, 416  
 CURRENT\_SESSION • 861  
 CURRENT\_TIME • 51, 197  
 CURRENT\_TIMESTAMP • 52, 198, 228, 229,  
 245  
 CURRENT\_USER • 382, 383, 384  
 CURRVAL • 322, 608, 683, 684, 686, 733
- D**
- Data Collector Functions • 388, 507, 865, 868  
 Data Type Coercion • 19, 40, 74, 108  
 Data Type Coercion Chart • 111, 188, 189, 190  
 Data Type Coercion Operators (CAST) • 17, 18,  
 39, 70, 72, 111, 112  
 Data Type Mappings Between Vertica and Oracle  
 • 935  
 DATA\_COLLECTOR • 442, 483, 484, 507, 508,  
 509, 864  
 Database Management Functions • 388, 509  
 Database size • 11  
 DATABASE\_CONNECTIONS • 868  
 DATABASE\_SNAPSHOT • 477, 510, 514  
 DATABASE\_SNAPSHOTS • 477, 514, 869  
 DATABASES • 824  
 DATE • 29, 76, 203, 205  
 Date/Time Data Types • 39, 75, 199, 632, 694  
 Date/Time Expressions • 29, 50, 210, 218, 219,  
 233, 241, 242  
 Date/Time Functions • 191, 630  
 Date/Time Literals • 29, 792  
 Date/Time Operators • 40  
 DATE\_PART • 198, 215, 218  
 DATE\_TRUNC • 203  
 DATEDIFF • 205  
 DATETIME • 77  
 DAY • 210  
 Day of the Week Names • 30  
 DAYOFMONTH • 211  
 DAYOFWEEK • 211  
 DAYOFWEEK\_ISO • 212, 213, 248, 250  
 DAYOFYEAR • 213
- DAYS • 214  
 DECIMAL • 12, 99, 104  
 DECODE • 332  
 DEGREES • 262, 295, 303  
 DELETE • 10, 431, 542, 710, 711, 723, 735, 752,  
 814, 819  
 DELETE\_VECTORS • 869  
 DENSE\_RANK [Analytic] • 147, 172, 174  
 Depth of nesting subqueries • 11  
 DISABLE\_DUPLICATE\_KEY\_ERROR • 392,  
 396, 417, 462, 498, 502, 503, 507  
 DISABLE\_LOCAL\_SEGMENTS • 419, 496  
 DISCONNECT • 627, 628, 724  
 DISK\_RESOURCE\_REJECTIONS • 410, 510,  
 870, 911  
 DISK\_STORAGE • 389, 584, 871  
 DISPLAY\_LICENSE • 420, 529  
 DISTANCE • 262, 295  
 DISTANCEV • 262, 296  
 DO\_TM\_TASK • 420, 424, 428, 429, 430, 456,  
 458, 512, 532, 533, 534, 536, 537, 592, 930  
 DOUBLE PRECISION • 17  
 DOUBLE PRECISION (FLOAT) • 17, 42, 101,  
 105, 214, 290, 304  
 Doubled single quotes • 21, 358  
 DROP FUNCTION • 599, 654, 655, 658, 725,  
 850  
 DROP LIBRARY • 726  
 DROP PROCEDURE • 660, 726  
 DROP PROFILE • 727  
 DROP PROJECTION • 450, 520, 693, 728, 735  
 DROP RESOURCE POOL • 605, 675, 729, 803,  
 804  
 DROP ROLE • 730  
 DROP SCHEMA • 607, 682, 730  
 DROP SEQUENCE • 322, 323, 325, 507, 608,  
 684, 686, 732  
 DROP TABLE • 692, 708, 711, 724, 728, 734,  
 814  
 DROP TRANSFORM FUNCTION • 731  
 DROP USER • 720, 721, 736  
 DROP VIEW • 721, 722, 736  
 DROP\_LOCATION • 421, 478, 585, 588  
 DROP\_PARTITION • 421, 422, 429, 430, 456,  
 458, 513, 530, 533, 534, 536, 537, 593, 693  
 DROP\_STATISTICS • 398, 400, 424, 438, 449,  
 578, 580, 583  
 DUAL • 824  
 DUMP\_CATALOG • 427, 491  
 DUMP\_LOCKTABLE • 428, 512, 888

DUMP\_PARTITION\_KEYS • 421, 424, 428, 429, 456, 458, 512, 532, 533, 536, 537, 593, 693  
DUMP\_PROJECTION\_PARTITION\_KEYS • 421, 424, 429, 430, 456, 458, 513, 532, 534, 536, 537, 593, 693  
DUMP\_TABLE\_PARTITION\_KEYS • 421, 424, 429, 430, 431, 456, 458, 513, 532, 533, 534, 536, 537, 593, 693  
DWITHIN • 264, 266, 272

## E

ECEF\_CHORD • 263, 267  
ECEF\_x • 263, 268  
ECEF\_y • 263, 268  
ECEF\_z • 263, 269  
ENABLE\_LOCAL\_SEGMENTS • 431, 497  
encoding-type • 663, 667, 691, 694, 695, 712, 713  
Epoch Management Functions • 388, 517  
EVALUATE\_DELETE\_PERFORMANCE • 431, 542, 837  
EVENT\_CONFIGURATIONS • 875  
EVENT\_NAME • 434, 453, 459, 538, 540, 541, 791  
Examples • 643  
EXECUTION\_ENGINE\_PROFILES • 738, 743, 864, 876  
EXP • 297  
EXPLAIN • 738  
EXPONENTIAL\_MOVING\_AVERAGE [Analytic] • 149  
EXPORT • 627, 628, 743  
EXPORT\_CATALOG • 435, 491  
EXPORT\_OBJECTS • 436, 492  
EXPORT\_STATISTICS • 398, 400, 426, 437, 449, 578, 580, 582, 583  
EXPORT\_TABLES • 435, 438, 491, 513  
Expressions • 44  
Extended String Literals • 19, 23, 24  
Extended string syntax • 21, 629  
EXTRACT • 199, 202, 214, 252

## F

FALSE • 38  
FIRST\_VALUE [Analytic] • 151, 158, 238  
fixed length • 11  
FLOAT • 12, 99, 101, 107, 130, 847  
FLOAT(n) • 99, 101, 107  
FLOAT8 • 99, 101, 107

FLOOR • 293, 297  
FOREIGN\_KEYS • 825  
Formatting Functions • 251, 630  
FROM Clause • 55, 56, 60, 66, 116, 772, 775, 818

## G

Geospatial Package SQL Functions • 261  
Geospatial SQL Functions • 262, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282  
GET\_AHM\_EPOCH • 439, 518  
GET\_AHM\_TIME • 440, 518  
GET\_AUDIT\_TIME • 441, 527  
GET\_COMPLIANCE\_STATUS • 408, 440, 527, 528  
GET\_CURRENT\_EPOCH • 441, 519  
GET\_DATA\_COLLECTOR\_POLICY • 442, 507  
GET\_LAST\_GOOD\_EPOCH • 442, 519  
GET\_NUM\_ACCEPTED\_ROWS • 443, 573  
GET\_NUM\_REJECTED\_ROWS • 443, 573  
GET\_PROJECTION\_STATUS • 443, 445, 544, 547, 665, 728  
GET\_PROJECTIONS, GET\_TABLE\_PROJECTIONS • 444, 545, 665, 728  
GETDATE • 218, 233  
GETUTCDATE • 219  
GRANT (Database) • 681, 745  
GRANT (Function) • 599, 654, 655, 657, 658, 681, 718, 725, 746, 761, 850  
GRANT (Procedure) • 660, 681, 746, 761  
GRANT (Resource Pool) • 681, 747, 762, 804  
GRANT (Role) • 616, 748  
GRANT (Schema) • 681, 746, 750, 751, 752, 753  
GRANT (Sequence) • 608, 681, 686, 733, 750  
GRANT (Table) • 681, 750, 752, 764  
GRANT (Transform Function) • 753, 767  
GRANT (View) • 681, 722, 750, 753, 764  
GRANTS • 826  
GREATEST • 333, 346  
GREATESTB • 335, 347  
GROUP BY Clause • 73, 114, 772, 779, 781

## H

HAS\_TABLE\_PRIVILEGE • 382  
HASH • 298, 302, 669, 670, 703, 704, 715, 716  
hash-segmentation-clause • 663, 664, 669

hash-segmentation-clause (table) • 688, 689, 690, 703  
 hash-segmentation-clause (temp table) • 706, 707, 708, 709, 715  
 HAVING Clause • 114, 772, 781  
 HEX\_TO\_BINARY • 69, 70, 72, 336  
 Hexadecimal • 21, 68, 101, 255, 861, 915  
 HOST\_RESOURCES • 881  
 HOUR • 219

**I**

Identifiers • 15, 357, 604, 674  
 IFNULL • 312  
 IMPORT\_STATISTICS • 398, 400, 426, 438, 448, 578, 580, 582, 583  
 INET\_ATON • 72, 282, 284, 290, 337, 339, 380  
 INET\_NTOA • 72, 283, 338  
 INITCAP • 339  
 INITCAPB • 339  
 IN-predicate • 56, 777  
 INSERT • 340, 755  
 INSTALL\_LICENSE • 493  
 INSTR • 341, 344  
 INSTRB • 343  
 INT • 12, 99, 104  
 INT8 • 99, 104  
 INTEGER • 17, 104, 105  
 INTERPOLATE • 10, 56  
 INTERRUPT\_STATEMENT • 397, 398, 399, 400, 408, 409, 446, 566, 573, 577, 578, 579, 580  
 INTERVAL • 76, 77, 95, 194, 196, 206, 795  
 Interval Values • 30, 88, 617, 720, 804, 808  
 interval-literal • 32, 77, 80, 88, 794  
 interval-qualifier • 34, 76, 77, 80, 87, 90, 787  
 INTO Clause • 772, 773  
 IP Conversion Functions • 282  
 ISFINITE • 220  
 ISLEFT • 264, 269  
 ISNULL • 312, 313, 314, 318  
 ISO 8601 • 29  
 ISUTF8 • 344, 449, 552

**J**

joined-table • 775, 776  
 join-predicate • 60, 663, 776, 777  
 JULIAN\_DAY • 220

**K**

Key size • 11  
 Keywords • 12, 357  
 Keywords and Reserved Words • 12  
 KM2MILES • 264, 270

**L**

LAG [Analytic] • 154, 161  
 LAST\_DAY • 221  
 LAST\_INSERT\_ID • 323, 505  
 LAST\_VALUE [Analytic] • 154, 157, 238  
 LAT\_WITHIN • 264, 271  
 LCOPY • 629, 649, 756  
 LEAD [Analytic] • 155, 157, 158  
 LEAST • 335, 345  
 LEASTB • 336, 346  
 LEFT • 347  
 Length  
   Basic names • 11  
   Length for a variable-length column • 11  
 LENGTH • 36, 72, 328, 330, 348, 353  
 Length of basic names • 11  
 License Management Functions • 388, 523  
 LICENSE\_AUDITS • 828  
 LIKE-predicate • 61, 468, 552, 557, 777  
 LIMIT Clause • 772, 784, 785  
 Limits  
   Basic names • 11  
   Columns per table • 11  
   Concurrent connections per cluster • 11  
   Connections per node, number • 11  
   Database size • 11  
   Depth of nesting subqueries • 11  
   Key size • 11  
   Projections per database • 11  
   Row size • 11  
   Rows per load • 11  
   Table size • 11  
   Tables per database • 11  
   Variable-length column • 11  
 Literals • 17, 773  
 LL\_WITHIN • 264, 271  
 LLD\_WITHIN • 264, 272  
 LN • 299  
 LOAD\_STREAMS • 443, 573, 641, 883  
 LOCALTIME • 52, 222  
 LOCALTIMESTAMP • 52, 222

LOCKS • 392, 428, 498, 512, 773, 814, 885, 891, 915  
 LOG • 300  
 LON\_WITHIN • 264, 273  
 LOWER • 349  
 LOWERB • 349  
 LPAD • 350  
 LTRIM • 330, 351, 362, 373

## M

MAKE\_AHM\_NOW • 449, 481, 482, 519, 522, 523  
 MARK\_DESIGN\_KSAFE • 435, 450, 451, 487, 492, 495, 520, 549, 728  
 MATCH Clause • 434, 435, 452, 453, 458, 459, 538, 539, 540, 541, 788  
 MATCH\_ID • 435, 452, 459, 539, 541, 791  
 Mathematical Functions • 107, 290, 630  
 Mathematical Operators • 41, 773  
 MAX [Aggregate] • 70, 71, 120, 122, 162  
 MAX [Analytic] • 161, 164  
 MD5 • 351  
 MEASURE\_LOCATION\_PERFORMANCE • 453, 484, 485, 586, 589  
 MEDIAN [Analytic] • 162, 169, 170  
 MERGE\_PARTITIONS • 423, 424, 454, 460, 461, 530, 532, 534, 550, 551, 552, 930  
 MICROSECOND • 223  
 MIDNIGHT\_SECONDS • 223  
 MILES2KM • 264, 274  
 MIN [Aggregate] • 70, 71, 121, 164  
 MIN [Analytic] • 162, 163  
 MINUTE • 224  
 MOD • 42, 300  
 MODULARHASH • 299, 302, 669, 670, 703, 704, 715, 716  
 MONEY • 12, 99, 104  
 MONTH • 225  
 Month Names • 30  
 MONTHS\_BETWEEN • 226  
 Multi-column key • 614

## N

named\_windows • 140  
 NaN • 17, 52, 107  
 NEXT\_DAY • 227  
 NEXTVAL • 321, 322, 323, 609, 683, 684, 686, 733  
 NODE\_RESOURCES • 672, 705, 717, 888

NODES • 828  
 Nonstandard conforming strings • 21  
 Notes • 642  
 NOW [Date/Time] • 51, 228  
 NTILE [Analytic] • 164, 311  
 NULL Operators • 42  
 NULL Value • 52, 66, 72  
 NULL-handling Functions • 52, 311, 630  
 NULLIF • 315  
 NULLIFZERO • 316  
 NULL-predicate • 54, 55, 66, 777  
 Number of columns per table • 11  
 Number of connections per node • 11  
 Number of rows per load • 11  
 Number-type Literals • 17, 31  
 NUMERIC • 17, 104  
 Numeric Data Type Overflow • 107  
 Numeric Data Types • 39, 99, 115, 126, 127, 128, 129, 130, 131, 132, 133, 162, 176, 177, 178, 179, 180, 181, 182, 183, 694  
 Numeric Expressions • 18, 52  
 NVL • 312, 313, 314, 315, 317  
 NVL2 • 318

## O

Octal • 68, 629  
 OCTET\_LENGTH • 328, 330, 348, 352  
 OFFSET Clause • 772, 784, 785  
 Operators • 35  
 OR operator • 38  
 ORDER BY Clause • 114, 137, 138, 168, 170, 772, 779, 782, 784, 785  
 OVERLAPS • 229  
 OVERLAY • 353  
 OVERLAYB • 354

## P

Pacific Standard Time • 29  
 Parameters • 630  
 Partition Management Functions • 388, 530  
 PARTITION\_PROJECTION • 421, 424, 429, 430, 431, 455, 457, 458, 513, 532, 533, 534, 535, 536, 537, 593, 693  
 PARTITION\_TABLE • 424, 429, 430, 431, 456, 457, 460, 513, 532, 533, 534, 535, 536, 550, 693  
 PARTITIONS • 429, 513, 889  
 PASSWORDS • 829  
 PATH ID • 738, 876

- Pattern Matching Functions • 388, 538, 791  
 PATTERN\_ID • 435, 453, 458, 539, 540, 791  
 Pattern-matching predicates • 61  
 PERCENT\_RANK [Analytic] • 147, 166  
 PERCENTILE\_CONT [Analytic] • 163, 166, 168, 171  
 PERCENTILE\_DISC [Analytic] • 147, 170  
 Performance Optimization for Analytic Sort Computation • 135, 137, 184  
 PI • 302  
 POSITION • 355, 366  
 POSITIONB • 356, 366  
 POWER • 303  
 Predicates • 53  
 Preface • 9  
 PRIMARY\_KEYS • 829  
 Printing Full Books • 4  
 PROFILE • 757  
 PROFILE\_PARAMETERS • 830  
 PROFILE\$ • 830  
 Projection Management Functions • 388, 542  
 PROJECTION\_COLUMNS • 542, 832, 840  
 PROJECTION\_DELETE\_CONCERNS • 837  
 PROJECTION\_REFRESHES • 409, 410, 462, 463, 487, 488, 542, 547, 548, 549, 566, 676, 820, 888, 890  
 PROJECTION\_STORAGE • 542, 839, 840, 893  
 PROJECTIONS • 424, 430, 487, 488, 532, 533, 542, 549, 833, 837, 895  
 Projections per database • 11  
 PST • 29  
 PURGE • 459, 460, 461, 480, 521, 550, 551, 552, 930  
 Purge Functions • 388, 550  
 PURGE\_PROJECTION • 460, 550  
 PURGE\_TABLE • 460, 461, 550, 551, 552
- Q**
- QUARTER • 230  
 QUERY\_METRICS • 409, 479, 510, 515, 895  
 QUERY\_PROFILES • 409, 479, 510, 515, 864, 897, 899, 901, 904  
 QUOTE\_IDENT • 16, 357  
 QUOTE\_LITERAL • 358  
 Quoted identifiers • 15
- R**
- RADIANS • 262, 295, 303  
 RADIUS\_LON • 263, 274  
 RADIUS\_M • 263, 275  
 RADIUS\_N • 263, 275  
 radius\_r • 263, 276  
 RADIUS\_Ra • 263, 277, 278  
 RADIUS\_Rc • 263, 277  
 RADIUS\_Rv • 263, 277, 278  
 RADIUS\_SI • 263, 278  
 RANDOM • 304  
 RANDOMINT • 305  
 range-segmentation-clause • 663, 665, 670  
 range-segmentation-clause (table) • 688, 689, 690, 704  
 range-segmentation-clause (temp table) • 706, 707, 708, 709, 716  
 RANK [Analytic] • 148, 149, 172, 175  
 RAW • 12, 68  
 RAYCROSSING • 264, 279  
 Reading the Online Documentation • 2  
 REAL • 99, 101  
 REENABLE\_DUPLICATE\_KEY\_ERROR • 392, 417, 419, 461, 498, 503, 505, 507  
 REFRESH • 410, 462, 547  
 REGEXP\_COUNT • 464, 553  
 REGEXP\_INSTR • 466, 555  
 REGEXP\_LIKE • 344, 449, 468, 552, 557  
 REGEXP\_REPLACE • 472, 561  
 REGEXP\_SUBSTR • 475, 564  
 REGR\_AVGX • 122  
 REGR\_AVGY • 122  
 REGR\_COUNT • 123  
 REGR\_INTERCEPT • 123  
 REGR\_R2 • 124  
 REGR\_SLOPE • 124  
 REGR\_SXX • 125  
 REGR\_SXY • 125  
 REGR\_SYY • 126  
 Regular Expression Functions • 388, 552  
 RELEASE\_SAVEPOINT • 758, 770, 771  
 REMOVE\_DATABASE\_SNAPSHOT • 477, 514  
 REPEAT • 36, 72, 359  
 REPLACE • 360, 472, 561  
 Reserved Words • 14  
 RESOURCE\_ACQUISITIONS • 898, 899, 904, 909, 910  
 RESOURCE\_ACQUISITIONS\_HISTORY • 898, 899, 900, 901, 909, 910  
 RESOURCE\_POOL\_STATUS • 605, 901, 904, 905

RESOURCE\_POOLS • 605, 674, 840, 901, 904, 909, 910  
 RESOURCE\_QUEUES • 901, 904, 909  
 RESOURCE\_REJECTIONS • 410, 510, 871, 901, 904, 909, 910  
 RESOURCE\_USAGE • 911  
 RESTORE\_LOCATION • 477, 478, 479, 587, 588, 589  
 RETIRE\_LOCATION • 389, 391, 422, 454, 478, 584, 585, 586, 587, 588  
 REVOKE (Database) • 759  
 REVOKE (Function) • 599, 654, 655, 657, 658, 718, 725, 746, 760, 850  
 REVOKE (Procedure) • 747, 761  
 REVOKE (Resource Pool) • 729, 748, 762  
 REVOKE (Role) • 762  
 REVOKE (Schema) • 764  
 REVOKE (Sequence) • 764  
 REVOKE (Table) • 766  
 REVOKE (Transform Function) • 767  
 REVOKE (View) • 722, 767  
 RIGHT • 360  
 ROLES • 841  
 ROLLBACK • 642, 769  
 ROLLBACK TO SAVEPOINT • 759, 769, 771  
 ROUND • 305  
 ROUND [Date/Time] • 230  
 Row size • 11  
 ROW\_NUMBER [Analytic] • 143, 174  
 RPAD • 361  
 RTRIM • 330, 351, 362, 373

## S

SAVE\_QUERY\_REPOSITORY • 479, 514  
 SAVEPOINT • 759, 770  
 SECOND • 231  
 See Also • 649  
 SELECT • 114, 689, 693, 721, 722, 724, 743, 744, 755, 772, 817, 888  
 Sequence Functions • 321  
 SEQUENCES • 325, 507, 842  
 Session Management Functions • 388, 566  
 SESSION\_PROFILES • 864, 888, 914  
 SESSION\_USER • 382, 383  
 SESSIONS • 409, 411, 413, 416, 428, 448, 486, 512, 517, 566, 567, 569, 570, 572, 576, 864, 915  
 SET DATESTYLE • 76, 80, 88, 440, 482, 519, 523, 792, 794, 811

SET ESCAPE\_STRING\_WARNING • 21, 23, 27, 793, 807, 811  
 SET INTERVALSTYLE • 76, 77, 80, 88, 794, 811  
 SET LOCALE • 795, 811, 860  
 SET ROLE • 798, 811  
 SET SEARCH\_PATH • 49, 682, 688, 706, 799, 811  
 SET SESSION CHARACTERISTICS • 10, 801, 811  
 SET SESSION MEMORYCAP • 605, 675, 802, 804, 811  
 SET SESSION RESOURCE POOL • 605, 675, 747, 803, 811  
 SET SESSION RUNTIMECAP • 804, 811  
 SET SESSION TEMPSPACECAP • 805, 811  
 SET STANDARD\_CONFORMING\_STRINGS • 20, 21, 23, 24, 27, 793, 806, 811  
 SET TIME\_ZONE • 807, 811  
 SET\_AHM\_EPOCH • 450, 479, 482, 520, 523  
 SET\_AHM\_TIME • 450, 479, 481, 520, 521, 522  
 SET\_AUDIT\_TIME • 441, 482, 527, 529  
 SET\_CONFIG\_PARAMETER • 515  
 SET\_DATA\_COLLECTOR\_POLICY • 483, 508  
 SET\_LOCATION\_PERFORMANCE • 484, 589  
 SET\_LOGLEVEL • 516  
 SET\_SCALING\_FACTOR • 485, 497  
 SHOW • 77, 792, 794, 808, 811  
 SHUTDOWN • 412, 413, 416, 485, 516, 569, 572  
 SIGN • 307  
 SIN • 307  
 Single quotes, doubled • 21, 358  
 SMALLDATETIME • 91  
 SMALLINT • 12, 99, 104  
 SPACE • 362  
 SPLIT\_PART • 363  
 SPLIT\_PARTB • 364  
 SQL Data Types • 67, 598, 649, 653, 659, 725, 731  
 SQL Functions • 113, 773  
 SQL Language Elements • 12  
 SQL Overview • 10  
 SQL Statements • 598  
 SQL System Tables (Monitoring APIs) • 452, 496, 676, 820  
 SQRT • 308  
 Standard conforming strings • 10, 21, 629, 793, 806, 811

- 
- START\_REFRESH • 409, 410, 463, 487, 548, 566, 666  
 STATEMENT\_TIMESTAMP • 196, 218, 232, 233, 245  
 Statistic Management Functions • 388, 576  
 Statistical analysis • 126, 127, 128, 131, 132  
 STDDEV [Aggregate] • 126, 128, 129, 176, 178, 179  
 STDDEV [Analytic] • 175, 179  
 STDDEV\_POP [Aggregate] • 127, 178  
 STDDEV\_POP [Analytic] • 177  
 STDDEV\_SAMP [Aggregate] • 126, 127, 128, 176, 179  
 STDDEV\_SAMP [Analytic] • 175, 176, 178  
 Storage Management Functions • 388, 583  
 STORAGE\_CONTAINERS • 460, 461, 550, 551, 552, 918  
 STRATA • 921, 924  
 STRATA\_STRUCTURES • 921, 924  
 String Concatenation Operators • 43, 773  
 String Functions • 326, 630  
 String literals • 10, 77, 358, 629, 793, 806  
 String Literals • 19, 635, 642  
 String Literals (Character) • 21, 359, 636, 638  
 String Literals (Dollar-Quoted) • 28  
 Strings, standard conforming • 10, 629, 793, 806, 811  
 STRPOS • 365  
 STRPOSB • 366  
 SUBSTR • 348, 361, 367  
 SUBSTRB • 368  
 SUBSTRING • 36, 72, 367, 368  
 Suggested Reading Paths • 2, 5  
 SUM [Aggregate] • 104, 107, 115, 119, 129, 180  
 SUM [Analytic] • 142, 146, 179  
 SUM\_FLOAT [Aggregate] • 104, 107, 130, 179  
 SYSDATE • 218, 233  
 SYSTEM • 452, 480, 481, 496, 521, 522, 925  
 System Information Functions • 381, 630  
 System Limits • 11  
 SYSTEM\_TABLES • 843
- T**
- Table size • 11  
 TABLE\_CONSTRAINTS • 498, 845  
 table-constraint • 610, 611, 614, 694, 702  
 table-primary • 775, 776  
 table-reference • 775  
 TABLES • 17, 62, 821, 846  
 Tables per database • 11  
 TAN • 308  
 Technical Support • 1, 4, 427, 428, 491, 512, 738, 797, 859, 860, 865, 886  
 Template Pattern Modifiers for Date/Time Formatting • 255, 257, 258, 260, 290, 632  
 Template Patterns for Date/Time Formatting • 192, 252, 254, 256, 257, 258, 632  
 Template Patterns for Numeric Formatting • 252, 254, 256, 257, 260  
 TIME • 29, 91, 206  
 TIME AT TIME\_ZONE • 91, 92, 93  
 Time Zone Names for Setting TIME\_ZONE • 808  
 Time Zone Values • 29  
 TIME\_SLICE • 154, 158, 233, 787, 788  
 TIMEOFDAY • 238  
 TIMESERIES Clause • 151, 238, 488, 489, 490, 590, 591, 592, 772, 786  
 Timeseries Functions • 388, 589  
 TIMESTAMP • 77, 91, 93, 205, 440, 481, 482, 519, 522, 523  
 TIMESTAMP AT TIME\_ZONE • 98  
 TIMESTAMP\_ROUND • 242  
 TIMESTAMP\_TRUNC • 243  
 TIMESTAMPADD • 239  
 TIMESTAMPDIF • 241  
 TINYINT • 12, 99, 104  
 TO\_BITSTRING • 72, 251, 370  
 TO\_CHAR • 252  
 TO\_DATE • 254, 632  
 TO\_HEX • 36, 70, 72, 255, 336, 337, 370  
 TO\_NUMBER • 257  
 TO\_TIMESTAMP • 255  
 TRANSACTION\_TIMESTAMP • 196, 232, 233, 245  
 TRANSLATE • 371  
 TRIM • 330, 351, 362, 372  
 TRUE • 38  
 TRUNC • 203, 309  
 TRUNC [Date/Time] • 245  
 TRUNCATE TABLE • 710, 724, 735, 813  
 TS\_FIRST\_VALUE • 238, 488, 490, 590, 592, 787, 788  
 TS\_LAST\_VALUE • 238, 489, 591, 787, 788  
 TUNING\_RECOMMENDATIONS • 403, 404, 596, 597, 927  
 Tuple Mover Functions • 388, 592  
 TUPLE\_MOVER\_OPERATIONS • 929  
 TYPES • 847  
 Typographical Conventions • 7

**U**

Unicode characters • 806  
Unicode String Literals • 27  
UNION • 772, 814  
Unquoted identifiers • 15  
UPDATE • 10, 724, 752, 818, 819  
UPPER • 373  
UPPERB • 374  
URI Encode/Decode Functions • 385  
URI\_PERCENT\_DECODE • 385  
URI\_PERCENT\_ENCODE • 386  
USER • 382, 384  
USER\_AUDITS • 848  
USER\_FUNCTIONS • 599, 654, 655, 657, 658, 725, 849  
USER\_LIBRARIES • 930  
USER\_LIBRARY\_MANIFEST • 931  
USER\_PROCEDURES • 850  
USER\_TRANSFORMS • 718, 850  
USERS • 851  
UTC • 29

**V**

V\_CATALOG Schema • 820, 821, 843  
V\_MONITOR Schema • 820, 843, 855  
V6\_ATON • 72, 284, 286, 374, 376  
V6\_NTOA • 72, 285, 375, 376  
V6\_SUBNETA • 72, 286, 288, 377, 378  
V6\_SUBNETN • 72, 287, 377  
V6\_TYPE • 72, 288, 378  
VAR\_POP [Aggregate] • 131, 181  
VAR\_POP [Analytic] • 180  
VAR\_SAMP [Aggregate] • 131, 132, 133, 183, 184  
VAR\_SAMP [Analytic] • 181, 183, 184  
VARBINARY • 35, 68, 847  
VARCHAR2 • 12  
variable length • 11  
VARIANCE [Aggregate] • 132, 184  
VARIANCE [Analytic] • 182, 183  
VERSION • 385  
Vertica Functions • 113, 388, 630  
VIEW\_COLUMNS • 852, 855  
VIEWS • 854

**W**

WEEK • 246

WEEK\_ISO • 213, 247, 250  
WGS84\_a • 263, 280  
WGS84\_b • 263, 280  
WGS84\_e2 • 263, 281  
WGS84\_f • 263, 281  
WGS84\_if • 263, 281  
WGS84\_r1 • 263, 282  
WHERE Clause • 723, 752, 772, 777, 818, 819  
Where to Find Additional Information • 6  
Where to Find the Vertica Documentation • 2  
WIDTH\_BUCKET • 166, 310  
window\_frame\_clause • 134, 138, 141, 145, 151, 157, 161, 163, 176, 177, 178, 179, 180, 182, 183  
window\_order\_clause • 134, 137, 141, 143, 144, 145, 146, 148, 149, 151, 154, 157, 158, 161, 163, 165, 166, 172, 174, 176, 177, 178, 179, 180, 182, 183, 788  
window\_partition\_clause • 134, 135, 137, 138, 141, 143, 144, 145, 146, 148, 149, 151, 154, 157, 158, 161, 162, 163, 165, 166, 168, 170, 172, 174, 176, 177, 178, 179, 180, 182, 183, 786  
Workload Management Functions • 388, 593  
WOS\_CONTAINER\_STORAGE • 931

**Y**

YEAR • 248  
YEAR\_ISO • 248, 249

**Z**

ZEROIFNULL • 320  
Zulu • 29

# Copyright Notice

---

Copyright© 2006-2011 Vertica, An HP Company, and its licensors. All rights reserved.

Vertica, An HP Company  
8 Federal Street  
Billerica, MA 01821  
Phone: (978) 600-1000  
Fax: (978) 600-1001  
E-Mail: [info@vertica.com](mailto:info@vertica.com)  
Web site: <http://www.vertica.com>  
(<http://www.vertica.com>)

The software described in this copyright notice is furnished under a license and may be used or copied only in accordance with the terms of such license. Vertica, An HP Company software contains proprietary information, as well as trade secrets of Vertica, An HP Company, and is protected under international copyright law. Reproduction, adaptation, or translation, in whole or in part, by any means — graphic, electronic or mechanical, including photocopying, recording, taping, or storage in an information retrieval system — of any part of this work covered by copyright is prohibited without prior written permission of the copyright owner, except as allowed under the copyright laws.

This product or products depicted herein may be protected by one or more U.S. or international patents or pending patents.

## Trademarks

Vertica™, the Vertica® Analytic Database™, and FlexStore™ are trademarks of Vertica, An HP Company.

Adobe®, Acrobat®, and Acrobat® Reader® are registered trademarks of Adobe Systems Incorporated.

AMD™ is a trademark of Advanced Micro Devices, Inc., in the United States and other countries.

DataDirect® and DataDirect Connect® are registered trademarks of Progress Software Corporation in the U.S. and other countries.

Fedora™ is a trademark of Red Hat, Inc.

Intel® is a registered trademark of Intel.

Linux® is a registered trademark of Linus Torvalds.

Microsoft® is a registered trademark of Microsoft Corporation.

Novell® is a registered trademark and SUSE™ is a trademark of Novell, Inc., in the United States and other countries.

Oracle® is a registered trademark of Oracle Corporation.

Red Hat® is a registered trademark of Red Hat, Inc.

VMware® is a registered trademark or trademark of VMware, Inc., in the United States and/or other jurisdictions.

Other products mentioned may be trademarks or registered trademarks of their respective companies.

## Open Source Software Acknowledgments

Vertica makes no representations or warranties regarding any third party software. All third-party software is provided or recommended by Vertica on an AS IS basis.

This product includes cryptographic software written by Eric Young ([eay@cryptsoft.com](mailto:eay@cryptsoft.com)).

## ASMJIT

Copyright (c) 2008-2010, Petr Kobalicek <[kobalicek.petr@gmail.com](mailto:kobalicek.petr@gmail.com)>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## Boost

Boost Software License - Version 1.38 - February 8th, 2009

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## **bzip2**

This file is a part of bzip2 and/or libbzip2, a program and library for lossless, block-sorting data compression.

Copyright © 1996-2005 Julian R Seward. All rights reserved.

- 1 Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
- 2 Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 3 The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
- 4 Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
- 5 The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Julian Seward, Cambridge, UK.

[jseward@bzip.org](mailto:jseward@bzip.org) <<mailto:jseward@bzip.org>>

bzip2/libbzip2 version 1.0 of 21 March 2000

This program is based on (at least) the work of:

Mike Burrows

David Wheeler

Peter Fenwick

Alistair Moffat

Radioed Neal

Ian H. Witten

Robert Sedgewick

Jon L. Bentley

## **Daemonize**

Copyright © 2003-2007 Brian M. Clapper.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the clapper.org nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### **Ganglia Open Source License**

Copyright © 2001 by Matt Massie and The Regents of the University of California.  
All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without written agreement is hereby granted, provided that the above copyright notice and the following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

### **ICU (International Components for Unicode) License - ICU 1.8.1 and later**

COPYRIGHT AND PERMISSION NOTICE

Copyright © 1995-2009 International Business Machines Corporation and others  
All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

### **jQuery**

Copyright © 2009 John Resig, <http://jquery.com/>

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

### **Keepalived Vertica IPVS (IP Virtual Server) Load Balancer**

Copyright © 2007 Free Software Foundation, Inc.

<http://fsf.org/>

The keepalived software contained in the `VerticaIPVSLoadBalancer-5.0.x-0.RHEL5.x86_64.rpm` software package is licensed under the GNU General Public License ("GPL"). You are entitled to receive the source code for such software. For no less than three years from the date you obtained this software package, you may download a copy of the source code for the software in this package licensed under the GPL at no charge by visiting <http://www.vertica.com/licenses/keepalived-1.1.17.tar.gz> **<http://www.vertica.com/licenses/keepalived-1.1.17.tar.gz>**. You may download this source code so that it remains separate from other software on your computer system.

### Lighttpd Open Source License

Copyright © 2004, Jan Kneschke, incremental  
All rights reserved.

- 1 Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
- 2 Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 3 Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 4 Neither the name of the 'incremental' nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### MersenneTwister.h

Copyright © 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,  
Copyright © 2000 - 2009, Richard J. Wagner  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1 Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2 Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- 3 The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### MIT Kerberos

Copyright © 1985-2007 by the Massachusetts Institute of Technology.

Export of software employing encryption from the United States of America may require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Furthermore if you modify this software you must label your software as modified software and not distribute it in such a fashion that it might be confused with the original MIT software. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Individual source code files are copyright MIT, Cygnus Support, Novell, OpenVision Technologies, Oracle, Red Hat, Sun Microsystems, FundsXpress, and others.

Project Athena, Athena, Athena MUSE, Discuss, Hesiod, Kerberos, Moira, and Zephyr are trademarks of the Massachusetts Institute of Technology (MIT). No commercial use of these trademarks may be made without prior written permission of MIT.

"Commercial use" means use of a name in a product or other for-profit manner. It does NOT prevent a commercial firm from referring to the MIT trademarks in order to convey information (although in doing so, recognition of their trademark status should be given).

Portions of src/lib/crypto have the following copyright:

Copyright © 1998 by the FundsXpress, INC.

All rights reserved.

Export of this software from the United States of America may require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Funds Xpress. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. FundsXpress makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The implementation of the AES encryption algorithm in `src/lib/crypto/aes` has the following copyright:

Copyright © 2001, Dr Brian Gladman <[brg@gladman.uk.net](mailto:brg@gladman.uk.net)>, Worcester, UK.  
All rights reserved.

#### LICENSE TERMS

The free distribution and use of this software in both source and binary form is allowed (with or without changes) provided that:

- 1 Distributions of this source code include the above copyright notice, this list of conditions and the following disclaimer.
- 2 Distributions in binary form include the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other associated materials.
- 3 The copyright holder's name is not used to endorse products built using this software without specific written permission.

#### DISCLAIMER

This software is provided 'as is' with no explicit or implied warranties in respect of any properties, including, but not limited to, correctness and fitness for purpose.

The implementations of GSSAPI mechglue in GSSAPI-SPNEGO in `src/lib/gssapi`, including the following files:

- `lib/gssapi/generic/gssapi_err_generic.et`
- `lib/gssapi/mechglue/g_accept_sec_context.c`
- `lib/gssapi/mechglue/g_acquire_cred.c`
- `lib/gssapi/mechglue/g_canon_name.c`
- `lib/gssapi/mechglue/g_compare_name.c`
- `lib/gssapi/mechglue/g_context_time.c`
- `lib/gssapi/mechglue/g_delete_sec_context.c`
- `lib/gssapi/mechglue/g_dsp_name.c`
- `lib/gssapi/mechglue/g_dsp_status.c`
- `lib/gssapi/mechglue/g_dup_name.c`
- `lib/gssapi/mechglue/g_exp_sec_context.c`
- `lib/gssapi/mechglue/g_export_name.c`
- `lib/gssapi/mechglue/g_glue.c`
- `lib/gssapi/mechglue/g_imp_name.c`

- lib/gssapi/mechglue/g\_imp\_sec\_context.c
- lib/gssapi/mechglue/g\_init\_sec\_context.c
- lib/gssapi/mechglue/g\_initialize.c
- lib/gssapi/mechglue/g\_inquire\_context.c
- lib/gssapi/mechglue/g\_inquire\_cred.c
- lib/gssapi/mechglue/g\_inquire\_names.c
- lib/gssapi/mechglue/g\_process\_context.c
- lib/gssapi/mechglue/g\_rel\_buffer.c
- lib/gssapi/mechglue/g\_rel\_cred.c
- lib/gssapi/mechglue/g\_rel\_name.c
- lib/gssapi/mechglue/g\_rel\_oid\_set.c
- lib/gssapi/mechglue/g\_seal.c
- lib/gssapi/mechglue/g\_sign.c
- lib/gssapi/mechglue/g\_store\_cred.c
- lib/gssapi/mechglue/g\_unseal.c
- lib/gssapi/mechglue/g\_userok.c
- lib/gssapi/mechglue/g\_utils.c
- lib/gssapi/mechglue/g\_verify.c
- lib/gssapi/mechglue/gssd\_pname\_to\_uid.c
- lib/gssapi/mechglue/mglueP.h
- lib/gssapi/mechglue/oid\_ops.c
- lib/gssapi/spnego/gssapiP\_spnego.h
- lib/gssapi/spnego/spnego\_mech.c

are subject to the following license:

Copyright © 2004 Sun Microsystems, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

### **Npgsql-.Net Data Provider for Postgresql**

Copyright © 2002-2008, The Npgsql Development Team

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE NPGSQL DEVELOPMENT TEAM BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE NPGSQL DEVELOPMENT TEAM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE NPGSQL DEVELOPMENT TEAM SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE NPGSQL DEVELOPMENT TEAM HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

### **Open LDAP**

The OpenLDAP Public License  
Version 2.8, 17 August 2003

Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met:

- 1** Redistributions in source form must retain copyright statements and notices,
- 2** Redistributions in binary form must reproduce applicable copyright statements and notices, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution, and
- 3** Redistributions must contain a verbatim copy of this document.

The OpenLDAP Foundation may revise this license from time to time. Each revision is distinguished by a version number. You may use this Software under terms of this license revision or under the terms of any subsequent revision of the license.

THIS SOFTWARE IS PROVIDED BY THE OPENLDAP FOUNDATION AND ITS CONTRIBUTORS ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OPENLDAP FOUNDATION, ITS CONTRIBUTORS, OR THE AUTHOR(S) OR OWNER(S) OF THE SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The names of the authors and copyright holders must not be used in advertising or otherwise to promote the sale, use or other dealing in this Software without specific, written prior permission. Title to copyright in this Software shall at all times remain with copyright holders.

OpenLDAP is a registered trademark of the OpenLDAP Foundation.

Copyright 1999-2003 The OpenLDAP Foundation, Redwood City, California, USA. All Rights Reserved. Permission to copy and distribute verbatim copies of this document is granted.

## Open SSL

OpenSSL License

Copyright © 1998-2008 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1 Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2 Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3 All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"
- 4 The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact [openssl-core@openssl.org](mailto:openssl-core@openssl.org).
- 5 Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
- 6 Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## PCRE LICENCE

PCRE is a library of functions to support regular expressions whose syntax and semantics are as close as possible to those of the Perl 5 language.

Release 8 of PCRE is distributed under the terms of the "BSD" licence, as specified below. The documentation for PCRE, supplied in the "doc" directory, is distributed under the same terms as the software itself.

The basic library functions are written in C and are freestanding. Also included in the distribution is a set of C++ wrapper functions.

## THE BASIC LIBRARY FUNCTIONS

Written by: Philip Hazel

Email local part: ph10  
Email domain: cam.ac.uk  
University of Cambridge Computing Service,  
Cambridge, England.  
Copyright (c) 1997-2010 University of Cambridge  
All rights reserved.

#### THE C++ WRAPPER FUNCTIONS

Contributed by: Google Inc.  
Copyright (c) 2007-2010, Google Inc.  
All rights reserved.

#### THE "BSD" LICENCE

- Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the University of Cambridge nor the name of Google Inc. nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

End

#### **Perl Artistic License**

Copyright © August 15, 1997

##### Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

## Definitions

"Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified below.

"Copyright Holder" is whoever is named in the copyright or copyrights for the package.

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

- 1 You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.
- 2 You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.
- 3 You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:
- 4 place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.
  1. use the modified Package only within your corporation or organization.
  2. rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.
  3. make other distribution arrangements with the Copyright Holder.
- 5 You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:
  1. distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.
  2. accompany the distribution with the machine-readable source of the Package with your modifications.
  3. give non-standard executables non-standard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.
  4. make other distribution arrangements with the Copyright Holder.

- 6 You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.
- 7 The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package via the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.
- 8 C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.
- 9 Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.
- 10 The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

### **Pexpect**

Copyright © 2010 Noah Spurrier

Credits: Noah Spurrier, Richard Holden, Marco Molteni, Kimberley Burchett, Robert Stone, Hartmut Goebel, Chad Schroeder, Erick Tryzelaar, Dave Kirby, Ids vander Molen, George Todd, Noel Taylor, Nicolas D. Cesar, Alexander Gattin, Geoffrey Marshall, Francisco Lourenco, Glen Mabey, Karthik Gurusamy, Fernando Perez, Corey Minyard, Jon Cohen, Guillaume Chazarain, Andrew Ryan, Nick Craig-Wood, Andrew Stone, Jorgen Grahn (Let me know if I forgot anyone.)

Free, open source, and all that good stuff.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## PHP License

The PHP License, version 3.01

Copyright © 1999 - 2009 The PHP Group. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted provided that the following conditions are met:

- 1 Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2 Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3 The name "PHP" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact [group@php.net](mailto:group@php.net).
- 4 Products derived from this software may not be called "PHP", nor may "PHP" appear in their name, without prior written permission from [group@php.net](mailto:group@php.net). You may indicate that your software works in conjunction with PHP by saying "Foo for PHP" instead of calling it "PHP Foo" or "phpfoo"
- 5 The PHP Group may publish revised and/or new versions of the license from time to time. Each version will be given a distinguishing version number.
  - Once covered code has been published under a particular version of the license, you may always continue to use it under the terms of that version. You may also choose to use such covered code under the terms of any subsequent version of the license published by the PHP Group. No one other than the PHP Group has the right to modify the terms applicable to covered code created under this License.
- 6 Redistributions of any form whatsoever must retain the following acknowledgment:

"This product includes PHP software, freely available from <<http://www.php.net/software/>>".

THIS SOFTWARE IS PROVIDED BY THE PHP DEVELOPMENT TEAM ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE PHP DEVELOPMENT TEAM OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the PHP Group.

The PHP Group can be contacted via Email at [group@php.net](mailto:group@php.net).

For more information on the PHP Group and the PHP project, please see <<http://www.php.net>>.

PHP includes the Zend Engine, freely available at <<http://www.zend.com>>.

## PostgreSQL

This product uses the PostgreSQL Database Management System (formerly known as Postgres, then as Postgres95)

Portions Copyright © 1996-2005, The PostgreSQL Global Development Group

Portions Copyright © 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

## Psqlodbc

The client drivers for Vertica Analytic Database use psqlodbc library, the Postgresql ODBC driver.

License:

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the library and complete text of the GNU Lesser General Public License can be obtained by contacting Vertica support.

## Python 2.7

This is the official license for the Python 2.7 release:

### A. HISTORY OF THE SOFTWARE

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see <http://www.cwi.nl>) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see <http://www.cnri.reston.va.us>) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation, see <http://www.zope.com>). In 2001, the Python Software Foundation (PSF, see <http://www.python.org/psf/>) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see <http://www.opensource.org> for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

Release	Derived from	Year	Owner	GPL-compatible? (1)
0.9.0 thru 1.2		1991-1995	CWI	yes
1.3 thru 1.5.2	1.2	1995-1999	CNRI	yes
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	yes (2)
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	yes
2.1.1	2.1+2.0.1	2001	PSF	yes
2.2	2.1.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2.1	2.2	2002	PSF	yes
2.2.2	2.2.1	2002	PSF	yes
2.2.3	2.2.2	2003	PSF	yes
2.3	2.2.2	2002-2003	PSF	yes
2.3.1	2.3	2002-2003	PSF	yes
2.3.2	2.3.1	2002-2003	PSF	yes
2.3.3	2.3.2	2002-2003	PSF	yes
2.3.4	2.3.3	2004	PSF	yes
2.3.5	2.3.4	2005	PSF	yes
2.4	2.3	2004	PSF	yes
2.4.1	2.4	2005	PSF	yes
2.4.2	2.4.1	2005	PSF	yes
2.4.3	2.4.2	2006	PSF	yes
2.5	2.4	2006	PSF	yes
2.7	2.6	2010	PSF	yes

Footnotes:

- 1 GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.
- 2 According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is "not incompatible" with the GPL.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

## **B. TERMS AND CONDITIONS FOR ACCESSING OR OTHERWISE USING PYTHON**

---

### **PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2**

---

- 
- 1 This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.
  - 2 Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006 Python Software Foundation; All Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee.
  - 3 In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python.
  - 4 PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
  - 5 PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
  - 6 This License Agreement will automatically terminate upon a material breach of its terms and conditions.
  - 7 Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
  - 8 By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.

---

## BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

-----

### BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

- 1 This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").
- 2 Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.
- 3 BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
- 4 BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF
- 5 This License Agreement will automatically terminate upon a material breach of its terms and conditions.
- 6 This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at <http://www.pythonlabs.com/logos.html> may be used according to the permissions granted on that web page.
- 7 By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

## CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

-----

- 1 This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

- 2 Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright (c) 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: <http://hdl.handle.net/1895.22/1013>".
- 3 In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.
- 4 CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
- 5 CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
- 6 This License Agreement will automatically terminate upon a material breach of its terms and conditions.
- 7 This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
- 8 By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

**CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2**

-----

Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

### **Python Dialog**

The Administration Tools part of this product uses Python Dialog, a Python module for doing console-mode user interaction.

Upstream Author:

Peter Astrand <[peter@cendio.se](mailto:peter@cendio.se)>

Robb Shecter <[robb@acm.org](mailto:robb@acm.org)>

Sultanbek Tezadov

Florent Rougon <[flo@via.ecp.fr](mailto:flo@via.ecp.fr)>

Copyright © 2000 Robb Shecter, Sultanbek Tezadov

Copyright © 2002, 2003, 2004 Florent Rougon

License:

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the Python dialog package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at

<http://www.vertica.com/licenses/pythondialog-2.7.tar.bz2>

<http://www.vertica.com/licenses/pythondialog-2.7.tar.bz2>

## Rsync

### COPYRIGHT

-----

Rsync was originally written by Andrew Tridgell and is currently maintained by Wayne Davison. It has been improved by many developers from around the world.

Rsync may be used, modified and redistributed only under the terms of the GNU General Public License, found in the file COPYING in this distribution, or at:

<http://www.fsf.org/licenses/gpl.html>

The following GNU General Public License applies **only to the version of rsync** distributed with Vertica.

### GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

#### Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## GNU GENERAL PUBLIC LICENSE

### TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

#### NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989

Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

## RRDTool Open Source License

Note: rrdtool is a dependency of using the ganglia-web third-party tool. RRDTool allows the graphs displayed by ganglia-web to be produced.

RRDTOOL - Round Robin Database Tool

A tool for fast logging of numerical data graphical display of this data.

Copyright © 1998-2008 Tobias Oetiker

All rights reserved.

## GNU GPL License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

## FLOSS License Exception

(Adapted from <http://www.mysql.com/company/legal/licensing/foss-exception.html>)

I want specified Free/Libre and Open Source Software ("FLOSS") applications to be able to use specified GPL-licensed RRDtool libraries (the "Program") despite the fact that not all FLOSS licenses are compatible with version 2 of the GNU General Public License (the "GPL").

As a special exception to the terms and conditions of version 2.0 of the GPL:

You are free to distribute a Derivative Work that is formed entirely from the Program and one or more works (each, a "FLOSS Work") licensed under one or more of the licenses listed below, as long as:

- 1 You obey the GPL in all respects for the Program and the Derivative Work, except for identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves
- 2 All identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves
  - are distributed subject to one of the FLOSS licenses listed below, and
  - the object code or executable form of those sections are accompanied by the complete corresponding machine-readable source code for those sections on the same medium and under the same FLOSS license as the corresponding object code or executable forms of those sections.
- 3 Any works which are aggregated with the Program or with a Derivative Work on a volume of a storage or distribution medium in accordance with the GPL, can reasonably be considered independent and separate works in themselves which are not derivatives of either the Program, a Derivative Work or a FLOSS Work.

If the above conditions are not met, then the Program may only be copied, modified, distributed or used under the terms and conditions of the GPL.

#### FLOSS License List

License name	Version(s)/Copyright Date
Academic Free License	2.0
Apache Software License	1.0/1.1/2.0
Apple Public Source License	2.0
Artistic license	From Perl 5.8.0
BSD license	"July 22 1999"
Common Public License	1.0
GNU Library or "Lesser" General Public License (LGPL)	2.0/2.1
IBM Public License, Version	1.0
Jabber Open Source License	1.0
MIT License (As listed in file MIT-License.txt)	-
Mozilla Public License (MPL)	1.0/1.1
Open Software License	2.0
OpenSSL license (with original SSLeay license)	"2003" ("1998")
PHP License	3.0
Python license (CNRI Python License)	-
Python Software Foundation License	2.1.1
Sleepycat License	"1999"
W3C License	"2001"
X11 License	"2001"
Zlib/libpng License	-
Zope Public License	2.0/2.1

#### Spread

This product uses software developed by Spread Concepts LLC for use in the Spread toolkit. For more information about Spread see <http://www.spread.org> (<http://www.spread.org>).

Copyright © 1993-2006 Spread Concepts LLC.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1 Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer and request.
- 2 Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer and request in the documentation and/or other materials provided with the distribution.
- 3 All advertising materials (including web pages) mentioning features or use of this software, or software that uses this software, must display the following acknowledgment: "This product uses software developed by Spread Concepts LLC for use in the Spread toolkit. For more information about Spread see <http://www.spread.org>"

- 4 The names "Spread" or "Spread toolkit" must not be used to endorse or promote products derived from this software without prior written permission.
- 5 Redistributions of any form whatsoever must retain the following acknowledgment:
- 6 "This product uses software developed by Spread Concepts LLC for use in the Spread toolkit. For more information about Spread, see <http://www.spread.org>"
- 7 This license shall be governed by and construed and enforced in accordance with the laws of the State of Maryland, without reference to its conflicts of law provisions. The exclusive jurisdiction and venue for all legal actions relating to this license shall be in courts of competent subject matter jurisdiction located in the State of Maryland.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, SPREAD IS PROVIDED UNDER THIS LICENSE ON AN AS IS BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT SPREAD IS FREE OF DEFECTS, MERCHANTABILITY, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. ALL WARRANTIES ARE DISCLAIMED AND THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE CODE IS WITH YOU. SHOULD ANY CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE COPYRIGHT HOLDER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY OTHER CONTRIBUTOR BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES FOR LOSS OF PROFITS, REVENUE, OR FOR LOSS OF INFORMATION OR ANY OTHER LOSS.

YOU EXPRESSLY AGREE TO FOREVER INDEMNIFY, DEFEND AND HOLD HARMLESS THE COPYRIGHT HOLDERS AND CONTRIBUTORS OF SPREAD AGAINST ALL CLAIMS, DEMANDS, SUITS OR OTHER ACTIONS ARISING DIRECTLY OR INDIRECTLY FROM YOUR ACCEPTANCE AND USE OF SPREAD.

Although NOT REQUIRED, we at Spread Concepts would appreciate it if active users of Spread put a link on their web site to Spread's web site when possible. We also encourage users to let us know who they are, how they are using Spread, and any comments they have through either e-mail ([spread@spread.org](mailto:spread@spread.org)) or our web site at (<http://www.spread.org/comments>).

## **SNMP**

Various copyrights apply to this package, listed in various separate parts below. Please make sure that you read all the parts. Up until 2001, the project was based at UC Davis, and the first part covers all code written during this time. From 2001 onwards, the project has been based at SourceForge, and Networks Associates Technology, Inc hold the copyright on behalf of the wider Net-SNMP community, covering all derivative work done since then. An additional copyright section has been added as Part 3 below also under a BSD license for the work contributed by Cambridge Broadband Ltd. to the project since 2001. An additional copyright section has been added as Part 4 below also under a BSD license for the work contributed by Sun Microsystems, Inc. to the project since 2003.

Code has been contributed to this project by many people over the years it has been in development, and a full list of contributors can be found in the README file under the THANKS section.

**Part 1: CMU/UCD copyright notice: (BSD like)**

Copyright © 1989, 1991, 1992 by Carnegie Mellon University  
Derivative Work - 1996, 1998-2000  
Copyright © 1996, 1998-2000 The Regents of the University of California  
All Rights Reserved

Permission to use, copy, modify and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of CMU and The Regents of the University of California not be used in advertising or publicity pertaining to distribution of the software without specific written permission.

CMU AND THE REGENTS OF THE UNIVERSITY OF CALIFORNIA DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL CMU OR THE REGENTS OF THE UNIVERSITY OF CALIFORNIA BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM THE LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

**Part 2: Networks Associates Technology, Inc copyright notice (BSD)**

Copyright © 2001-2003, Networks Associates Technology, Inc  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Networks Associates Technology, Inc nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**Part 3: Cambridge Broadband Ltd. copyright notice (BSD)**

Portions of this code are copyright (c) 2001-2003, Cambridge Broadband Ltd.  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The name of Cambridge Broadband Ltd. may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**Part 4: Sun Microsystems, Inc. copyright notice (BSD)**

Copyright © 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,  
California 95054, U.S.A. All rights reserved.

Use is subject to license terms below.

This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Sun Microsystems, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**Part 5: Sparta, Inc copyright notice (BSD)**

Copyright © 2003-2006, Sparta, Inc  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Sparta, Inc nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**Part 6: Cisco/BUPTNIC copyright notice (BSD)**

Copyright © 2004, Cisco, Inc and Information Network Center of Beijing University of Posts and Telecommunications.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Cisco, Inc, Beijing University of Posts and Telecommunications, nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**Part 7: Fabasoft R&D Software GmbH & Co KG copyright notice (BSD)**

Copyright © Fabasoft R&D Software GmbH & Co KG, 2003

[oss@fabasoft.com](mailto:oss@fabasoft.com)

Author: Bernhard Penz

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The name of Fabasoft R&D Software GmbH & Co KG or any of its subsidiaries, brand or product names may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE

LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### **Tecla Command-line Editing**

Copyright © 2000 by Martin C. Shepherd.

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

### **Webmin Open Source License**

Copyright © Jamie Cameron

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1** Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2** Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3** Neither the name of the developer nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE DEVELOPER ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE DEVELOPER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### **xerces**

NOTICE file corresponding to section 4(d) of the Apache License, Version 2.0, in this case for the Apache Xerces distribution.

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

Portions of this software were originally based on the following:

Software copyright © 1999, IBM Corporation., <http://www.ibm.com>.

### **zlib**

This is used by the project to load zipped files directly by COPY command. [www.zlib.net/](http://www.zlib.net/)

zlib.h -- interface of the 'zlib' general purpose compression library version 1.2.3, July 18th, 2005

Copyright © 1995-2005 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

- 1** The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
- 2** Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
- 3** This notice may not be removed or altered from any source distribution.

Jean-loup Gailly [jloup@gzip.org](mailto:jloup@gzip.org)

Mark Adler [madler@alumni.caltech.edu](mailto:madler@alumni.caltech.edu)