IBM Db2 V11.5

Compatibility Features
2020-08-21

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Contents

Tables

Chapter 1. Compatibility features

You might have an application that was written for use with a relational database that is not a Db2® database. Compatibility features enable such applications to use Db2 databases without having to be rewritten.

Compatibility features for Oracle

Db2 provides features that enable applications that were written for an Oracle database to use a Db2 database without having to be rewritten.

Attention: Setting the "DB2_COMPATIBILITY_VECTOR registry variable" on page 29 is required to enable some Oracle compatibility features.

Data types provided for Oracle compatability

DATE data type based on TIMESTAMP(0)
The DATE data type supports applications that use the Oracle DATE data type and expect that the DATE values include time information (for example, '2009-04-01-09.43.05').

Enablement

You enable DATE as TIMESTAMP(0) support at the database level, before creating the database where you require the support. To enable the support, set the **DB2_COMPATIBILITY_VECTOR** registry variable to hexadecimal value 0x40 (bit position 7), and then stop and restart the instance to have the new setting take effect.

```
db2set DB2_COMPATIBILITY_VECTOR=40
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Oracle applications, the recommended setting for the DB2_COMPATIBILITY_VECTOR is ORA, which sets all of the compatibility bits.

After you create a database with DATE as TIMESTAMP(0) support enabled, the **date_compat** database configuration parameter is set to `ON`.

If you create a database with DATE as TIMESTAMP(0) support enabled, you cannot disable that support for that database, even if you reset the **DB2_COMPATIBILITY_VECTOR** registry variable. Similarly, if you create a database with DATE as TIMESTAMP(0) support disabled, you cannot enable that support for that database later, even by setting the **DB2_COMPATIBILITY_VECTOR** registry variable.

Effects

The **date_compat** database configuration parameter indicates whether the DATE compatibility semantics associated with the TIMESTAMP(0) data type are applied to the connected database. The effects of setting **date_compat** to `ON` are as follows.

When the DATE data type is explicitly encountered in SQL statements, it is implicitly mapped to TIMESTAMP(0) in most cases. An exception is the specification of SQL DATE in the *xml-index-specification* clause of a CREATE INDEX statement. As a result of the implicit mapping, messages refer to the TIMESTAMP data type instead of DATE, and any operations that describe data types for columns or routines return TIMESTAMP instead of DATE.

Datetime literal support is changed as follows:

- The value of an explicit DATE literal is a TIMESTAMP(0) value in which the time portion is all zeros. For example, `DATE '2008-04-28'` represents the timestamp value '2008-04-28-00.00.00'.

- The database manager supports two additional formats for the string representation of a date, which correspond to 'DD-MON-YYYY' and 'DD-MON-RR'. Only English abbreviations of the month are supported. For example, '28-APR-2008' or '28-APR-08' can be used as string representations of a date, which represents the TIMESTAMP(0) value '2008-04-28-00.00.00'.

  Starting from Version 9.7 Fix Pack 6, the database manager also supports the following formats for the string representation of a date in English only:

  - 'DDMONYYYY' or 'DDMONRR'
  - 'DD-MONYYYY' or 'DD-MONRR'
  - 'DDMON-YYYY' or 'DDMON-RR'

  For example, the following strings all represent the TIMESTAMP(0) value '2008-04-28-00.00.00':

  - '28APR2008' or '28APR08'
  - '28-APR2008' or '28-APR08'
  - '28APR-2008' or '28APR-08'

  For a description of the format elements, see TIMESTAMP_FORMAT scalar function (see *SQL Reference Volume 1*).

The CURRENT_DATE (also known as CURRENT DATE) special register returns a TIMESTAMP(0) value that is the same as the CURRENT_TIMESTAMP(0) value.

When you add a numeric value to a TIMESTAMP value or subtract a numeric value from a TIMESTAMP value, it is assumed that the numeric value represents a number of days. The numeric value can have any numeric data type, and any fractional value is considered to be a fractional portion of a day. For example, `TIMESTAMP '2008-03-28 12:00:00' + 1.3` adds 1 day, 7 hours, and 12 minutes to the TIMESTAMP value, resulting in '2008-03-29 19:12:00'. If you are using expressions for partial days, such as 1/24 (1 hour) or 1/24/60 (1 minute), ensure that the **`number_compat`** database configuration parameter is set to `ON` so that the division is performed using DECFLOAT arithmetic.

The results of some functions change:

- If you pass a string argument to the ADD_YEARS, ADD_MONTHS, or ADD_DAYS scalar function, it returns a TIMESTAMP(0) value.
- The DATE scalar function returns a TIMESTAMP(0) value for all input types.
- If you pass a string argument to the FIRST_DAY or LAST_DAY scalar function, it returns a TIMESTAMP(0) value.
- If you pass a DATE argument to the ADD_YEARS, ADD_MONTHS, ADD_DAYS, ADD_HOURS, ADD_MINUTES, ADD_SECONDS, LAST_DAY, NEXT_DAY, ROUND, or TRUNCATE scalar function, the function returns a TIMESTAMP(0) value.
- The adding of one date value to another returns TIMESTAMP(0) value.
- Subtracting one timestamp value from another returns DECFLOAT(34), which represents the difference as a number of days, with exception to expressions that are used to define the following scalar functions:

  - YEARS_BETWEEN
  - YMD_BETWEEN
  - AGE

  The expressions in these scalar functions retain the existing semantic of returning a timestamp duration.
- Subtracting one date value from another returns DECFLOAT(34), which represents a number of days.
- If you specify the fractional second format (FF) for the `TO_DATE` or `TO_TIMESTAMP` function, the fractional second format is equivalent to specifying 10^{-9} precision value (FF9).
- The second parameter in the TIMESTAMPDIFF scalar function does not represent a timestamp duration. Rather it represents the difference between two timestamps as a number of days. The

returned estimate may vary by a number of days. For example, if the number of months (interval 64) is requested for the difference between '2010-03-31-00.00.00.000000' and '2010-03-01-00.00.00.000000', the result is 1. This is because the difference between the timestamps is 30 days, and the assumption of 30 days in a month applies. The following table shows how the returned value is determined for each interval.

Table 1. TIMESTAMPDIFF computations

Result interval	Computation using the difference between two timestamps as a number of days
Years	integer value of (days/365)
Quarters	integer value of (days/90)
Months	integer value of (days/30)
Weeks	integer value of (days/7)
Days	integer value of days
Hours	integer value of (days*24)
Minutes (the absolute value of the number of days must not exceed 1491308.0888888888888882)	integer value of (days*24*60)
Seconds (the absolute value of the number of days must be less than 24855.1348148148148148)	integer value of (days*24*60*60)
Microseconds (the absolute value of the number of days must be less than 0.02485513481481481)	integer value of (days*24*60*60*1000000)

If you use the import or load utility to input data into a DATE column, you must use the timestampformat file type modifier instead of the dateformat file type modifier.

NUMBER data type

The NUMBER data type supports applications that use the Oracle NUMBER data type.

Enablement

You enable NUMBER support at the database level, before creating the database where you require the support. To enable the support, set the **DB2_COMPATIBILITY_VECTOR** registry variable to hexadecimal value 0x10 (bit position 5), and then stop and restart the instance to have the new setting take effect.

```
db2set DB2_COMPATIBILITY_VECTOR=10
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Oracle applications, the recommended setting for the DB2_COMPATIBILITY_VECTOR is ORA, which sets all of the compatibility bits.

When you create a database with NUMBER support enabled, the **number_compat** database configuration parameter is set to ON.

If you create a database with NUMBER support enabled, you cannot disable NUMBER support for that database, even if you reset the **DB2_COMPATIBILITY_VECTOR** registry variable. Similarly, if you create a database with NUMBER support disabled, you cannot enable NUMBER support for that database later, even by setting the **DB2_COMPATIBILITY_VECTOR** registry variable.

Effects

The effects of setting the **number_compat** database configuration parameter to ON are as follows.

When the NUMBER data type is explicitly encountered in SQL statements, the data type is implicitly mapped as follows:

- If you specify NUMBER without precision and scale attributes, it is mapped to DECFLOAT(16).
- If you specify NUMBER(*p*), it is mapped to DECIMAL(*p*).
- If you specify NUMBER(*p*,*s*), it is mapped to DECIMAL(*p*,*s*).

The maximum supported precision is 31, and the scale must be a positive value that is no greater than the precision. Also, a result of the implicit mapping, messages refer to data types DECFLOAT and DECIMAL instead of NUMBER. In addition, any operations that describe data types for columns or routines return either DECIMAL or DECFLOAT instead of NUMBER.

Tip: The DECFLOAT(16) data type provides a lower maximum precision than that of the Oracle NUMBER data type. If you need more than 16 digits of precision for storing numbers in columns, then explicitly define those columns as DECFLOAT(34).

Numeric literal support is unchanged: the rules for integer, decimal, and floating-point constants continue to apply. These rules limit decimal literals to 31 digits and floating-point literals to the range of binary double-precision floating-point values. If necessary, you can use a string-to-DECFLOAT(34) cast, using the CAST specification or the DECFLOAT function, for values beyond the range of DECIMAL or DOUBLE up to the range of DECFLOAT(34). There is currently no support for a numeric literal that ends in either D, representing 64-bit binary floating-point values, or F, representing 32-bit binary floating-point values. A numeric literal that includes an E has the data type of DOUBLE, which you can cast to REAL using the CAST specification or the cast function REAL

If you cast NUMBER data values to character strings, using either the CAST specification or the VARCHAR or CHAR scalar function, all leading zeros are stripped from the result.

The default data type that is used for a sequence value in the CREATE SEQUENCE statement is DECIMAL(27) instead of INTEGER.

All arithmetic operations and arithmetic or mathematical functions involving DECIMAL or DECFLOAT data types are effectively performed using decimal floating-point arithmetic and return a value with a data type of DECFLOAT(34). This type of performance also applies to arithmetic operations where both operands have a DECIMAL or DECFLOAT(16) data type, which differs from the description of decimal arithmetic in the "Expressions with arithmetic operators" section of Expressions (see *SQL Reference Volume 1*). Additionally, all division operations involving only integer data types (SMALLINT, INTEGER, or BIGINT) are effectively performed using decimal floating-point arithmetic. These operations return a value with a data type of DECFLOAT(34) instead of an integer data type. Division by zero with integer operands returns infinity and a warning instead of an error.

In some cases function resolution is also changed, such that an argument of data type DECIMAL is considered to be a DECFLOAT value during the resolution process. Also functions with arguments that correspond to the NUMBER(*p*[,*s*]) data type are effectively treated as if the argument data types were NUMBER. However, this change in function resolution does not apply to the set of functions that have a variable number of arguments and base their result data types on the set of data types of the arguments. The functions included in this set are as follows:

- COALESCE
- DECODE
- GREATEST
- LEAST
- MAX (scalar)
- MIN (scalar)
- NVL
- VALUE

In addition to the general changes to numeric functions, a special consideration also applies to the MOD function; if the second argument in the MOD function is zero, then the function returns the value of the first argument.

For more information on how the rules for result data types are extended to make DECFLOAT(34) the result data type if the precision of a DECIMAL result data type would have exceeded, see rules for result data types (see *SQL Reference Volume 1*). These rules also apply to the following items:

- Corresponding columns in set operations: UNION, EXCEPT(MINUS), and INTERSECT
- Expression values in the IN list of an IN predicate
- Corresponding expressions of a multiple row VALUES clause

The rounding mode that is used for assignments and casts depends on the data types that are involved. In some cases, truncation is used. In cases where the target is a binary floating-point (REAL or DOUBLE) value, round-half-even is used, as usual. In other cases, usually involving a DECIMAL or DECFLOAT value, the rounding is based on the value of the **`decflt_rounding`** database configuration parameter. The value of this parameter defaults to round-half-even, but you can set it to round-half-up to match the Oracle rounding mode. The following table summarizes the rounding that is used for various numeric assignments and casts.

Table 2. Rounding for numeric assignments and casts

Source data type	**Target data type**			
	Integer types	**DECIMAL**	**DECFLOAT**	**REAL/DOUBLE**
Integer types	not applicable	not applicable	decflt_rounding	round_half_even
DECIMAL	decflt_rounding	decflt_rounding	decflt_rounding	round_half_even
DECFLOAT	decflt_rounding	decflt_rounding	decflt_rounding	round_half_even
REAL/DOUBLE	truncate	decflt_rounding	decflt_rounding	round_half_even
String (cast only)	not applicable	decflt_rounding	decflt_rounding	round_half_even

The Db2 decimal floating-point values are based on the IEEE 754R standard. Retrieval of DECFLOAT data and casting of DECFLOAT data to character strings removes any trailing zeros after the decimal point.

Starting in Db2 Version 10.5 Fix Pack 4, in a database with NUMBER support enabled, the built-in functions STDDEV, VAR and VARIANCE with integer input returns DECFLOAT instead of DOUBLE. Any view column or materialized query table (MQT) column with a result type that depends on this function continues to return the old result type until the view or MQT is regenerated or recreated. In the case of an MQT, any queries that previously routed to the MQT will not longer do so until the MQT is recreated.

Client-server compatibility

Client applications working with a Db2 database server that you enable for NUMBER data type support never receive a NUMBER data type from the server. Any column or expression that would report NUMBER from an Oracle server report either DECIMAL or DECFLOAT from a Db2 database server.

Because an Oracle environment expects the rounding mode to be round-half-up, it is important that the client rounding mode match the server rounding mode. This means that the `db2cli.ini` file setting must match the value of the **`decflt_rounding`** database configuration parameter. To most closely match the Oracle rounding mode, you should specify `ROUND_HALF_UP` for the database configuration parameter.

Restrictions

NUMBER data type support has the following restrictions:

- There is no support for the following items:
  - A precision attribute greater than 31
  - A precision attribute of asterisk (*)
  - A scale attribute that exceeds the precision attribute

– A negative scale attribute

There is no corresponding DECIMAL precision and scale support for NUMBER data type specifications.

- You cannot invoke the trigonometric functions or the DIGITS scalar function with arguments of data type NUMBER without a precision (DECFLOAT).
- You cannot create a distinct type with the name NUMBER.

VARCHAR2 and NVARCHAR2 data types
The VARCHAR2 and NVARCHAR2 data types support applications that use the Oracle VARCHAR2 and NVARCHAR2 data types.

Enablement

You enable VARCHAR2 and NVARCHAR2 (subsequently jointly referred to as VARCHAR2) support at the database level, before creating the database where you require support. To enable the support, set the **DB2_COMPATIBILITY_VECTOR** registry variable to hexadecimal value 0x20 (bit position 6), and then stop and restart the instance to have the new setting take effect. Create your Db2 database by issuing the **CREATE DATABASE** command. By default, databases are created as Unicode databases. For example, to create a database that is named DB, issue the following command:

```
db2set DB2_COMPATIBILITY_VECTOR=20
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Oracle applications, the recommended setting for the **DB2_COMPATIBILITY_VECTOR** is **ORA**, which sets all of the compatibility bits.

When you create a database with VARCHAR2 support enabled, the **varchar2_compat** database configuration parameter is set to ON.

If you create a database with VARCHAR2 support enabled, you cannot disable VARCHAR2 support for that database, even if you reset the **DB2_COMPATIBILITY_VECTOR** registry variable. Similarly, if you create a database with VARCHAR2 support disabled, you cannot enable VARCHAR2 support for that database later, even by setting the **DB2_COMPATIBILITY_VECTOR** registry variable.

To use the NVARCHAR2 data type, a database must be a Unicode database.

Effects

The effects of setting the **varchar2_compat** database configuration parameter to ON are as follows.

When the VARCHAR2 data type is explicitly encountered in SQL statements, it is implicitly mapped to the VARCHAR data type. The maximum length for VARCHAR2 is 32672 BYTE or 8168 CHAR which is the same as the maximum length for VARCHAR of 32672 OCTETS or 8168 CODEUNITS32. Similarly, when the NVARCHAR2 data type is explicitly encountered in SQL statements, it is implicitly mapped following the same rules as the NVARCHAR data type.

Character string literals can have a data type of CHAR or VARCHAR, depending on the length and the string units of the environment. Character string literals up to the maximum length of a CHAR in the string units of the environment (255 OCTETS or 63 CODEUNITS32) have a data type of CHAR. Character string literals longer than the maximum length of a CHAR in the string units of the environment have a data type of VARCHAR.

Comparisons involving varying-length string types use non-padded comparison semantics, and comparisons with only fixed-length string types continue to use blank-padded comparison semantics, with two exceptions:

- Comparisons involving string column information from catalog views always use the IDENTITY collation with blank-padded comparison semantics, regardless of the database collation.
- String comparisons involving a data type with the FOR BIT DATA attribute always use the IDENTITY collation with blank-padded comparison semantics.

The rules for result data types are modified as follows:

Table 3. Modified rules for result data types that involve character strings		
If one operand is...	**And the other operand is...**	**The data type of the result is...**
CHAR(x)	CHAR(x)	CHAR(x)
CHAR(x)	CHAR(y)	VARCHAR(z), where x != y and z = max(x,y)
GRAPHIC(x)	GRAPHIC(x)	GRAPHIC(x)
GRAPHIC(x)	GRAPHIC(y)	VARGRAPHIC(z), where x != y and z = max(x,y)
GRAPHIC(x)	CHAR(y)	VARGRAPHIC(z), where z = max(x,y)

If the result type for the IN list of an IN predicate would resolve to a fixed-length string data type and the left operand of the IN predicate is a varying-length string data type, the IN list expressions are treated as having a varying-length string data type.

Character and binary string values (other than LOB values) with a length of zero are generally treated as null values. An assignment or cast of an empty string value to CHAR, NCHAR, VARCHAR, NVARCHAR, BINARY, or VARBINARY produces a null value.

Functions that return character or binary string arguments, or that are based on parameters with character or binary string data types, also treat empty string CHAR, NCHAR, VARCHAR, NVARCHAR, BINARY, or VARBINARY values as null values. Therefore, the result of some built-in functions and casts that return character or graphic string can be null even when all of the arguments are not null. Special considerations apply for some functions when the **`varchar2_compat`** database configuration parameter is set to ON, as follows:

- CONCAT function and the concatenation operator. A null or empty string value is ignored in the concatenated result. The result type of the concatenation is shown in the following table.

Table 4. Data Type and lengths of concatenated operands		
Operands	**Combined length attributes[1]**	**Result[1]**
CHAR(A) CHAR(B)	<=*S*	CHAR(A+B)
CHAR(A) CHAR(B)	>*S*	VARCHAR(A+B)
CHAR(A) VARCHAR(B)	-	VARCHAR(MIN(A+B, *W*))
VARCHAR(A) VARCHAR(B)	-	VARCHAR(MIN(A+B, *W*))
CLOB(A) CHAR(B)	-	CLOB(MIN(A+B, *X*))
CLOB(A) VARCHAR(B)	-	CLOB(MIN(A+B *X*))
CLOB(A) CLOB(B)		CLOB(MIN(A+B, *X*))
GRAPHIC(A) GRAPHIC(B)	<=*T*	GRAPHIC(A+B)
GRAPHIC(A) GRAPHIC(B)	>*T*	VARGRAPHIC(A+B)
GRAPHIC(A) VARGRAPHIC(B)	-	VARGRAPHIC(MIN(A+B, *Y*))
VARGRAPHIC(A) VARGRAPHIC(B)	-	VARGRAPHIC(MIN(A+B, *Y*))
DBCLOB(A) CHAR(B)	-	DBCLOB(MIN(A+B, *Z*))
DBCLOB(A) VARCHAR(B)	-	DBCLOB(MIN(A+B, *Z*))

Table 4. Data Type and lengths of concatenated operands (continued)		
Operands	**Combined length attributes**[1]	**Result**[1]
DBCLOB(A) DCLOB(B)		DBCLOB(MIN(A+B, *Z*))
1. See the following table for values for italicized variables.		

Table 5. The italicized variables in the previous table have the following values		
Variable	**If no operand has string units of CHAR (or CODEUNITS32)**	**If either operand has string units of CHAR (or CODEUNITS32**
S	255	63
T	127	63
W	32672	8168
X	2G	536870911
Y	16336	8168
Z	1G	536870911

- DECODE function. If the first result expression is an untyped null it is assumed to be VARCHAR(0). If the first result expression is CHAR or GRAPHIC, it is promoted to VARCHAR or VARGRAPHIC.
- GREATEST function. If the first expression is CHAR, BINARY, or GRAPHIC, it is promoted to VARCHAR, VARBINARY or VARGRAPHIC.
- INSERT function. A null value or empty string as the fourth argument results in deletion of the number of bytes indicated by the third argument, beginning at the byte position indicated by the second argument from the first argument.
- LEAST function. If the first expression is CHAR, BINARY, or GRAPHIC, it is promoted to VARCHAR, VARBINARY or VARGRAPHIC.
- LENGTH function. The value returned by the LENGTH function is the number of bytes in the character string. An empty string value returns the null value.
- NVL function. If the first expression is CHAR, BINARY, or GRAPHIC, it is promoted to VARCHAR, VARBINARY, or VARGRAPHIC.
- NVL2 function. If the result expression is an untyped null it is assumed to be VARCHAR(0). If the result expression is CHAR, BINARY, or GRAPHIC, it is promoted to VARCHAR, VARBINARY, or VARGRAPHIC.
- REGEXP_REPLACE function. A null value or empty string as the third argument is treated as an empty string. Nothing replaces the string that is removed from the source string that is based on the matched string that is determined by the other arguments.
- REPLACE function. If all of the argument values have a data type of CHAR, VARCHAR, , BINARY, VARBINARY, GRAPHIC, or VARGRAPHIC, then:
  - A null value or empty string as the second argument is treated as an empty string, and consequently the first argument is returned as the result
  - A null value or empty string as the third argument is treated as an empty string, and nothing replaces the string that is removed from the source string by the second argument.

  If any argument value has a data type of CLOB or BLOB and any argument is the null value, the result is the null value. All three arguments of the REPLACE function must be specified.
- SUBSTR function. References to SUBSTR are replaced with the following function invocation based on the first argument:

- SUBSTRB when the first argument is a binary string or character string with string units defined as OCTETS.
- SUBSTR2 when the first argument is a graphic string with string units defined as CODEUNITS16.
- SUBSTR4 when the first argument is a character string or graphic string with string units defined as CODEUNITS32.

- TO_CHAR function. If two arguments are specified and the first argument is a string, the first argument is cast to a decimal floating point. This behavior applies to Version 10.5 Fix Pack 3 and later fix packs.
- TO_NCHAR function. If two arguments are specified and the first argument is a string, the first argument is cast to a decimal floating point. This behavior applies to Version 10.5 Fix Pack 3 and later fix packs.
- TRANSLATE function. The *from-string-exp* is the second argument, and the *to-string-exp* is the third argument. If the *to-string-exp* is shorter than the *from-string-exp*, the extra characters in the *from-string-exp* that are found in the *char-string-exp* (the first argument) are removed; that is, the default *pad-char* argument is effectively an empty string, unless a different pad character is specified in the fourth argument.
- TRIM function. If the trim character argument of a TRIM function invocation is a null value or an empty string, the function returns a null value.
- VARCHAR_FORMAT function. If two arguments are specified and the first argument is a string, the first argument is cast to a decimal floating point. This behavior applies to Version 10.5 Fix Pack 3 and later fix packs.

In the ALTER TABLE statement or the CREATE TABLE statement, when a DEFAULT clause is specified without an explicit value for a column defined with the VARCHAR or the VARGRAPHIC data type, the default value is a blank character. If the column is defined with the VARBINARY data type, the default value is a hexadecimal zero.

Empty strings in catalog view columns are converted to a blank character when the database configuration parameter **`varchar2_compat`** is set to ON. For example:

- SYSCAT.DATAPARTITIONS.STATUS has a single blank character when the data partition is visible.
- SYSCAT.PACKAGES.PKGVERSION has a single blank character when the package version has not been explicitly set.
- SYSCAT.ROUTINES.COMPILE_OPTIONS has a null value when compile options have not been set.

If SQL statements use parameter markers, a data type conversion that affects VARCHAR2 usage can occur. For example, if the input value is a VARCHAR of length zero and it is converted to a LOB, the result will be a null value. However, if the input value is a LOB of length zero and it is converted to a LOB, the result will be a LOB of length zero. The data type of the input value can be affected by deferred prepare.

When defining a data type, CHAR can be used as a synonym for CODEUNITS32, and BYTE can be used as a synonym for OCTETS.

Restrictions

The VARCHAR2 data type and associated character string processing support have the following restrictions:

- The VARCHAR2 length attribute qualifier CHAR is accepted only in a Unicode database as a synonym for CODEUNITS32.
- The LONG VARCHAR and LONG VARGRAPHIC data types are not supported (but are not explicitly blocked) when the **`varchar2_compat`** database configuration parameter is set to ON.
- Without specifying the maximum length for a VARCHAR2 parameter, the default is 4000 bytes.
- NLSCHAR collation, used for sorting characters in a TIS620-1 (code page 874) Thai database, is not supported when setting **`DB2_COMPATIBILITY_VECTOR=ORA`**.

Implicit casting for character and graphic constants

Implicit casting (or weak typing) is an alternative way to parse character or graphic constants for applications that expect these constants to be assigned the data types CHAR or GRAPHIC.

Enablement

To enable implicit casting for character and graphic constants, set the **DB2_COMPATIBILITY_VECTOR** registry variable to hexadecimal value `0x100` (bit position 9), then stop and restart the instance:

```
db2set DB2_COMPATIBILITY_VECTOR=100
db2stop
db2start
```

To activate all the compatibility features for Oracle applications, set the **DB2_COMPATIBILITY_VECTOR** to `ORA`, then stop and restart the instance:

```
db2set DB2_COMPATIBILITY_VECTOR=ORA
db2stop
db2start
```

If implicit casting for character and graphic constants is disabled, an application that uses weak typing in its SQL will fail to compile for the Db2 product. If enabled, strings and numbers can be compared, assigned, and operated on in a very flexible fashion. Because this data type assignment affects the result types of some SQL statements, it is strongly recommended that this registry variable setting not be toggled for a particular database.

Effects

When implicit casting for character and graphic constants is enabled, the data type of a character or graphic constant depends on the setting of the environment string unit and on the length of the constant:

String Constant Type	Environment String Unit	Size	Data Type
Character	CODEUNITS16 or OCTETS	≤ 255 bytes	CHAR
	CODEUNITS32	≤ 63 code units	
	CODEUNITS16 or OCTETS	> 255 bytes	VARCHAR
	CODEUNITS32	> 63 code units	
Graphic	CODEUNITS16 or OCTETS	≤ 254 bytes	GRAPHIC
	CODEUNITS32	≤ 63 code units	
	CODEUNITS16 or OCTETS	> 254 bytes	VARGRAPHIC
	CODEUNITS32	> 63 code units	

SQL data-access-level enforcement

The degree to which a routine (stored procedure or user-defined function) can execute SQL statements is determined by its SQL-access-level.

There are four SQL data-access-levels:

- NO SQL
- CONTAINS SQL
- READS SQL DATA

• MODIFIES SQL DATA

By default, SQL PL and PL/SQL routines enforce data-access levels at compile time. If a routine contains an SQL statement that requires a data-access level that exceeds that of the routine, an error is returned when you create the routine. Similarly, if a routine invokes another routine whose data-access level exceeds that of the calling routine, an error is returned when you create the first routine. Additionally, if you define a compiled user-defined function as MODIFIES SQL DATA, you can use it only as the sole element on the right side of an assignment statement within a compound SQL (compiled) statement. This check is also performed when you compile the statement.

Starting with Version 9.7 Fix Pack 3, you can have SQL PL and PL/SQL routines enforce data-access levels at run time instead of at compile time by setting the **DB2_COMPATIBILITY_VECTOR** registry variable. To enable the support, set the registry variable to hexadecimal value 0x10000 (bit position 17), and then stop and restart the instance to have the new setting take effect.

```
db2set DB2_COMPATIBILITY_VECTOR=10000
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Oracle applications, the recommended setting for the DB2_COMPATIBILITY_VECTOR is ORA, which sets all of the compatibility bits.

The enforcement is performed at run-time at the statement level. An error is returned when a statement that exceeds the current SQL data access level is performed. If a routine invokes another routine defined with a more restrictive SQL data-access level, the called routine inherits the data-access level of its parent. Additionally, if you define a compiled user-defined function as MODIFIES SQL DATA and it is not the sole element on the right side of an assignment statement within a compound SQL (compiled) statement, an error is returned only if the function issues an SQL statement that modifies SQL data.

In addition, starting with Version 9.7 Fix Pack 6, COMMIT and ROLLBACK statements are allowed in a compiled PL/SQL user-defined function and a compiled language SQL user-defined function that has been defined with the MODIFIES SQL DATA clause in a CREATE FUNCTION statement.

Outer join operator

Queries can use the outer join operator (+) as alternative syntax within predicates of the WHERE clause.

A join is the process of combining data from two or more tables based on some common domain of information. Rows from one table are paired with rows from another table when information in the corresponding rows match on the basis of the joining criterion. An outer join returns all rows that satisfy the join condition and also returns some or all of the rows from one or both tables for which no rows satisfy the join condition. You should use the outer join syntax of RIGHT OUTER JOIN, LEFT OUTER JOIN, or FULL OUTER JOIN wherever possible. You should use the outer join operator only when enabling applications from database products other than the Db2 product to run on a Db2 database system.

Enablement

You enable outer join operator support by setting the DB2_COMPATIBILITY_VECTOR registry variable to hexadecimal value 0x04 (bit position 3), and then stop and restart the instance to have the new setting take effect.

```
db2set DB2_COMPATIBILITY_VECTOR=04
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Oracle applications, the recommended setting for the DB2_COMPATIBILITY_VECTOR is ORA, which sets all of the compatibility bits.

Examples

You apply the outer join operator (+) in parentheses following a column name within predicates that refer to columns from two tables, as shown in the following examples:

- The following query performs a left outer join of tables T1 and T2. Include both tables in the FROM clause, separated by a comma. Apply the outer join operator to all columns of T2 in predicates that also reference T1.

```
SELECT * FROM T1
  LEFT OUTER JOIN T2 ON T1.PK1 = T2.FK1
    AND T1.PK2 = T2.FK2
```

  The previous query is equivalent to the following one, which uses the outer join operator:

```
SELECT * FROM T1, T2
  WHERE T1.PK1 = T2.FK1(+)
    AND T1.PK2 = T2.FK2(+)
```

- The following query performs a right outer join of tables T1 and T2. Include both tables in the FROM clause, separated by a comma, and apply the outer join operator to all columns of T1 in predicates that also reference T2.

```
SELECT * FROM T1
  RIGHT OUTER JOIN T2 ON T1.FK1 = T2.PK1
    AND T1.FK2 = T2.PK2
```

  The previous query is equivalent to the following one, which uses the outer join operator:

```
SELECT * FROM T1, T2
  WHERE T1.FK1(+) = T2.PK1
    AND T1.FK2(+) = T2.PK2
```

A table that has columns marked with the outer join operator is sometimes referred to as a *NULL-producer*.

A set of predicates that are separated by AND operators is known as an *AND-factor*. If there are no AND operators in a WHERE clause, the set of predicates in the WHERE clause is considered to be the only AND-factor.

Rules

The following rules apply to the outer join operator:

- Predicates
  - The WHERE predicate is considered on a granularity of ANDed Boolean factors.
  - Local predicates such as `T1.A(+) = 5` can exist, but they are executed with the join. A local predicate without (+) is executed after the join.
- Boolean
  - Each Boolean term can refer to at most two tables, for example, `T1.C11 + T2.C21 = T3.C3(+)` is not allowed.
  - Correlation for outer join Boolean terms is not allowed.
- Outer join operator
  - You cannot specify the outer join operator in the same subselect as the explicit JOIN syntax
  - You can specify the outer join operator only in the WHERE clause on columns that are associated with tables that you specify in the FROM clause of the same subselect.
  - You cannot apply the outer join operator to an entire expression. Within an AND-factor, each column reference from the same table must be followed by the outer join operator, for example, `T1.COL1 (+) - T1.COL2 (+) = T2.COL1`.
  - You can specify the outer join operator only in the WHERE clause on columns that are associated with tables that you specify in the FROM clause of the same subselect.
- NULL-producer

- Each table can be the NULL-producer with respect to at most one other table. If a table is joined to a third table, it must be the outer table.
- You can use a table only once as the NULL-producer for one other table within a query.
- You cannot use the same table as both the outer table and the NULL-producer in separate outer joins that form a cycle. A cycle can be formed across multiple joins when the chain of predicates comes back to an earlier table.

  For example, the following query starts with T1 as the outer table in the first predicate and then cycles back to T1 in the third predicate. T2 is used as both the NULL-producer in the first predicate and the outer table in the second predicate, but this usage is not itself a cycle.

  ```
  SELECT ... FROM T1,T2,T3
    WHERE T1.a1 = T2.b2(+)
      AND T2.b2 = T3.c3(+)
      AND T3.c3 = T1.a1(+)   -- invalid cycle
  ```

• AND-factor

- An AND-factor can have only one table as a NULL-producer. Each column reference that is followed by the outer join operator must be from the same table.
- An AND-factor that includes an outer join operator can reference at most two tables.
- If you require multiple AND-factors for the outer join between two tables, you must specify the outer join operator in all of these AND-factors. If an AND-factor does not specify the outer join operator, it is processed on the result of the outer join.
- An AND-factor with predicates that involve only one table can specify the outer join operator if there is at least one other AND-factor that meets the following criteria:
  - The AND-factor must involve the same table as the NULL-producer.
  - The AND-factor must involve another table as the outer table.
- An AND-factor with predicates involving only one table and without an outer join operator is processed on the result of the join.
- An AND-factor that includes an outer join operator must follow the rules for a join-condition of an ON clause that is defined under a joined-table.

Hierarchical queries

A hierarchical query is a form of recursive query that retrieves a hierarchy, such as a bill of materials, from relational data by using a CONNECT BY clause.

Enablement

You enable hierarchical query support by setting the DB2_COMPATIBILITY_VECTOR registry variable to hexadecimal value 0x08 (bit position 4), and then stop and restart the instance to have the new setting take effect.

```
db2set DB2_COMPATIBILITY_VECTOR=08
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Oracle applications, the recommended setting for the DB2_COMPATIBILITY_VECTOR is ORA, which sets all of the compatibility bits.

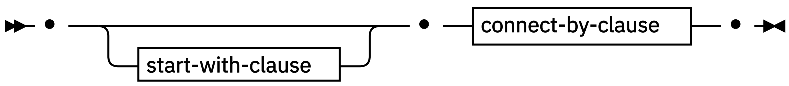
You can then use CONNECT BY syntax, including pseudocolumns, unary operators, and the SYS_CONNECT_BY_PATH scalar function.

A hierarchical query contains a CONNECT BY clause that defines the join conditions between parent and child elements. Connect-by recursion uses the same subquery for the seed (START WITH clause) and the recursive step (CONNECT BY clause). This combination provides a concise method of representing recursions such as bills-of-material, reports-to-chains, or email threads.

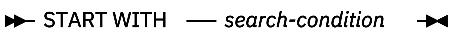
Connect-by recursion returns an error if a cycle occurs. A *cycle* occurs when a row produces itself, either directly or indirectly. By using the optional CONNECT BY NOCYCLE clause, you can direct the recursion to ignore the duplicated row, thus avoiding both the cycle and the error. Hierarchical queries or connect-by recursion differs fromDb2 recursion. For more information about the differences, see Port CONNECT BY to DB2®.

hierarchical-query-clause

A subselect that includes a *hierarchical-query-clause* is called a hierarchical query.



start-with-clause



connect-by-clause

start-with-clause
`START WITH` denotes the seed of the recursion. The *start-with-clause* specifies the intermediate result table H_1 for the hierarchical query. Table H_1 consists of those rows of R for which the *search-condition* is true. If you do not specify the *start-with-clause*, H_1 is the entire intermediate result table R. The rules for the *search-condition* within the *start-with-clause* are the same as those within the WHERE clause.

connect-by-clause
`CONNECT BY` describes the recursive step. The *connect-by-clause* produces the intermediate result table H_{n+1} from H_n by joining H_n with R, using the search condition. If you specify the NOCYCLE keyword, the repeated row is not included in the intermediate result table H_{n+1}. An error is not returned. The rules for the *search-condition* within the *connect-by-clause* are the same as those within the WHERE clause, except that OLAP specifications cannot be specified (SQLSTATE 42903).

After a first intermediate result table H_1 is established, subsequent intermediate result tables H_2, H_3, and so forth are generated. The subsequently created intermediate result tables are generated by joining H_n with table R using the *connect-by-clause* as a join condition to produce H_{n+1}. R is the result of the FROM clause of the subselect and any join predicates in the WHERE clause. The process stops when H_{n+1} yields an empty result table. The result table H of the *hierarchical-query-clause* is the result as if UNION ALL were applied for every intermediate result table..

You can use the unary operator PRIOR to distinguish column references to H_n, the previous recursive step or parent, from column references to R. Consider the following example:

```
CONNECT BY MGRID = PRIOR EMPID
```

MGRID is resolved with R, and EMPID is resolved within the columns of the previous intermediate result table H_n.

Rules

- If the intermediate result table H_{n+1} would return a row from R for a hierarchical path that is the same as a row from R that is already in that hierarchical path, an error is returned (SQLSTATE 560CO).
- If the NOCYCLE keyword is specified, an error is not returned, but the repeated row is not included in the intermediate result table H_{n+1}.
- A maximum of 64 levels of recursion is supported (SQLSTATE 54066).
- A subselect that is a hierarchical query returns the intermediate result set in a partial order, unless you destroy that order by using an explicit ORDER BY clause, a GROUP BY or HAVING clause, or a DISTINCT keyword in the select list. The partial order returns rows such that rows produced in H_{n+1} for a particular

hierarchy immediately follow the row in H_n that produced them. You can use the ORDER SIBLINGS BY clause to enforce order within a set of rows produced by the same parent.

- A hierarchical query is not supported for a materialized query table (SQLSTATE 428EC).
- You cannot use the CONNECT BY clause with XML functions or XQuery (SQLSTATE 428H4).
- You cannot specify a NEXT VALUE expression for a sequence in the following places (SQLSTATE 428F9):
  - The parameter list of the CONNECT_BY_ROOT operator or a SYS_CONNECT_BY_PATH function
  - START WITH and CONNECT BY clauses

Notes

- Hierarchical query support affects the subselect in the following ways:
  - The clauses of the subselect are processed in the following sequence:
    1. FROM clause
    2. *hierarchical-query-clause*
    3. WHERE clause
    4. GROUP BY clause
    5. HAVING clause
    6. SELECT clause
    7. ORDER BY clause
    8. FETCH FIRST clause
  - Special rules apply to the order of processing the predicates in the WHERE clause. The *search-condition* is factored into predicates along with its AND conditions (conjunctions). If a predicate is an implicit join predicate (that is, it references more than one table in the FROM clause), the predicate is applied before the *hierarchical-query-clause* is applied. Any predicate referencing at most one table in the FROM clause is applied to the intermediate result table of the *hierarchical-query-clause*.

    If you write a hierarchical query involving joins, use explicit joined tables with an ON clause to avoid confusion about the application of WHERE clause predicates.
  - You can specify the ORDER SIBLINGS BY clause. This clause specifies that the ordering applies only to siblings within the hierarchies.
- A *pseudocolumn* is a qualified or unqualified identifier that has meaning in a specific context and shares the same namespace as columns and variables. If an unqualified identifier does not identify a column or a variable, the identifier is checked to see whether it identifies a pseudocolumn.

  LEVEL is a pseudocolumn for use in hierarchical queries. The LEVEL pseudocolumn returns the recursive step in the hierarchy at which a row was produced. All rows that are produced by the START WITH clause return the value 1. Rows that are produced by applying the first iteration of the CONNECT BY clause return 2, and so on. The data type of the column is INTEGER NOT NULL.

  You must specify LEVEL in the context of a hierarchical query. You cannot specify LEVEL in the START WITH clause, as an argument of the CONNECT_BY_ROOT operator, or as an argument of the SYS_CONNECT_BY_PATH function (SQLSTATE 428H4).
- Unary operators that support hierarchical queries are CONNECT_BY_ROOT and PRIOR.
- A functions that supports hierarchical queries is the SYS_CONNECT_BY_PATH scalar function.

Examples

- The following reports-to-chain example illustrates connect-by recursion. The example is based on a table named MY_EMP, which is created and populated with data as follows:

```
CREATE TABLE MY_EMP(
  EMPID  INTEGER NOT NULL PRIMARY KEY,
  NAME   VARCHAR(10),
  SALARY DECIMAL(9, 2),
```

```
  MGRID  INTEGER);

INSERT INTO MY_EMP VALUES ( 1, 'Jones',    30000, 10);
INSERT INTO MY_EMP VALUES ( 2, 'Hall',     35000, 10);
INSERT INTO MY_EMP VALUES ( 3, 'Kim',      40000, 10);
INSERT INTO MY_EMP VALUES ( 4, 'Lindsay',  38000, 10);
INSERT INTO MY_EMP VALUES ( 5, 'McKeough', 42000, 11);
INSERT INTO MY_EMP VALUES ( 6, 'Barnes',   41000, 11);
INSERT INTO MY_EMP VALUES ( 7, 'O''Neil',  36000, 12);
INSERT INTO MY_EMP VALUES ( 8, 'Smith',    34000, 12);
INSERT INTO MY_EMP VALUES ( 9, 'Shoeman',  33000, 12);
INSERT INTO MY_EMP VALUES (10, 'Monroe',   50000, 15);
INSERT INTO MY_EMP VALUES (11, 'Zander',   52000, 16);
INSERT INTO MY_EMP VALUES (12, 'Henry',    51000, 16);
INSERT INTO MY_EMP VALUES (13, 'Aaron',    54000, 15);
INSERT INTO MY_EMP VALUES (14, 'Scott',    53000, 16);
INSERT INTO MY_EMP VALUES (15, 'Mills',    70000, 17);
INSERT INTO MY_EMP VALUES (16, 'Goyal',    80000, 17);
INSERT INTO MY_EMP VALUES (17, 'Urbassek', 95000, NULL);
```

The following query returns all employees working for Goyal, as well as some additional information, such as the reports-to-chain:

```
1 SELECT NAME,
2        LEVEL,
3        SALARY,
4        CONNECT_BY_ROOT NAME AS ROOT,
5        SUBSTR(SYS_CONNECT_BY_PATH(NAME, ':'), 1, 25) AS CHAIN
6   FROM MY_EMP
7   START WITH NAME = 'Goyal'
8   CONNECT BY PRIOR EMPID = MGRID
9   ORDER SIBLINGS BY SALARY;
```

```
NAME       LEVEL       SALARY      ROOT  CHAIN
---------- ----------- ----------- ----- ---------------
Goyal                1    80000.00 Goyal :Goyal
Henry                2    51000.00 Goyal :Goyal:Henry
Shoeman              3    33000.00 Goyal :Goyal:Henry:Shoeman
Smith                3    34000.00 Goyal :Goyal:Henry:Smith
O'Neil               3    36000.00 Goyal :Goyal:Henry:O'Neil
Zander               2    52000.00 Goyal :Goyal:Zander
Barnes               3    41000.00 Goyal :Goyal:Zander:Barnes
McKeough             3    42000.00 Goyal :Goyal:Zander:McKeough
Scott                2    53000.00 Goyal :Goyal:Scott
```

Lines 7 and 8 comprise the core of the recursion: The optional START WITH clause describes the WHERE clause that is to be used on the source table to seed the recursion. In this case, only the row for employee Goyal is selected. If the START WITH clause is omitted, the entire source table is used to seed the recursion. The CONNECT BY clause describes how, given the existing rows, the next set of rows is to be found. The unary operator PRIOR is used to distinguish values in the previous step from those in the current step. PRIOR identifies EMPID as the employee ID of the previous recursive step, and MGRID as originating from the current recursive step.

The LEVEL pseudocolumn in line 2 indicates the current level of recursion.

CONNECT_BY_ROOT is a unary operator that always returns the value of its argument as it was during the first recursive step; that is, the values that are returned by an explicit or implicit START WITH clause.

SYS_CONNECT_BY_PATH() is a binary function that prepends the second argument to the first and then appends the result to the value that it produced in the previous recursive step. The arguments must be character types.

Unless explicitly overridden, connect-by recursion returns a result set in a partial order; that is, the rows that are produced by a recursive step always follow the row that produced them. Siblings at the same level of recursion have no specific order. The ORDER SIBLINGS BY clause in line 9 defines an order for these siblings, which further refines the partial order, potentially into a total order.

- Return the organizational structure of the DEPARTMENT table. Use the level of the department to visualize the hierarchy.

```
SELECT LEVEL, CAST(SPACE((LEVEL - 1) * 4) || '/' || DEPTNAME
    AS VARCHAR(40)) AS DEPTNAME
  FROM DEPARTMENT
  START WITH DEPTNO = 'A00'
  CONNECT BY NOCYCLE PRIOR DEPTNO = ADMRDEPT
```

The query returns:

```
LEVEL       DEPTNAME
----------- ----------------------------------------
          1 /SPIFFY COMPUTER SERVICE DIV.
          2     /PLANNING
          2     /INFORMATION CENTER
          2     /DEVELOPMENT CENTER
          3         /MANUFACTURING SYSTEMS
          3         /ADMINISTRATION SYSTEMS
          2     /SUPPORT SERVICES
          3         /OPERATIONS
          3         /SOFTWARE SUPPORT
          3         /BRANCH OFFICE F2
          3         /BRANCH OFFICE G2
          3         /BRANCH OFFICE H2
          3         /BRANCH OFFICE I2
          3         /BRANCH OFFICE J2
```

CONNECT_BY_ROOT unary operator

The CONNECT_BY_ROOT unary operator is for use only in hierarchical queries. For every row in the hierarchy, this operator returns the expression for the root ancestor of the row.

```
►►─ CONNECT_BY_ROOT ── expression ─►◄
```

expression

An expression that does not contain a NEXT VALUE expression, a hierarchical query construct (such as the LEVEL pseudocolumn), the SYS_CONNECT_BY_PATH function, or an OLAP function. If you specify any of these items, SQLSTATE 428H4 is returned.

Usage

The result type of the operator is the result type of the expression.

The following rules apply to the CONNECT_BY_ROOT operator:

- A CONNECT_BY_ROOT operator has a higher precedence than that of any infix operator, such as the plus sign (+) or double vertical bar (||). Therefore, to pass an expression with infix operators as an argument, you must use parentheses. For example, the following expression returns the FIRSTNME value of the root ancestor row concatenated with the LASTNAME value of the actual row in the hierarchy:

  ```
  CONNECT_BY_ROOT FIRSTNME || LASTNAME
  ```

  That expression is equivalent to the first one in the following list but not the second one:

  ```
  (CONNECT_BY_ROOT FIRSTNME) || LASTNAME
  CONNECT_BY_ROOT (FIRSTNME || LASTNAME)
  ```

- A CONNECT_BY_ROOT operator cannot be specified in the START WITH clause or the CONNECT BY clause of a hierarchical query (SQLSTATE 428H4).
- A CONNECT_BY_ROOT operator cannot be specified as an argument to the SYS_CONNECT_BY_PATH function (SQLSTATE 428H4).

The following query returns the hierarchy of departments and their root departments in the DEPARTMENT table:

```
SELECT CONNECT_BY_ROOT DEPTNAME AS ROOT, DEPTNAME
  FROM DEPARTMENT START WITH DEPTNO IN ('B01','C01','D01','E01')
  CONNECT BY PRIOR DEPTNO = ADMRDEPT
```

This query returns the following results:

```
ROOT               DEPTNAME
------------------ ------------------------
PLANNING           PLANNING
INFORMATION CENTER INFORMATION CENTER
DEVELOPMENT CENTER DEVELOPMENT CENTER
DEVELOPMENT CENTER MANUFACTURING SYSTEMS
DEVELOPMENT CENTER ADMINISTRATION SYSTEMS
SUPPORT SERVICES   SUPPORT SERVICES
SUPPORT SERVICES   OPERATIONS
SUPPORT SERVICES   SOFTWARE SUPPORT
SUPPORT SERVICES   BRANCH OFFICE F2
SUPPORT SERVICES   BRANCH OFFICE G2
SUPPORT SERVICES   BRANCH OFFICE H2
SUPPORT SERVICES   BRANCH OFFICE I2
SUPPORT SERVICES   BRANCH OFFICE J2
```

PRIOR unary operator

The PRIOR unary operator is for use only in the CONNECT BY clause of hierarchical queries. To get all subordinates over all levels, the PRIOR operator must be added to the CONNECT BY clause of the hierarchical query.

▶▶─ PRIOR ── *expression* ─▶◀

expression

Any expression that does not contain a NEXT VALUE expression, an hierarchical query construct (such as the LEVEL pseudocolumn), the SYS_CONNECT_BY_PATH function, or an OLAP function. If you specify any of these items, SQLSTATE 428H4 is returned.

Usage

The CONNECT BY clause performs an inner join between the intermediate result table H_n of a hierarchical query and the source result table that you specify in the FROM clause. All column references to tables that are referenced in the FROM clause and that are arguments to the PRIOR operator are considered to range over table H_n.

The result data type of the operator is the result data type of the expression.

As shown in the following example, you typically join the primary key of the intermediate result table H_n to the foreign keys of the source result table to recursively traverse the hierarchy:.

```
CONNECT BY PRIOR T.PK = T.FK
```

If the primary key is a composite key, prefix each column with PRIOR, as shown in the following example:

```
CONNECT BY PRIOR T.PK1 = T.FK1 AND PRIOR T.PK2 = T.FK2
```

A PRIOR operator has a higher precedence than any infix operator, such as the plus sign (+) or double vertical bar (||). Therefore, to pass an expression with infix operators as an argument, you must use parentheses. The parentheses surrounding groups of operands and operators are necessary to indicate the intended order in which operations are to be performed. For example, the following expression returns the FIRSTNME value of the prior row concatenated with the LASTNAME value of the actual row in the hierarchy:

```
PRIOR FIRSTNME || LASTNAME
```

That expression is equivalent to the first one in the following list but not the second one:

```
(PRIOR FIRSTNME) || LASTNAME
PRIOR (FIRSTNME || LASTNAME)
```

If you specify the PRIOR operator outside a CONNECT BY clause of a hierarchical query, SQLSTATE 428H4 is returned.

Example

- The following query returns the hierarchy of departments in the DEPARTMENT table:

```
SELECT LEVEL, DEPTNAME
  FROM DEPARTMENT START WITH DEPTNO = 'A00'
  CONNECT BY NOCYCLE PRIOR DEPTNO = ADMRDEPT
```

This query returns the following results:

```
LEVEL       DEPTNAME
----------- ---------------------------------------
          1 SPIFFY COMPUTER SERVICE DIV.
          2 PLANNING
          2 INFORMATION CENTER
          2 DEVELOPMENT CENTER
          3 MANUFACTURING SYSTEMS
          3 ADMINISTRATION SYSTEMS
          2 SUPPORT SERVICES
          3 OPERATIONS
          3 SOFTWARE SUPPORT
          3 BRANCH OFFICE F2
          3 BRANCH OFFICE G2
          3 BRANCH OFFICE H2
          3 BRANCH OFFICE I2
          3 BRANCH OFFICE J2
```

SYS_CONNECT_BY_PATH

The SYS_CONNECT_BY_PATH function builds a string representing a path from the root to a node in hierarchical queries.

```
►►─ SYS_CONNECT_BY_PATH ─ ( ─ string-expression1 ─ , ─ string-expression2 ─ ) ─►◄
```

The schema is SYSIBM.

string-expression1

A character string expression that identifies the row. The expression must not include any of the items in the following list; otherwise, the SQLSTATE in parentheses is returned:

- A NEXT VALUE expression for a sequence (SQLSTATE 428F9)
- Any hierarchical query construct, such as the LEVEL pseudocolumn or the CONNECT_BY_ROOT operator (SQLSTATE 428H4)
- An OLAP function (SQLSTATE 428H4)
- An aggregate function (SQLSTATE 428H4)

string-expression2

A constant string that serves as a separator. The expression must not include any of the items in the following list; otherwise, the SQLSTATE in parentheses is returned:

- A NEXT VALUE expression for a sequence (SQLSTATE 428F9)
- Any hierarchical query construct, such as the LEVEL pseudocolumn or the CONNECT_BY_ROOT operator (SQLSTATE 428H4)
- An OLAP function (SQLSTATE 428H4)
- An aggregate function (SQLSTATE 428H4)

The result is a varying-length character string. The length attribute of the result data type is the greater of 1000 and the length attribute of *string-expression1*.

The string units of the result data type is the same as the string units of the data type of *string-expression1.*

The string for a particular row at pseudocolumn LEVEL *n* is built as follows:

- Step 1 (using the values of the root row from the first intermediate result table H_1):

```
path1 := string-expression2 || string-expression1
```

- Step *n* (based on the row from the intermediate result table H_n):

```
pathn := pathn-1 || string-expression2 || string-expression1
```

The following rules apply to the SYS_CONTEXT_BY_PATH function:

- If you specify the function outside the context of a hierarchical query, SQLSTATE 428H4 is returned.
- If you use the function in a START WITH clause or a CONNECT BY clause, SQLSTATE 428H4 is returned.

The following example returns the hierarchy of departments in the DEPARTMENT table:

```
   SELECT CAST(SYS_CONNECT_BY_PATH(DEPTNAME, '/')
       AS VARCHAR(76)) AS ORG
     FROM DEPARTMENT START WITH DEPTNO = 'A00'
     CONNECT BY NOCYCLE PRIOR DEPTNO = ADMRDEPT
```

This query returns the following results:

```
ORG
-----------------------------------------------------------------------
/SPIFFY COMPUTER SERVICE DIV.
/SPIFFY COMPUTER SERVICE DIV./PLANNING
/SPIFFY COMPUTER SERVICE DIV./INFORMATION CENTER
/SPIFFY COMPUTER SERVICE DIV./DEVELOPMENT CENTER
/SPIFFY COMPUTER SERVICE DIV./DEVELOPMENT CENTER/MANUFACTURING SYSTEMS
/SPIFFY COMPUTER SERVICE DIV./DEVELOPMENT CENTER/ADMINISTRATION SYSTEMS
/SPIFFY COMPUTER SERVICE DIV./SUPPORT SERVICES
/SPIFFY COMPUTER SERVICE DIV./SUPPORT SERVICES/OPERATIONS
/SPIFFY COMPUTER SERVICE DIV./SUPPORT SERVICES/SOFTWARE SUPPORT
/SPIFFY COMPUTER SERVICE DIV./SUPPORT SERVICES/BRANCH OFFICE F2
/SPIFFY COMPUTER SERVICE DIV./SUPPORT SERVICES/BRANCH OFFICE G2
/SPIFFY COMPUTER SERVICE DIV./SUPPORT SERVICES/BRANCH OFFICE H2
/SPIFFY COMPUTER SERVICE DIV./SUPPORT SERVICES/BRANCH OFFICE I2
/SPIFFY COMPUTER SERVICE DIV./SUPPORT SERVICES/BRANCH OFFICE J2
```

Compatibility database configuration parameters

You can use database configuration parameters to indicate whether the compatibility semantics associated with the certain data types are applied to the connected database.

The compatibility parameters that can be checked are:

date_compat
: Indicates whether the DATE data type compatibility semantics that are associated with the TIMESTAMP(0) data type are applied to the connected database.

number_compat
: Indicates whether the compatibility semantics that are associated with the NUMBER data type are applied to the connected database.

varchar2_compat
: Indicates whether the compatibility semantics that are associated with the VARCHAR2 data type are applied to the connected database.

The value of each of these parameters is determined at database creation time, and is based on the setting of the **DB2_COMPATIBILITY_VECTOR** registry variable. You cannot change the value.

ROWNUM pseudocolumn

Any unresolved and unqualified column reference to the ROWNUM pseudocolumn is converted to the OLAP specification ROW_NUMBER() OVER().

Enablement

You enable ROWNUM pseudocolumn support by setting the **DB2_COMPATIBILITY_VECTOR** registry variable to hexadecimal value 0x01 (bit position 1), and then stop and restart the instance to have the new setting take effect.

```
db2set DB2_COMPATIBILITY_VECTOR=01
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Oracle applications, the recommended setting for the DB2_COMPATIBILITY_VECTOR is ORA, which sets all of the compatibility bits.

ROWNUM numbers the records in a result set. The first record that meets the WHERE clause criteria in a SELECT statement is given a row number of 1, and every subsequent record meeting that same criteria increases the row number.

Both ROWNUM and ROW_NUMBER() OVER() are allowed in the WHERE clause of a subselect and are useful for restricting the size of a result set. If you use ROWNUM in the WHERE clause and there is an ORDER BY clause in the same subselect, the ordering is applied before the ROWNUM predicate is evaluated. Similarly, if you use the ROW_NUMBER() OVER() function in the WHERE clause and there is an ORDER BY clause in the same subselect, the ordering is applied before the ROW_NUMBER() OVER() function is evaluated. If you use the ROW_NUMBER() OVER() function in the WHERE clause, you cannot specify a window-order-clause or a window-partition-clause.

Before translating an unqualified reference to 'ROWNUM' as ROW_NUMBER() OVER() function, Db2 attempts to resolve the reference to one of the following items:

- A column within the current SQL query
- A local variable
- A routine parameter
- A global variable

Avoid using 'ROWNUM' as a column name or a variable name while ROWNUM pseudocolumn support is enabled.

Example

Assuming that ROWNUM pseudocolumn support is enabled for the connected database, retrieve the 20th to the 40th rows of a result set that is stored in a temporary table.

```
    SELECT TEXT FROM SESSION.SEARCHRESULTS
      WHERE ROWNUM BETWEEN 20 AND 40
      ORDER BY ID
```

Note that ROWNUM is affected by the ORDER BY clause.

DUAL table

Any unqualified reference to the table with the name DUAL is resolved as a built-in view that returns one row and one column. The name of the column is DUMMY and its value is 'X'. The DUAL table is similar to the SYSIBM.SYSDUMMY1 table.

Enablement

To enable DUAL table support, set the **DB2_COMPATIBILITY_VECTOR** registry variable to hexadecimal 0x02 (bit position 2), then stop and restart the instance:

```
db2set DB2_COMPATIBILITY_VECTOR=02
db2stop
db2start
```

To activate all compatibility features for Oracle applications, set the **DB2_COMPATIBILITY_VECTOR** registry variable to ORA, then stop and restart the instance:

```
db2set DB2_COMPATIBILITY_VECTOR=ORA
db2stop
db2start
```

Unqualified table references to the DUAL table are resolved as SYSIBM.DUAL.

If a user-defined table named DUAL exists, the Db2 server resolves a table reference to the user-defined table.

Example 1

Generate a random number by selecting from DUAL.

```
SELECT RAND() AS RANDOM_NUMBER FROM DUAL
```

Example 2

Retrieve the value of the CURRENT SCHEMA special register.

```
SET SCHEMA = MYSCHEMA;
SELECT CURRENT SCHEMA AS CURRENT_SCHEMA FROM DUAL;
```

Changed syntax for the TRUNCATE statement

A TRUNCATE statement does not require that the IMMEDIATE keyword to be specified explicitly. The truncate operation is processed immediately regardless of whether IMMEDIATE is specified. If the TRUNCATE statement is not the first statement in the logical unit of work, an implicit commit operation is performed before the TRUNCATE statement is run.

Otherwise, the TRUNCATE statement behaves as described in TRUNCATE statement.

Insensitive cursor

You can make cursors insensitive to subsequent statements by materializing the cursor at OPEN time. Statements that are executed while the cursor is open do not affect the result table once all the rows have been materialized in the temporary copy of the result table.

Enablement

You can enable insensitive cursors by setting the **DB2_COMPATIBILITY_VECTOR** registry variable to hexadecimal value 0x1000 (bit position 13), and then stop and restart the instance to have the new setting take effect.

```
db2set DB2_COMPATIBILITY_VECTOR=1000
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Oracle applications, the recommended setting for the DB2_COMPATIBILITY_VECTOR is ORA, which sets all of the compatibility bits.

When the result set is materialized at OPEN time, the cursor behaves as a read only cursor. All cursors defined as WITH RETURN are INSENSITIVE as long as they are not explicitly marked as FOR UPDATE. If you do not enable insensitive cursor support, there is no guarantee that Db2 cursors will be materialized at OPEN time. Therefore, the result sets that are generated when you run the same query against a Db2 database and a relational database that immediately materializes cursors might be different. For example, Sybase TSQL includes the capability of issuing a query from a batch statement or a procedure that produces a result set for the invoker. The query is materialized immediately. Other statements in the block expect that they cannot affect the result and issue statements, such as DELETE, against the same table that was referenced in the query. When a similar scenario is run without an insensitive cursor, the result set from that cursor will be different from the Sybase result.

Insensitive cursors can also be set in the following ways:

- You can define a cursor as INSENSITIVE in a DECLARE CURSOR statement that is used in a compound SQL (compiled) statement.
- If you bind a package with the **`STATICREADONLY INSENSITIVE`** parameter of the **`BIND`** command, all read-only and ambiguous cursors are insensitive.
- If you specify the `STATICREADONLY INSENSITIVE` option for the **`DB2_SQLROUTINE_PREPOPTS`** registry variable or the SET_ROUTINE_OPTS procedure, at OPEN time, SQL routines materialize all-read only and ambiguous cursors that are issued as static SQL.

Restrictions

The INSENSITIVE keyword is not supported by any of the precompilers. CLI and JDBC do not provide support for identifying insensitive nonscrollable cursors (either cursor attributes or result set attributes).

Examples

This code returns the entire result set of the SELECT statement to the client before executing the DELETE statement.

```
BEGIN
 DECLARE res INSENSITIVE CURSOR WITH RETURN TO CLIENT FOR
  SELECT * FROM T;
  OPEN T;
  DELETE FROM T;
END
```

INOUT parameters

You can define the INOUT parameter for a procedure to have a default value, by using the DEFAULT keyword.

Enablement

You enable INOUT parameter support by setting the **`DB2_COMPATIBILITY_VECTOR`** registry variable to hexadecimal value 0x2000 (bit position 14), and then stop and restart the instance to have the new setting take effect.

```
db2set DB2_COMPATIBILITY_VECTOR=2000
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Sybase applications, the recommended setting for the DB2_COMPATIBILITY_VECTOR is SYB, which sets all of the compatibility bits.

An INOUT parameter is both an input and an output parameter. You can use the DEFAULT keyword to define the default value for an INOUT parameter as either an expression or NULL. If you then invoke the procedure by specifying DEFAULT or no argument for the parameter, the default value that you defined for the parameter is used to initialize it. No value is returned for this parameter when the procedure exits.

Restrictions

The DEFAULT keyword is not supported for INOUT parameters in functions.

Examples

The following code creates a procedure with optional INOUT parameters:

```
CREATE OR REPLACE PROCEDURE paybonus
  (IN empid INTEGER,
  IN percentbonus DECIMAL(2, 2),
  INOUT budget DECFLOAT DEFAULT NULL)
  ...
```

The procedure computes the amount of bonus from the employee's salary, issues the bonus, and then deducts the bonus from the departmental budget. If no budget is specified for the procedure, then the deduction portion is ignored. Examples of how to invoke the procedure follow:

```
CALL paybonus(12, 0.05, 50000);
CALL paybonus(12, 0.05, DEFAULT);
CALL paybonus(12, 0.05);
```

Currently committed semantics

Under *currently committed* semantics, only committed data is returned to readers. However, readers do not wait for writers to release row locks. Instead, readers return data that is based on the currently committed version of data: that is, the version of the data before the start of the write operation.

Lock timeouts and deadlocks can occur under the cursor stability (CS) isolation level with row-level locking, especially with applications that are not designed to prevent such problems. Some high-throughput database applications cannot tolerate waiting on locks that are issued during transaction processing. Also, some applications cannot tolerate processing uncommitted data but still require non-blocking behavior for read transactions.

Currently committed semantics are turned on by default for new databases. You do not have to make application changes to take advantage of the new behavior. To override the default behavior, Set the **cur_commit** database configuration parameter to DISABLED. Overriding the behavior might be useful, for example, if applications require the blocking of writers to synchronize internal logic. During database upgrade from V9.5 or earlier, the **cur_commit** configuration parameter is set to DISABLED to maintain the same behavior as in previous releases. If you want to use currently committed on cursor stability scans, you need to set the **cur_commit** configuration parameter to ON after the upgrade.

Currently committed semantics apply only to read-only scans that do not involve catalog tables and internal scans that are used to evaluate or enforce constraints. Because currently committed semantics are decided at the scan level, the access plan of a writer might include currently committed scans. For example, the scan for a read-only subquery can involve currently committed semantics.

Because currently committed semantics obey isolation level semantics, applications running under currently committed semantics continue to respect isolation levels.

Currently committed semantics require increased log space for writers. Additional space is required for logging the first update of a data row during a transaction. This data is required for retrieving the currently committed image of the row. Depending on the workload, this can have an insignificant or measurable impact on the total log space used. The requirement for additional log space does not apply when **cur_commit** database configuration parameter is set to DISABLED.

Applications running under currently committed semantics will always disregard uncommitted insertions. See Option to disregard uncommitted insertions or ../../com.ibm.db2.luw.admin.regvars.doc/doc/r0005665.dita for details.

Restrictions

The following restrictions apply to currently committed semantics:

- The target table object in a section that is to be used for data update or deletion operations does not use currently committed semantics. Rows that are to be modified must be lock protected to ensure that they do not change after they have satisfied any query predicates that are part of the update operation.
- A transaction that makes an uncommitted modification to a row forces the currently committed reader to access appropriate log records to determine the currently committed version of the row. Although log records that are no longer in the log buffer can be physically read, currently committed semantics do not support the retrieval of log files from the log archive. This affects only databases that you configure to use infinite logging.
- The following scans do not use currently committed semantics:
  - Catalog table scans. Currently committed semantics to apply only to read-only scans that do not involve internal scans that are used to evaluate or enforce constraints. Currently Committed does not apply to internal scans on catalog tables, but may be applied to external scans on catalog tables if the registry variable, **`DB2COMPOPT`**, is set to `LOCKAVOID_EXT_CATSCANS`.
  - Scans that are used to enforce referential integrity constraints
  - Scans that reference LONG VARCHAR or LONG VARGRAPHIC columns
  - Range-clustered table (RCT) scans
  - Scans that use spatial or extended indexes
- If an indoubt transaction occurs (due to an application crash or an unexpected transaction or resource manager outage), any rows that were locked by this indoubt transaction will remain locked until the indoubt transaction is resolved. During this time, a currently committed reader can read the currently committed version of these rows only if the indoubt transaction had previously used currently committed semantics prior to the crash or outage. If not, a currently committed reader will wait for the indoubt transaction to be resolved (and these locks to be released) before reading these rows.
- In a Db2®pureScale® environment, currently committed semantics apply to applications on any member. If a row reader is attempting to access a row which is being updated or deleted by an application on a remote member, the local member will retrieve the currently committed row data from the remote member. However, with the following restrictions:
  - If the remote member is down or performing member crash recovery, then currently committed semantics do not apply. The row reader will wait for member crash recovery to complete on the remote member and for the lock to be released, before reading the row.
  - If communication to the remote member fails while attempting to retrieve the currently committed row data, then currently committed semantics do not apply. The row reader will wait for the lock to be released before reading the row.
  - If the transaction which is updating or deleting the row on the remote member has committed or rolled back before the row reader's communication reaches it, then the row reader will attempt to read the row again. If a transaction on another remote member updates or deletes the row during this time, then the row reader will again attempt to retrieve the currently committed row data from that member. The row reader may give up after a number of attempts and instead wait for the lock to be released before reading the row data.

Monitoring
Currently committed row data retrievals can be monitored on a per-table basis through the **db2pd -tcbstats** option. See *CCLogReads*, *CCRemoteReqs*, *CCLockWaits*, and *CCRemRetryLckWs* values in ../../com.ibm.db2.luw.admin.cmd.doc/doc/r0011729.dita.

Examples

Example 1:

Consider the following scenario, in which deadlocks are avoided by using currently committed semantics. In this scenario, two applications update two separate tables, as shown in step 1, but do not yet commit. Each application then attempts to use a read-only cursor to read from the table that the other application updated, as shown in step 2. These applications are running under the CS isolation level.

Step	Application A	Application B
1	update T1 set col1 = ? where col2 = ?	update T2 set col1 = ? where col2 = ?
2	select col1, col3, col4 from T2 where col2 >= ?	select col1, col5, from T1 where col5 = ? and col2 = ?
3	commit	commit

Without currently committed semantics, these applications running under the cursor stability isolation level might create a deadlock, causing one of the applications to fail. This happens when each application must read data that is being updated by the other application.

Under currently committed semantics, if one of the applications that is running a query in step 2 requires the data that is being updated by the other application, the first application does not wait for the lock to be released. As a result, a deadlock is impossible. The first application locates and uses the previously committed version of the data instead.

Example 2:

Consider the following scenario, in a Db2®pureScale® environment, in which an application avoids a lock wait condition. Application-A on member 1 has updated data on table T1 but not yet committed its changes, and application-B on either member 1 (same member as Application-A) or member 2 (different member then Application-B) using cursor stability isolation level attempts to read that data:

Step	Application-A on pureScale member 1	Application-B on any pureScale member
1	update T1 set col1 = 12where col2 = 'Ava'	
2		select col1 from T1 where col2 = 'Ava'
3	commit	

Without currently committed semantics, application-B would wait until application-A committed its update and released the row lock, before reading the data. Under currently committed semantics, application-B will use the previously committed version of the data instead.

Oracle data dictionary-compatible views

When you set the **DB2_COMPATIBILITY_VECTOR** registry variable to support Oracle data dictionary-compatible views, the views are automatically created when you create a database.

You enable Oracle data dictionary-compatible view support by setting the **DB2_COMPATIBILITY_VECTOR** registry variable to hexadecimal value 0x400 (bit position 11), and then stop and restart the instance to have the new setting take effect.

```
db2set DB2_COMPATIBILITY_VECTOR=400
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Oracle applications, the recommended setting for the DB2_COMPATIBILITY_VECTOR is ORA, which sets all of the compatibility bits.

The data dictionary is a repository for database metadata. The data dictionary views are self-describing. The DICTIONARY view returns a listing of all data dictionary views with comments that describe the content of each view. The DICT_COLUMNS view returns a list of all columns in all data dictionary views. With these two views, you can determine what information is available and how to access it.

There are three different versions of each data dictionary view, and each version is identified by the prefix of the view name.

- ALL_* views return information about objects to which the current user has access.

- DBA_* views return information about all objects in the database, regardless of who owns them.
- USER_* views return information about objects that are owned by the current database user.

Not all versions apply to each view.

The data dictionary definition includes CREATE VIEW, CREATE PUBLIC SYNONYM, and COMMENT statements for each view that is compatible with the Oracle data dictionary. The views, which are created in the SYSIBMADM schema, are listed in Table 6 on page 27.

Table 6. Oracle data dictionary-compatible views

Category	Defined views
General	DICTIONARY, DICT_COLUMNS USER_CATALOG, DBA_CATALOG, ALL_CATALOG USER_DEPENDENCIES, DBA_DEPENDENCIES, ALL_DEPENDENCIES USER_OBJECTS, DBA_OBJECTS, ALL_OBJECTS USER_SEQUENCES, DBA_SEQUENCES, ALL_SEQUENCES USER_TABLESPACES, DBA_TABLESPACES
Tables or views	USER_CONSTRAINTS, DBA_CONSTRAINTS, ALL_CONSTRAINTS USER_CONS_COLUMNS, DBA_CONS_COLUMNS, ALL_CONS_COLUMNS USER_INDEXES, DBA_INDEXES, ALL_INDEXES USER_IND_COLUMNS, DBA_IND_COLUMNS, ALL_IND_COLUMNS USER_TAB_PARTITIONS, DBA_TAB_PARTITIONS, ALL_TAB_PARTITIONS USER_PART_TABLES, DBA_PART_TABLES, ALL_PART_TABLES USER_PART_KEY_COLUMNS, DBA_PART_KEY_COLUMNS, ALL_PART_KEY_COLUMNS USER_SYNONYMS, DBA_SYNONYMS, ALL_SYNONYMS USER_TABLES, DBA_TABLES, ALL_TABLES USER_TAB_COMMENTS, DBA_TAB_COMMENTS, ALL_TAB_COMMENTS USER_TAB_COLUMNS, DBA_TAB_COLUMNS, ALL_TAB_COLUMNS USER_COL_COMMENTS, DBA_COL_COMMENTS, ALL_COL_COMMENTS USER_TAB_COL_STATISTICS, DBA_TAB_COL_STATISTICS, ALL_TAB_COL_STATISTICS USER_VIEWS, DBA_VIEWS, ALL_VIEWS USER_VIEW_COLUMNS, DBA_VIEW_COLUMNS, ALL_VIEW_COLUMNS
Programming objects	USER_PROCEDURES, DBA_PROCEDURES, ALL_PROCEDURES USER_SOURCE, DBA_SOURCE, ALL_SOURCE USER_TRIGGERS, DBA_TRIGGERS, ALL_TRIGGERS USER_ERRORS, DBA_ERRORS, ALL_ERRORS USER_ARGUMENTS, DBA_ARGUMENTS, ALL_ARGUMENTS
Security	USER_ROLE_PRIVS, DBA_ROLE_PRIVS, ROLE_ROLE_PRIVS SESSION_ROLES USER_SYS_PRIVS, DBA_SYS_PRIVS, ROLE_SYS_PRIVS SESSION_PRIVS USER_TAB_PRIVS, DBA_TAB_PRIVS, ALL_TAB_PRIVS, ROLE_TAB_PRIVS USER_TAB_PRIVS_MADE, ALL_TAB_PRIVS_MADE USER_TAB_PRIVS_RECD, ALL_TAB_PRIVS_RECD DBA_ROLES

Examples

The following examples show how to enable, get information about, and use data dictionary-compatible views for a database that is named MYDB:

- Enable the creation of data dictionary-compatible views:

```
db2set DB2_COMPATIBILITY_VECTOR=ORA
db2stop
db2start
db2 create db mydb
```

- Determine what data dictionary-compatible views are available:

```
connect to mydb
select * from dictionary
```

- Use the USER_SYS_PRIVS view to show all the system privileges that the current user has been granted:

```
connect to mydb
select * from user_sys_privs
```

- Determine the column definitions for the DBA_TABLES view:

```
connect to mydb
describe select * from dba_tables
```

Oracle database link syntax

When you set the “DB2_COMPATIBILITY_VECTOR registry variable” on page 29 to enable the use of Oracle database link syntax, you can connect with a remote database, table, or view.

Enablement

You enable Oracle database link syntax support by setting the DB2_COMPATIBILITY_VECTOR registry variable to hexadecimal value 0x20000 (bit position 18), and then stop and restart the instance to have the new setting take effect.

```
db2set DB2_COMPATIBILITY_VECTOR=20000
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Oracle applications, the recommended setting for the DB2_COMPATIBILITY_VECTOR is ORA, which sets all of the compatibility bits.

The database link syntax uses the @ (at sign) to indicate an in or membership condition. For example, to access a remote object `pencils` under schema `user` using a database link to `stock`, you can use:

```
SELECT * FROM user.pencils@stock;
```

Note: The Db2 system supports the use of the @ character as a valid character in an ordinary identifier. For example, you can create a table with pencils@stock as its name. When database link support is enabled, the @ character is treated as a special delimiter in table, view, and column references. If you want to use @ in database object names when link support is enabled, you must enclose the name with double quotes.

Examples

Remote object references are formatted as:

```
<schema_name>,<object_name>@<server_name>
```

Column references can also be included:

```
<schema_name>,<object_name>,<column_name>@<server_name>
```

The following SELECT statements query a remote table named EMPLOYEE:

```
SELECT birthdate FROM rschema.employee@sudb WHERE firstname='SAM'
SELECT rschema.employee.birthdate@sudb FROM rschema.employee@sudb
   WHERE rschema.employee.firstname@sudb ='SAM'
```

You can also issue UPDATE, INSERT, and DELETE statements against a remote table:

```
UPDATE rschema.employee@sudb SET firstname='MARY'
INSERT INTO rschema.employee@sudb VALUES ('Bob')
DELETE FROM rschema.employee@sudb
```

Synonym usage

You can set the **`DB2_COMPATIBILITY_VECTOR`** registry variable to restrict the use of synonyms.

Enablement

You can restrict synonym usage by setting the **DB2_COMPATIBILITY_VECTOR** registry variable to the hexadecimal value 0x40000 (bit position 19), and then stopping and starting the database, as follows:

```
db2set DB2_COMPATIBILITY_VECTOR=40000
db2stop
db2start
```

To take full advantage of the Db2 compatibility features for Oracle applications, you can set the **DB2_COMPATIBILITY_VECTOR** registry variable to ORA, which sets all the compatibility bits.

When you set the **DB2_COMPATIBILITY_VECTOR** registry variable to restrict synonym usage, you cannot issue the following statements with a table synonym as the target:

- ALTER TABLE
- DROP TABLE
- RENAME TABLE
- TRUNCATE

You cannot issue the following statements with a view synonym as the target:

- ALTER VIEW
- DROP VIEW

You cannot issue the following statements with a sequence synonym as the target:

- ALTER SEQUENCE
- DROP SEQUENCE

Examples

The following DROP statement for a table synonym returns an error when you set the **DB2_COMPATIBILITY_VECTOR** registry variable to support the use of synonyms:

```
   CREATE TABLE T (C1 INT)
   CREATE SYNONYM S FOR TABLE T
   DROP TABLE S
```

DB2_COMPATIBILITY_VECTOR registry variable

The Db2 product provides optional features that simplify the task of migrating applications from other relational database products such as Oracle, Sybase, and MySQL. These features are inactive by default, but the **DB2_COMPATIBILITY_VECTOR** registry variable can be used to activate any subset of them.

Registry variable settings

The following values can be set for the **DB2_COMPATIBILITY_VECTOR** registry variable:

NULL
: No compatibility features are activated. This is the default.

A hexadecimal number in the range 00000000 - FFFFFFFF
: A hexadecimal number that represents a binary string. The value (0 or 1) of each bit position in the string indicates whether the corresponding compatibility feature is enabled (1) or disabled (0). Table 7 on page 30 maps each bit to the feature that it controls.

ORA
: This value improves the compatibility of **Oracle** applications. It activates the compatibility features for which there is a bullet in the ORA column of Table 7 on page 30. (For more information about the Oracle compatibility features, see Oracle to DB2 Conversion Guide: Compatibility Made Easy.) In addition, it changes the default value of the DB2_DEFERRED_PREPARE_SEMANTICS registry variable

to either 'YES' (in a single-byte character set environment) or 'YES_DBCS_GRAPHIC_TO_CHAR' (in a double-byte character set environment).

SYB

This value improves the compatibility of **Sybase** applications. It activates the compatibility features for which there is a bullet in the SYB column of Table 7 on page 30. In addition, it changes the default value of the DB2_DEFERRED_PREPARE_SEMANTICS registry variable to 'YES'.

MYS

This value improves the compatibility of **MySQL** applications. It changes the default value of the DB2_DEFERRED_PREPARE_SEMANTICS registry variable to either 'YES' (in a single-byte character set environment) or 'YES_DBCS_GRAPHIC_TO_CHAR' (in a double-byte character set environment).

For more information about the DB2_DEFERRED_PREPARE_SEMANTICS registry variable, see Query compiler variables.

Important: When you enable a compatibility feature, some SQL behavior will vary from what is documented in the SQL reference information. These behavior differences are described in the documentation for the corresponding features.

Table 7. DB2_COMPATIBILITY_VECTOR bit positions

Bit position	Hexadecimal value	ORA	SYB	Compatibility feature	Description
1	0x01	•		ROWNUM pseudocolumn	This bit enables the use of the ROWNUM pseudocolumn as a synonym for the ROW_NUMBER() OVER() function and permits the ROWNUM pseudocolumn to appear in the WHERE clause of SQL statements.
2	0x02	•		DUAL table	This bit resolves unqualified references to the DUAL table as SYSIBM.DUAL.
3	0x04			(obsolete)	This bit formerly activated support for the outer join operator. That feature is now always active and now this bit is ignored.
4	0x08	•		Hierarchical queries	This bit enables support for hierarchical queries, which use the CONNECT BY clause.
5	0x10	•		NUMBER data type [1]	This bit enables support for the NUMBER data type and associated numeric processing. When you create a database with this support enabled, the **`number_compat`** database configuration parameter is set to ON.
6	0x20	•		VARCHAR2 data type [1]	This bit enables support for the VARCHAR2 and NVARCHAR2 data types and associated character string processing. When you create a database with this support enabled, the **`varchar2_compat`** database configuration parameter is set to ON.

Table 7. DB2_COMPATIBILITY_VECTOR bit positions (continued)					
Bit position	**Hexadecimal value**	**ORA**	**SYB**	**Compatibility feature**	**Description**
7	0x40	•		DATE data type [1]	This bit enables the interpretation of the DATE data type as the TIMESTAMP(0) data type so that it includes time information as well as date information. For example, in date compatibility mode, the statement "VALUES CURRENT DATE" returns a value like 2016-02-17-10.43.55. When you create a database with this support enabled, the **date_compat** database configuration parameter is set to ON.
8	0x80	•		TRUNCATE TABLE	This bit enables alternative semantics for the TRUNCATE statement so that IMMEDIATE is an optional keyword and is the default. If the TRUNCATE statement is not the first statement in the logical unit of work, an implicit commit operation is carried out before the TRUNCATE statement is executed.
9	0x100	•	•	Character literals	This bit enables the ability to assign a CHAR or GRAPHIC data type instead of a VARCHAR or VARGRAPHIC data type to a character or graphic string constant whose byte length is less than or equal to 254.
10	0x200	•		Collection methods	This bit enables the use of methods to perform operations on arrays, such as first, last, next, and previous. This value also enables the use of parentheses in place of square brackets in references to specific elements in an array. For example, array1(*i*) refers to element *i* of array1.
11	0x400	•		Oracle data dictionary-compatible views [1]	This bit enables the creation of Oracle data dictionary-compatible views.
12	0x800	•		PL/SQL compilation [2]	This bit enables the compilation and execution of PL/SQL statements and language elements.
13	0x1000		•	Insensitive cursors	This bit enables cursors that are defined with WITH RETURN to be insensitive if the select-statement does not explicitly specify FOR UPDATE.
14	0x2000		•	"INOUT parameters" on page 23	This bit enables the specification of DEFAULT for INOUT parameter declarations.

Table 7. DB2_COMPATIBILITY_VECTOR bit positions (continued)					
Bit position	**Hexadecimal value**	**ORA**	**SYB**	**Compatibility feature**	**Description**
15	0x4000			(obsolete)	This bit formerly activated LIMIT and OFFSET support, but that feature is now always active and this bit is now ignored.
16	0x8000			(reserved)	This bit is currently not used.
17	0x10000	•		"SQL data-access-level enforcement" on page 10	This bit enables routines to enforce SQL data-access levels at run time.
18	0x20000	•		"Oracle database link syntax" on page 28	This bit enables Oracle database link syntax for accessing objects in other databases.
19	0x40000	•		"Synonym usage" on page 28	This bit disables the use of synonyms in some SQL statements. When you set the **DB2_COMPATIBILITY_VECTOR** registry variable to restrict synonym usage, you cannot issue the alter, drop, rename, or truncate statements with a table synonym as the target. You cannot issue the alter or drop statements with a view synonym as the target. You cannot issue the alter or drop statement with a sequence synonym as the target.

1. This feature applies only at the time of database creation. Enabling or disabling this feature does not affect existing databases, but only newly created databases.
2. See Restrictions on PL/SQL support.

Usage

You set and update the **DB2_COMPATIBILITY_VECTOR** registry variable by using the **db2set** command. You can set the **DB2_COMPATIBILITY_VECTOR** registry variable with combination of the compatibility features by adding the digits of the hexadecimals values that are associated with the compatibility features. A new setting for the registry variable does not take effect until after you stop and restart the instance. Also, you must rebind Db2 packages for the change to take effect. Packages that you do not rebind explicitly pick up the change at the next implicit rebind.

Example 1

To set the registry variable to enable all the supported Oracle compatibility features:

```
db2set DB2_COMPATIBILITY_VECTOR=ORA
db2stop
db2start
```

Example 2

To set the registry variable to provide both the ROWNUM pseudocolumn (0x01) and DUAL table (0x02) support that is specified in the previous table:

```
db2set DB2_COMPATIBILITY_VECTOR=03
db2stop
db2start
```

Example 3

To disable all compatibility features by resetting the **DB2_COMPATIBILITY_VECTOR** registry variable:

```
db2set DB2_COMPATIBILITY_VECTOR=
db2stop
db2start
```

Setting up the Db2 environment for Oracle application enablement

You can reduce the time and complexity of enabling Oracle applications to work with Db2 data servers if you set up the Db2 environment appropriately.

Before you begin

- A Db2 data server product must be installed.
- You require SYSADM and the appropriate operating system authority to issue the **db2set** command.
- You require SYSADM or SYSCTRL authority to issue the **CREATE DATABASE** command.

About this task

The Db2 product can support many commonly referenced features from other database products. This task is a prerequisite for executing PL/SQL statements or SQL statements that reference Oracle data types from Db2 interfaces or for using any other SQL compatibility features. You enable Db2 compatibility features at the database level; you cannot disable them.

Procedure

To enable Oracle applications to work with Db2 data servers:

1. In a Db2 command window, start the Db2 database manager by issuing the following command:

   ```
   db2start
   ```

2. Set the **DB2_COMPATIBILITY_VECTOR** registry variable to one of the following values:

   - The hexadecimal value that enables the specific compatibility feature that you want to use.
   - To take advantage of all theDb2 compatibility features, ORA, as shown in the following command. This is the recommended setting.

   ```
   db2set DB2_COMPATIBILITY_VECTOR=ORA
   ```

3. Enable deferred prepare support by setting the **DB2_DEFERRED_PREPARE_SEMANTICS** registry variable to YES, as shown:

   ```
   db2set DB2_DEFERRED_PREPARE_SEMANTICS=YES
   ```

   If you set the **DB2_COMPATIBILITY_VECTOR** registry variable to ORA and do not set the **DB2_DEFERRED_PREPARE_SEMANTICS** registry variable, a default value of YES is used in the SBCS or Unicode database environment. When you are in the DBCS environment, the default value is **YES_DBCS_GRAPHIC_TO_CHAR**.

4. Stop the database manager by issuing the **db2stop** command:

```
db2stop
```

5. Start the database manager by issuing the **db2start** command:

```
db2start
```

6. Create your Db2 database by issuing the **CREATE DATABASE** command. By default, databases are created as Unicode databases.
   For example, to create a database that is named DB, issue the following command:

```
db2 CREATE DATABASE DB
```

7. Optional: Run a Command Line Processor Plus (CLPPlus) or command line processor (CLP) script (for example, `script.sql`) to verify that the database supports PL/SQL statements and data types.

   The following CLPPlus script creates and then calls a simple procedure:

```
CONNECT user@hostname:port/dbname;

CREATE TABLE t1 (c1 NUMBER);

CREATE OR REPLACE PROCEDURE testdb(num IN NUMBER, message OUT VARCHAR2)
AS
BEGIN
 INSERT INTO t1 VALUES (num);

 message := 'The number you passed is: ' || TO_CHAR(num);
END;
/

CALL testdb(100, ?);

DISCONNECT;
EXIT;
```

   To run the CLPPlus script, issue the following command:

```
clpplus @script.sql
```

   The following example shows the CLP version of the same script. This script uses the **SET SQLCOMPAT PLSQL** command to enable recognition of the forward slash character (/) on a new line as a PL/SQL statement termination character.

```
CONNECT TO DB;

SET SQLCOMPAT PLSQL;

-- Semicolon is used to terminate
-- the CREATE TABLE statement:
CREATE TABLE t1 (c1 NUMBER);

-- Forward slash on a new line is used to terminate
-- the CREATE PROCEDURE statement:
CREATE OR REPLACE PROCEDURE testdb(num IN NUMBER, message OUT VARCHAR2)
AS
BEGIN
 INSERT INTO t1 VALUES (num);

 message := 'The number you passed is: ' || TO_CHAR(num);
END;
/

CALL testdb(100, ?);

SET SQLCOMPAT DB2;

CONNECT RESET;
```

   To run the CLP script, issue the following command:

```
db2 -tvf script.sql
```

Results

The Db2 database that you created is enabled for Oracle applications. You can now use the compatibility features that you enabled. Only databases created after the **DB2_COMPATIBILITY_VECTOR** registry variable is set are enabled for Oracle applications.

What to do next

- Start using the CLPPlus interface.
- Execute PL/SQL scripts and statements.
- Transfer database object definitions.
- Enable database applications.

Terminology mapping: Oracle to Db2 products

Because Oracle applications can be enabled to work with Db2 data servers when the Db2 environment is set up appropriately, it is important to understand how certain Oracle concepts map to Db2 concepts.

This section provides an overview of the data management concepts used by Oracle, and the similarities or differences between these concepts and those used by Db2 products. Table 8 on page 35 provides a concise summary of commonly used Oracle terms and their Db2 equivalents.

Table 8. Mapping of common Oracle concepts to Db2 concepts

Oracle concept	**Db2 concept**	**Notes**
active log	active log	The concepts are the same.
actual parameter	argument	The concepts are the same.
alert log	`db2diag` log files and administration notification log	The `db2diag` log files are primarily intended for use by IBM Software Support for troubleshooting. The administration notification log is primarily intended for use by database and system administrators for troubleshooting. Administration notification log messages are also logged in the `db2diag` log files, using a standardized message format.
archive log	offline archive log	The concepts are the same.
archive log mode	log archiving	The concepts are the same.
background_dump_dest	diagpath	The concepts are the same.
created global temporary table	created global temporary table	The concepts are the same.
cursor sharing	statement concentrator	The concepts are the same.
data block	data page	The concepts are the same.
data buffer cache	buffer pool	The concepts are the same. However, in the Db2 product, you can have as many buffer pools as you like, of any page size.

Table 8. Mapping of common Oracle concepts to Db2 concepts (continued)		
Oracle concept	**Db2 concept**	**Notes**
data dictionary	system catalog	The Db2 system catalog contains metadata in the form of tables and views. The database manager creates and maintains two sets of system catalog views that are defined on the base system catalog tables: **SYSCAT views** Read-only views. **SYSSTAT views** Updatable views that contain statistical information that is used by the optimizer.
data dictionary cache	catalog cache	The concepts are the same.
data file	container	Db2 data is physically stored in containers, which contain objects.
database link	nickname	A nickname is an identifier that refers to an object at a remote data source, that is, a federated database object.
dual table	dual table	The concepts are the same.
dynamic performance views	SQL administrative views	SQL administrative views, which use schema SYSIBMADM, return system data, database configuration, and monitor data about a specific area of the database system.
extent	extent	A Db2 extent is made up of a set of contiguous data pages.
formal parameter	parameter	The concepts are the same.
global index	nonpartitioned index	The concepts are the same.
inactive log	online archive log	The concepts are the same.

Table 8. Mapping of common Oracle concepts to Db2 concepts (continued)		
Oracle concept	**Db2 concept**	**Notes**
`init.ora` file and Server Parameter File (SPFILE)	database manager configuration file and database configuration file	A Db2 instance can contain multiple databases. Therefore, configuration parameters and their values are stored at both the instance level, in the database manager configuration file, and at the database level, in the database configuration file. You manage the database manager configuration file through the **`GET DBM CFG`** or **`UPDATE DBM CFG`** command. You manage the database configuration file through the **`GET DB CFG`** or **`UPDATE DB CFG`** command.
instance	instance or database manager	An instance is a combination of background processes and shared memory. A Db2 instance is also known as a database manager.
large pool	utility heap	The utility heap is used by the backup, restore, and load utilities.
library cache	package cache	The package cache, which is allocated from database shared memory, is used to cache sections for static and dynamic SQL and XQuery statements executed on a database.
local index	partitioned index	This is the same concept.
materialized view	materialized query table (MQT)	An MQT is a table whose definition is based on the results of a query and can help improve performance. The Db2 SQL compiler determines whether a query would run more efficiently against an MQT than it would against the base table on which the MQT is based.
noarchive log mode	circular logging	The concepts are the same.
Oracle Call Interface (OCI) Oracle Call Interface (OCI)	DB2CI	DB2CI is a C and C++ application programming interface that uses function calls to connect to Db2 databases, manage cursors, and perform SQL statements. For more information, see "IBM Data Server Driver for DB2CI" for a list of OCI APIs supported by the Db2CI driver.

Table 8. Mapping of common Oracle concepts to Db2 concepts (continued)		
Oracle concept	**Db2 concept**	**Notes**
Oracle Call Interface (OCI) Oracle Call Interface (OCI)	Call Level Interface (CLI)	CLI is a C and C++ application programming interface that uses function calls to pass dynamic SQL statements as function arguments. In most cases, you can replace an OCI function with a CLI function and relevant changes to the supporting program code.
ORACLE_SID environment variable	**DB2INSTANCE** environment variable	The concepts are the same.
partitioned tables	partitioned tables	The concepts are the same.
Procedural Language/Structured Query Language (PL/SQL)	SQL Procedural Language (SQL PL)	SQL PL is an extension of SQL that consists of statements and other language elements. SQL PL provides statements for declaring variables and condition handlers, assigning values to variables, and implementing procedural logic. SQL PL is a subset of the SQL/ Persistent Stored Modules (SQL/ PSM) language standard. You can use Db2 data server interfaces to compile and execute Oracle PL/SQL statements.
program global area (PGA)	application shared memory and agent private memory	Application shared memory stores information that is shared between a database and a particular application: primarily, rows of data that are passed to or from the database. Agent private memory stores information that is used to service a particular application, such as sort heaps, cursor information, and session contexts.
redo log	transaction log	The transaction log records database transactions. You can use it for recovery.
role	role	The concepts are the same.
segment	storage object	The concepts are the same.
session	session; database connection	The concepts are the same.
startup nomount command	**db2start** command	The command used to start the instance.

Table 8. Mapping of common Oracle concepts to Db2 concepts (continued)

Oracle concept	Db2 concept	Notes
synonym	alias	An alias is an alternative name for a table, a view, a nickname, or another alias. The term *synonym* can be specified instead of *alias*. Aliases are not used to control what version of a Db2 procedure or user-defined function is used by an application. To control the version, use the SET PATH statement to add the required schema to the value of the CURRENT PATH special register.
system global area (SGA)	instance shared memory and database shared memory	The instance shared memory stores all of the information for a particular instance, such as lists of all active connections and security information. The database shared memory stores information for a particular database, such as package caches, log buffers, and buffer pools.
SYSTEM table space	SYSCATSPACE table space	The SYSCATSPACE table space contains the system catalog. This table space is created by default when you create a database.
table space	table space	The concepts are the same.
user global area (UGA)	application global memory	Application global memory comprises application shared memory and application-specific memory.

IBM PureData System for Analytics (Netezza) to Db2 migration

Use the information in this section to migrate from IBM PureData® System for Analytics (Netezza®) to Db2 systems and to learn about Netezza and Db2 compatibility. A Netezza system and a Db2 system are highly compatible.

Migrating from IBM PureData System for Analytics (Netezza) to a Db2 system

Use the information in this topic to understand and perform the end-to-end process of migrating from IBM PureData System for Analytics (Netezza) to a Db2 system.

Procedure

1. Plan your migration.
2. Acquire your system:
   - IBM provisions the Db2 managed service environment. For information, see the IBM Bluemix® page.

- You provision your own Db2 Warehouse environment. For information, see Deploying Db2 Warehouse.

3. Create database objects, such as tables.

   You do not need to create the Db2 database itself; it is created for you. To create database objects, execute the converted DDL on the web console **Analytics** or **Run SQL** tab, or use another command-line interface.
4. Move your data.

   For the Db2 managed service, use IBM BlueMix Lift. When you use BlueMix Lift, you can set up processes to keep the moved data synchronized with the source during the activities in the following steps. For Db2 Warehouse, use the **`db_migrate`** command; see Moving data using db_migrate.
5. Migrate your users, groups and security configuration.

   For more information, see Security.
6. Map your system views.
7. Migrate your queries.
8. Migrate your routines.
9. Migrate user-defined extensions.

   See "IBM PureData System for Analytics (Netezza) and Db2 user-defined extensions compatibility" on page 52, Deploying R, and Installing Python packages.
10. Redirect users and applications to the Db2 database. For example, change drivers or BI tools to point to the database.
11. Validate your migration.

Planning to migrate from IBM PureData System for Analytics to Db2

You should review the migration steps and prepare your environment before you migrate your system.

Procedure

- Become familiar with the differences between Db2 offerings. For information, see the Db2 overview.
- Learn about PureData® System for Analytics and Db2 compatibility by reviewing the migration process and the compatibility topics.
- Evaluate the compatibility of your PureData System for Analytics SQL commands in a Db2 environment by using IBM Database Harmony Profiler. For information about this tool, see the Database Conversion Workbench page. By submitting your evaluation reports to askdcw@ca.ibm.com, you can help IBM to prioritize and plan for the closure of compatibility gaps.
- Determine your network requirements. For example, for IBM Db2 Warehouse, review your firewall and port requirements and ground-to-cloud connectivity requirements, and review the "Network prerequisites" section in the applicable prerequisites topic. For the Db2 managed services, decide whether you need a VPN and understand how to set it up.

  You can use the IBM Cloud Integrated Analytics Environment (CIAE) to set up a secure connection between your Db2 system and services such as your business intelligence tool and the source database that feeds your Db2 system.
- Determine your security requirements.
- Review your architecture: the products that you are using and where they are located.
- Size your system.
- If you are using Db2 Warehouse:
  - Determine your file transfer requirements: your backup requirements, daily transfer volumes, and SLAs for load time.
  - Back up your data. For information, see Snapshot backup and restore on Db2 Warehouse.

    Restriction: You cannot use an IBM PureData System for Analytics backup to restore to a Db2 database.

IBM PureData System for Analytics (Netezza) and Db2 database, table, and schema compatibility

In a PureData System for Analytics (Netezza) system, you can have multiple databases. In a Db2 system, there is one database, named BLUDB.

By default, when you move Netezza databases to a Db2 system by using the **db_migrate** command or Database Harmony Profiler, the Netezza databases are migrated to schemas within the BLUDB database. For information about the **db_migrate** command, see Moving data using db_migrate; for information about Database Harmony Profiler, see the Database Conversion Workbench page. If you configured your Netezza databases to use multiple schemas, the tools move the Netezza schemas to the BLUDB database, but if you are using the multiple schemas within multiple databases, additional work is required before you migrate. For information about how to proceed with your migration if you are using multiple databases with multiple schemas, see "IBM PureData System for Analytics (Netezza) configuration choices that impact migration to Db2products" on page 60.

Other notable differences between a Netezza system and a Db2 include:

- By default, data in Db2 tables is organized by column (it is possible to organize data by row). Organizing data by column reduces the amount of I/O that is needed for processing a query because only the columns that are referenced in the query must be loaded into memory from disk. A column-based organization benefits analytic queries that access a large number of values from a subset of the columns and heavily use aggregations and joins.
- A Db2 system uses UTF-8 encoding. A Netezza system uses Latin-9 encoding for single-byte characters and UTF-8 for multibyte characters.
- Clustered base tables (CBTs) are not supported in a Db2 system.
- Materialized views are not supported in a Db2 system. Materialized query tables (MQTs) might be a suitable alternative. An MQT is a table whose definition is based on the result of a query and whose data is in the form of precomputed results that are taken from the table or tables on which the MQT definition is based.
- In a Db2 system, whether primary key constraints are enforced by default is determined by the setting of the **ddl_constraint_def** configuration parameter. The default setting of this parameter is YES (for ENFORCED). You can override the default behavior by explicitly specifying either ENFORCED or NOT ENFORCED in your DDL statements.
- Unlike Netezza, when Db2 performs query optimization, it always trusts that primary keys are unique. Consequently, if primary key constraints are not enforced, and if a primary key column contains duplicate values, query results might be incorrect.

For details about differences between Netezza and Db2 DDL, see "IBM PureData System for Analytics (Netezza) and Db2 SQL compatibility" on page 73.

IBM PureData System for Analytics (Netezza) and Db2 data loading compatibility

There are multiple options for loading data into a Db2 database. These including loading data from a local file by using the web console, loading from an IBM Cloudant® database, and loading by using IBM InfoSphere® DataStage® or IBM InfoSphere Data Replication. A CLPPlus **IMPORT** command is also available.

For more information about how to load data in a Db2 environment, see Loading your data and IMPORT CLPPlus command.

IBM PureData System for Analytics (Netezza) and Db2 security compatibility

In Netezza, authentication can occur either within the database or with an external entity such as an external LDAP server, if you configured the product to support that. Authorization occurs within the database. In Db2, authentication occurs outside the database, and authorization generally occurs inside the database.

In Netezza, privileges are typically not granted directly to the user. Instead, they are granted indirectly, to groups to which users are assigned. Db2 products also use database objects to help simplify administration. In Db2, privileges and authorities (a somewhat similar concept to privileges) are typically granted to roles to which users or other roles are assigned. All of the authorities and privileges that you grant to a particular role are inherited by whatever users or other roles that you assign to that particular role. A Db2 system comes with two built-in user roles:

Administrator
: People with the Administrator role have access to all of the features in the web console. They can manage database access by creating and deleting users, assigning users to roles, and performing other security-related functions, such as changing user passwords. A Db2 environment comes with a built-in ID called bluadmin that has the Administrator role.

User
: People with the User role have access to many of the features in the web console and can manage their own user profiles. They also have full access to their own tables and can give other users permission to access and use those tables.

If the built-in user roles do not provide enough flexibility, people with the Administrator role can create user-defined roles with different authorities and privileges and assign those roles to users.

To add or delete users and assign them to or remove them from built-in roles, people with the Administrator role use the web console. People with the Administrator role can also use the console to grant privileges and authorities to roles and revoke that access by creating GRANT and REVOKE SQL statements either directly or, in some cases, by using GUI controls. To create or delete user-defined roles and manage membership in those roles, people with the Administrator role can issue SQL statements by using the web console, CLPPlus, or the Data Studio client.

In a Netezza system, you can use Multi-Level Security (MLS) to define rules to control access to row-secure tables. In a Db2 system, you can use row and column access control (RCAC) to control access to a table at the row level, column level, or both. RCAC is based on two sets of rules: one set operates on rows (row permissions), and the other set operates on columns (column masks). To create, alter, and drop RCAC rules, you use SQL statements.

For details about differences between Netezza and Db2 SQL that are related to security, see "IBM PureData System for Analytics (Netezza) and Db2 SQL compatibility" on page 73.

Related information

Security features of IBM® Db2 on Cloud, IBM Db2 Warehouse on Cloud, and IBM Db2 Warehouse

Row and column access control (RCAC) overview

Setting up an external LDAP server for IBM Db2 Warehouse

Migrating users, groups, and privileges from IBM PureData System for Analytics (Netezza) to Db2

As part of Netezza to Db2 migration, you must migrate your Netezza users, groups, and privileges.

Procedure

1. Collect information about your Netezza environment by issuing the statements and commands in the following table for each Netezza database.

   To issue each SELECT statement, put it in a file and issue the following command:

   ```
   nzsql dbname -f sql_file > output_file_dbname
   ```

Information to collect	Statement or command to issue
List of users	`SELECT USERNAME FROM _V_USER`
List of groups	`SELECT DISTINCT(GROUPNAME) FROM _V_USERGROUPS`
List of users and their associated groups	`SELECT USERNAME, GROUPNAME FROM _V_GROUPUSERS`

Information to collect	Statement or command to issue
List of group privileges	`nz_ddl_grant_group -usrobj dbname > output_file_dbname`
List of user privileges	`nz_ddl_grant_user -usrobj dbname > output_file_dbname`

2. Replicate the roles on your Db2 system as follows:
   a. Create the roles by issuing the following statement:

   ```
   CREATE ROLE ROLE_NAME
   ```

   b. Verify that the roles were added in one of the following ways:
   - Issue the **dbsql** command with the **\dr** option.
   - Issue the following statement:

   ```
   SELECT ROLENAME FROM SYSCAT.ROLES
   ```

3. Grant privileges and authorities to roles as follows.

   For a mapping of Netezza privileges to Db2 privileges and authorities, see "IBM PureData System for Analytics (Netezza) and Db2 privilege compatibility" on page 44.

   a. Issue the following statements:

   ```
   GRANT PRIVILEGE ON OBJECT TO ROLE ID
   GRANT AUTHORITY ON DATABASE TO ROLE ID
   ```

   b. Verify that the privileges were granted by issuing the following statement:

   ```
   SELECT  *  FROM  SYSIBMADM.PRIVILEGES  WHERE  AUTHID='BLUADMIN' AND OBJECTNAME
   ='MY_OBJECT'
   ```

   c. Verify that the authorities were granted by issuing the following statement:

   ```
   SELECT CHAR(GRANTEE,8) AS GRANTEE, CHAR(GRANTEETYPE,1) AS TYPE,
   CHAR(DBADMAUTH,1) AS DBADM, CHAR(CREATETABAUTH,1) AS CREATETAB,
   CHAR(BINDADDAUTH,1) AS BINDADD, CHAR(CONNECTAUTH,1) AS CONNECT,
   CHAR(NOFENCEAUTH,1) AS NOFENCE, CHAR(IMPLSCHEMAAUTH,1) AS IMPLSCHEMA,
   CHAR(LOADAUTH,1) AS LOAD, CHAR(EXTERNALROUTINEAUTH,1) AS EXTROUTINE,
   CHAR(QUIESCECONNECT AUTH,1) AS QUIESCECONN, CHAR(LIBRARYADMAUTH,1) AS LIBADM,
   CHAR(SECURITYADMAUTH,1) AS SECURITYADM
   FROM  SYSCAT.DBAUTH WHERE GRANTEE='BLUADMIN'
   ```

4. Replicate the users on your Db2 system as follows:
   a. Create the users by using one of the following methods:
   - Use the web console.
   - Use the REST API as follows:

   ```
   curl -k -u "bluadmin:bluadmin"
   -H 'Content-Type: application/json'
   -H 'Accept: t'
   -d '{"users":
   [{"password":"password","userid":"userid","userProfile":"","userRole":
   [{"role":"Administrator"}]}]}'
   -X POST "https://<dashlocal-host>:8443/dashdb-api/users"
   ```

   b. Check that the users were created by using one of the following methods:
   - Use the web console.
   - Issue the **dbsql** command with the **\du** option.

5. Grant roles to users as follows:
   a. Issue the following statement:

   ```
   GRANT ROLE ROLE1 TO USER USER1
   ```

   b. Verify the role–user mapping by issuing the **dbsql** command with the **\dR** option.
6. If necessary, grant privileges and authorities directly to users.

   This is usually not recommended: using roles simplifies administration.

   To grant privileges and authorities to users, adapt the instructions in step “3” on page 43.

Related reference

GRANT (database authorities)

GRANT (schema privileges)

GRANT (table, view, or nickname privileges)

GRANT (sequence privileges)

GRANT (routine privileges)

IBM PureData System for Analytics (Netezza) and Db2 privilege compatibility

Many Netezza privileges have similar or equivalent Db2 privileges. In some cases, a Netezza privilege maps to a Db2 authority.

Object privileges

Use the following table to map Netezza object privileges to Db2 privileges or authorities.

For sequences, only the ALTER and USAGE privileges apply.

Table 9. Object privileges

Netezza privilege	Db2 mapping
ABORT	WLMADM authority (required if you want to cancel an activity by using the WLM_CANCEL_ACTIVITY procedure)
ALTER	ALTER privilege; also, ALTERIN privilege (applies only to schemas)
DELETE	DELETE privilege
DROP	DROPIN privilege (applies only to schemas)
EXECUTE	EXECUTE privilege (applies only to functions and procedures)
EXECUTE AS	No equivalent available
GENSTATS	CONTROL privilege (for a particular table) SQLADM authority (for all tables in the database)
GROOM	CONTROL privilege (for a particular table) SQLADM authority (for all tables in the database)
INSERT	INSERT privilege
LABEL ACCESS	SECADM authority
LABEL EXPAND	SECADM authority
LABEL RESTRICT	SECADM authority

Table 9. Object privileges (continued)	
Netezza privilege	**Db2 mapping**
LIST	No equivalent available
LOAD	INSERT privilege
SELECT	SELECT privilege
TRUNCATE	DELETE privilege
UPDATE	UPDATE privilege

Administration privileges
Use the following table to map Netezza administration privileges to Db2 privileges or authorities.

Table 10. Administration privileges	
Netezza privilege	**Db2 mapping**
BACKUP	No equivalent available
CREATE AGGREGATE	CREATE_EXTERNAL_ROUTINE authority
CREATE DATABASE	CREATE SCHEMA authority or DBADM authority, which includes CREATE SCHEMA authority (Netezza databases are mapped to Db2 schemas)
CREATE EXTERNAL TABLE	CREATETAB authority
CREATE FUNCTION	CREATE_EXTERNAL_ROUTINE authority
CREATE GROUP	SECADM authority (required to create a role)
CREATE INDEX	CONTROL privilege
CREATE LIBRARY	No equivalent available
CREATE MATERIALIZED VIEW	CREATETAB authority
CREATE PROCEDURE	CREATEIN privilege (use to restrict creation privileges at the schema level)
CREATE SCHEDULER RULE	WLMADM authority
CREATE SCHEMA	DBADM authority
CREATE SEQUENCE	CREATEIN (use to restrict creation privileges at the schema level)
CREATE SYNONYM	CREATEIN (use to restrict creation privileges at the schema level)
CREATE TABLE	CREATETAB authority
CREATE TEMP TABLE	CREATETAB authority
CREATE USER	No equivalent available
CREATE VIEW	CREATEIN (use to restrict creation privileges at the schema level)
MANAGE HARDWARE	No equivalent available
MANAGE SECURITY	SECADM authority
MANAGE SYSTEM	DBADM authority

Table 10. Administration privileges (continued)	
Netezza privilege	**Db2 mapping**
RESTORE	No equivalent available
UNFENCE	CREATE_NOT_FENCED_ROUTINE
VACUUM	No equivalent available

Related tasks

"Migrating users, groups, and privileges from IBM PureData System for Analytics (Netezza) to Db2" on page 42
As part of Netezza to Db2 migration, you must migrate your Netezza users, groups, and privileges.

IBM PureData System for Analytics (Netezza) and Db2 system view compatibility

Netezza system views are not automatically mapped to Db2 system views, but you can manually replace Netezza views with similar Db2 views.

The following table maps some of the most commonly used Netezza system views to their closest Db2 equivalents. Also, if you used _t_* Netezza system tables, use the following table to help determine which Db2 system views to use instead.

Netezza system view	Db2 system view
_v_aggregate	SYSCAT.ROUTINES
_v_environ	SYSIBMADM.ENV_INST_INFO
_v_external	SYSCAT.EXTERNAL_TABLEOPTIONS
_v_extobject	SYSCAT.TABLES
_v_function	SYSCAT.ROUTINES
_v_procedure	SYSCAT.ROUTINES
_v_relation_column	SYSCAT.COLUMNS
_v_relation_column_def	SYSCAT.COLUMNS
_v_schema	SYSCAT.SCHEMAMATA
_v_statistic	SYSSTAT.TABLES
_v_synonym	SYSCAT.TABLES (when querying, specify TYPE ='A', as shown in the following example: `select * from SYSCAT.TABLES where TYPE = 'A'`
_v_sys_columns	SYSCAT.COLUMNS
_v_sys_datatype	SYSCAT.DATATYPES
_v_table	SYSCAT.TABLES
_v_view	SYSCAT.VIEWS

In addition, instead of the Netezza _v_session view, you can use the Db2 MON_GET_CONNECTION table function.

Migrating IBM PureData System for Analytics (Netezza) queries to Db2

Netezza and Db2 queries are generally compatible, but some exceptions apply. You can automatically fix many of the incompatibilities.

Procedure

Use one of the following approaches:

- If your query was generated by a third-party tool, re-create the query by selecting the Db2 connector option in the tool.
- If your query was not generated by a third-party tool, perform the following steps:
  a. Evaluate compatibility by using the Database Conversion Workbench or Database Harmony Profiler. For information about how to obtain and use these tools, see the Database Conversion Workbench page and IBM Database Conversion Workbench (DCW) topic.
  b. If the tool flags incompatibilities that it cannot fix, manually fix them.
  c. Automatically fix the remaining incompatibilities by running the Database Conversion Workbench or Database Harmony Profiler again.

Related concepts

"IBM PureData System for Analytics (Netezza) and Db2 SQL compatibility" on page 73
In most cases, Db2 and Netezza products provides equivalent SQL support. For example, in most cases, a particular Netezza SQL command has an equivalent Db2 SQL statement, which you can use without modifying your code. In a limited number of cases, changes are required.

IBM PureData System for Analytics (Netezza) and Db2 query compatibility

Although Netezza and Db2 queries are generally compatible, you should be aware of certain differences in support.

Following are some of the differences:

Intervals

In a Db2 query, you can use a labeled duration. A labeled duration represents a specific unit of time as expressed by a number (which can be the result of an expression) followed by a duration keyword, such as YEARS, MONTHS, or DAYS. For example, 2 MONTHS is a labeled duration.

Consider the following Netezza examples:

```
select current_timestamp + INTERVAL '1 DAY' FROM FOO
SELECT CURRENT_TIMESTAMP + INTERVAL('1 DAY') FROM FOO
```

The equivalent Db2 example is as follows:

```
SELECT current_timestamp + 1 DAY FROM FOO
```

Isolation levels

A Netezza query supports serializable transaction isolation. A Db2 query supports the following isolation levels:

- The repeatable read (RR) isolation level, for row-organized tables only
- The read stability (RS) isolation level, for row-organized tables only
- The cursor stability (CS) isolation level
- The uncommitted read (UR) isolation level

SELECT statements without FROM clauses

In a Db2 query, the FROM clause is mandatory. For example, the following SELECT statement is not supported:

```
SELECT constant as
alias;
```

If you are referencing a value, use the VALUES statement or use `FROM sysibm.sysdummy1 as`. If you are invoking a procedure, use the CALL statement.

For more details about differences between Netezza and Db2 DML, see "IBM PureData System for Analytics (Netezza) and Db2 SQL compatibility" on page 73.

You can use the Db2 SQL_COMPAT global variable to enable certain Netezza behavior that is not supported by default in a Db2 system. For information, see Compatibility features for Netezza Platform Software (NPS®).

Db2 compatibility for IBM PureData System for Analytics (Netezza) built-in and SQL Extensions toolkit functions

Db2 products provide many of the same functions that are available in IBM PureData System for Analytics (Netezza). In some cases, you must use an alternative, such as a Db2 function with a different name that provides similar support.

The following tables identify the Netezza functions that do not have corresponding Db2 functions with the same names and behaviors. Alternatives to the Netezza functions are mentioned where possible.

Netezza SQL functions

Table 11. Fuzzy string search functions

Netezza function	Db2 alternatives
dle_dst	No alternative is available.
le_dst	No alternative is available.

Table 12. Phonetic matching functions

Netezza function	Db2 alternatives
dbl_mp	No alternative is available.
nysiis	No alternative is available.
pri_mp	No alternative is available.
score_mp	No alternative is available.
sec_mp	No alternative is available.

Table 13. Value functions

Netezza function	Db2 alternatives
current_catalog	Create a session variable.
current_db	Create a session variable.
current_tx_path	Create a session variable.
current_tx_schema	Create a session variable.
current_userid	Create a session variable.
current_useroid	Create a session variable.

Netezza SQL extensions functions

Table 14. Trigonometric functions

Netezza function	Db2 alternatives
pi	Create a user-defined function (UDF).

Table 15. Random number functions

Netezza function	Db2 alternatives
setseed	Pass the seed to the RANDOM function.

Table 16. Numeric functions

Netezza function	Db2 alternatives
dceil	Use the CEIL function with an input type of DOUBLE.
dfloor	Use the FLOOR function with an input type of DOUBLE.
fpow	Use the POW function with an input type of DOUBLE.
numeric_sqrt	Use the SQRT function with an input type of DECFLOAT.
n!	No alternative is available.

Table 17. Binary mathematical functions

Netezza function	Db2 alternatives
int*N*shl	Use two's complement multiplication.
int*N*shr	Use two's complement division.

Table 18. Date and time functions

Netezza function	Db2 alternatives
age (both versions)	A Db2 AGE function is available, but it returns INTEGER.
duration_add	Use a labeled duration or use date, time, or timestamp arithmetic, as documented in Datetime operations and durations.
duration_subtract	Use a labeled duration or use date, time, or timestamp arithmetic, as documented in Datetime operations and durations.
timeofday	Use the VARCHAR_FORMAT function with the CURRENT_TIMESTAMP special register.

Table 19. Character string functions	
Netezza function	**Db2 alternatives**
translate	A Db2 TRANSLATE function is available, but the default order of options for this function is different from the order of options for the Netezza translate function. To enable the Netezza order of options for the Db2 TRANSLATE function, set the **`SQL_COMPAT`** global variable to `'NPS'`.
unichr	No alternative is available.
unicode	No alternative is available.
unicodes	No alternative is available.

Table 20. Conversion functions	
Netezza function	**Db2 alternatives**
hex_to_binary	Use the HEXTORAW function.
hex_to_geometry	No alternative is available.
int_to_string	No alternative is available.
string_to_int	No alternative is available.

Table 21. Miscellaneous non-aggregate functions	
Netezza function	**Db2 alternatives**
get_viewdef	Query the SYSCAT.VIEWS TEXT column.

Table 22. Additional functions	
Netezza function	**Db2 alternatives**
dense_rank	This function is supported only in OLAP queries.
rank	This function is supported only in OLAP queries.
trim	There are differences in more complex usage.

SQL Extensions toolkit functions

Table 23. SQL Extensions toolkit functions		
Category of SQL Extensions toolkit functions	**Specific functions**	**Db2 alternatives**
XML	All functions, for example, IsValidXML	There is no one-to-one mapping between Netezza XML functions and Db2 XML functions. Use the appropriate Db2 XML function.

Table 23. SQL Extensions toolkit functions (continued)		
Category of SQL Extensions toolkit functions	**Specific functions**	**Db2 alternatives**
Data transformation	The compress, compress_nvarchar, decompress, decompress_nvarchar, uudecode, and uuencode functions	No alternatives are available. However, Db2 products use compression on BLU tables by default, so you might not need to compress data at the column level yourself.
	The encrypt, encrypt_nvarchar, decrypt, decrypt_nvarchar, fpe_decrypt, and fpe_encrypt functions	The Db2 databases are encrypted. However, if you want to use a function, consider using the Db2 ENCRYPT, DECRYPT_BIN, and DECRYPT_CHAR functions. For masking, you can use row and column access control (RCAC).
Hashing	The hash_nvarchar function	Using the Db2 HASH function, you can create a sourced function with the name hash_nvarchar.
Text analytics functions	The regexp_extract_all, regexp_extract_all_sp, regexp_extract_sp, and regexp_replace_sp functions	Consider rewriting your code to use the Db2 REGEXP_EXTRACT function.
Array functions	All functions, for example, add_element	Consider using SQL PL array support.
Collection functions	The collection and element_type functions	Consider using SQL PL array support instead.
Miscellaneous functions	The corr, covar_pop, and covar_samp functions	The following Db2 functions are available, with synonyms that correspond to the Netezza names: • CORR is a synonym for the CORRELATION function. • COVAR_POP is a synonym for the COVARIANCE function. • COVAR_SAMP is a synonym for the COVARIANCE_SAMP function.
	The mt_random function	If using the Mersenne Twister pseudorandom number generator is not necessary, use the Db2 RANDOM function.

Migrating IBM PureData System for Analytics (Netezza) routines to Db2

Using IBM tools, you can automatically change much of the source code of your Netezza NZPLSQL routines to Db2 SQL PL.

About this task

If you are moving from Netezza to Db2, you might want to permanently change your NZPLSQL source code to SQL PL, as described in the following procedure. You can then take advantage of any future SQL PL enhancements. However, if you do not want to permanently change your code, you can set the **`SQL_COMPAT`** global variable to `'NPS'` before you compile your NZPLSQL source in Db2. The routine is then sent to an NZPLSQL cross-compiler, which attempts to convert the routine from NZPLSQL to SQL PL before sending it to the SQL PL compiler. Some manual changes might still be needed.

For more information about NZPLSQL support in Db2, see Routines written in NZPLSQL.

Procedure

1. Evaluate compatibility by using the Database Conversion Workbench or Database Harmony Profiler. For information about how to obtain and use these tools, see the Database Conversion Workbench page and IBM Database Conversion Workbench (DCW) topic.
2. If the tool flags NZPLSQL code that it cannot convert to SQL PL, manually change the code.
3. Automatically convert the remaining code to SQL PL by running the Database Conversion Workbench or Database Harmony Profiler again.

IBM PureData System for Analytics (Netezza) and Db2 user-defined extensions compatibility

If your Netezza databases used user-defined functions (UDFs), user-defined aggregates (UDAs), or user-defined shared libraries and you want to run these objects in a Db2 environment, you must create new compiled source and register the objects by using Db2 processes.

For more information about using user-defined extensions with Db2 products, see Creating and managing user-defined extensions (UDXs).

For details about differences between Netezza and Db2 SQL that are related to user-defined extensions, see “IBM PureData System for Analytics (Netezza) and Db2 SQL compatibility” on page 73.

Migrating workload management settings

You cannot migrate your IBM PureData System for Analytics (formerly Netezza) workload management settings to IBM Db2 Warehouse. However, you can take advantage of the workload management capabilities offered by Db2 Warehouse.

PureData System for Analytics offers the following workload management features:

- Guaranteed resource allocation (GRA)
- Prioritized query execution (PQE)
- Scheduler rules
- Short query bias (SQB)

Db2 Warehouse offers customizable adaptive workload management capability that is comparable and that you can use to automatically manage your workload.

Db2 Warehouse uses *service classes*, which are analogous to Netezza resource groups. Db2 Warehouse provides the following pre-defined service classes:

- SYSDEFAULTMAINTENANCECLASS
- SYSDEFAULTSYSTEMCLASS
- SYSDEFAULTUSERCLASS

In addition, you can issue CREATE SERVICE CLASS statements to define your own service classes.

A service class can specify the type of the workloads (mixed, interactive, or batch) for which it is to be used. Each service class can be assigned a resource share and a minimum resource share. A resource share is conceptually similar to a guaranteed resource allocation (GRA). For example, the following statement creates a service class that assigns 2000 resource shares for mixed workloads:

```
CREATE SERVICE CLASS SC1 FOR WORKLOAD TYPE MIXED RESOURCE SHARES 2000
```

You can use a *workload* to group similar work items, to apply thresholds, or to route work to a particular service class. You can use the console to monitor a workload. For more information about query history and workload monitoring, see Query history and workload monitoring. For example, the following statement creates a workload that runs on the service class SC1:

```
CREATE WORKLOAD MONTHLYSALES APPLNAME('monthlyrpt') SERVICE CLASS SC1
```

Use *thresholds* to define and enforce rules on the database to detect and control rogue queries. A threshold is conceptually similar to a scheduler rule. For example:

- Create a threshold for a workload:

  ```
  CREATE THRESHOLD FORCELONGUOW FOR WORKLOAD MONTHLYSALES ACTIVITIES ENFORCEMENT DATABASE WHEN
  UOWTOTALTIME > 10 MINUTES FORCE APPLICATION
  ```

- Create a threshold for a service class:

  ```
  CREATE THRESHOLD BIGQUERIESLONGRUNNINGTIME FOR SERVICE CLASS SC1 ACTIVITIES ENFORCEMENT
  DATABASE WHEN ACTIVITYTOTALTIME > 10 HOURS COLLECT ACTIVITY DATA WITH DETAILS AND VALUES
  ```

- Create a threshold for a database:

  ```
  CREATE THRESHOLD DBMAX1HOURRUNTIME FOR DATABASE BLUDB ENFORCEMENT DATABASE WHEN
  ACTIVITYTOTALTIME > 1 HOUR STOP EXECUTION
  ```

Use *session priority* to influence the scheduling of work within a service class. Session priority is conceptually similar to prioritized query execution in PureData System for Analytics.

- Specify the priority of all work from a particular user or application by using the PRIORITY property of a WORKLOAD object. This method is similar to specifying the priority on a resource group. For example, to identify all work that is submitted by user NEWTON as high priority, you can create the following WORKLOAD object:

  ```
  CREATE WORKLOAD PAYROLL SESSION_USER('NEWTON') PRIORITY HIGH
  ```

- Specify the priority of a particular connection by using the WLM_SET_SESSION_PRIORITY stored procedure. This method is similar to specifying a priority for an application by using the **nzsession** command or the NzAdmin tool. For example, to reduce the priority of an application with handle 2361, use the following stored procedure call:

  ```
  call SYSPROC.WLM_SET_SESSION_PRIORITY(2361, 'LOW')
  ```

Validating migration of IBM PureData System for Analytics (Netezza) to Db2
After migrating your Netezza system to Db2, you should validate your new system.

Procedure

Perform the following steps:

- Ensure that your Netezza and Db2 objects are comparable, based on the conversions that you performed by using the Database Conversion Workbench or Database Harmony Profiler or by making manual changes.
- Test applications that contain converted SQL, such as by using the web console. Ensure that the results for your Netezza workloads and Db2 workloads are equivalent. Ensure that the performance of your Netezza workloads and the performance of your Db2 workloads are comparable.

IBM PureData System for Analytics (Netezza) and Db2 compatibility

A IBM PureData System for Analytics (Netezza) system and a Db2 system are highly compatible. Use the information in this section to learn about the differences between them to make it easier to migrate your Netezza system to a Db2 system and adapt to using it afterward.

Before reading this section, see "Migrating from IBM PureData System for Analytics (Netezza) to a Db2 system" on page 39 to learn about the overall migration process.

For more information about Db2 compatibility features, see Compatibility features.

IBM PureData System for Analytics (Netezza) and Db2 data type compatibility

Netezza and Db2 databases share a large common set of data types. To learn about data type incompatibilities and how to manage them when migrating from a Netezza database to a Db2 database, review the following topics.

You can use tools such as IBM Database Harmony Profiler to assess and migrate data definitions and SQL commands. Database Harmony Profiler can help to identify possible conversion considerations and convert SQL commands. During conversion, Database Harmony Profiler maps Netezza data types to the equivalent Db2 types. These changes typically do not alter the nature of the data nor the queries that process data that is stored in columns of those types. Database Harmony Profiler and similar tools cannot automatically make some data type changes, so manual intervention is required.

IBM PureData System for Analytics (Netezza) non-internal data types that are unsupported in Db2 products

As documented in this topic, some non-internal Netezza data types do not have corresponding Db2 data types. For information about Db2 support for internal Netezza data types, such as ROWID, see "IBM PureData System for Analytics (Netezza) and Db2 differences in internal data types " on page 57.

The following table lists the non-internal Netezza data types that do not have corresponding Db2 data types.

Table 24. Unsupported Netezza data types

Netezza data type	Db2 data type alternatives
BOOL	Use one of the following approaches: • Use the built-in BOOLEAN type. IBM Database Harmony Profiler automatically converts BOOL to BOOLEAN. • Create a type by using the CREATE TYPE (distinct) statement, as follows: `CREATE TYPE BOOL AS BOOLEAN WITH WEAK TYPE RULES`
BYTEINT (alias INT1)	Use one of the following approaches: • Use the built-in SMALLINT (SHORT) type. IBM Database Harmony Profiler automatically converts BYTEINT to SMALLINT. • Create a type by using the CREATE TYPE (distinct) statement, as follows: `CREATE TYPE BYTEINT AS SMALLINT WITH WEAK TYPE RULES` Test your queries to ensure that the data type change does not affect results.

Table 24. Unsupported Netezza data types (continued)	
Netezza data type	**Db2 data type alternatives**
INTERVAL (alias TIMESPAN)	Consider using the DECIMAL data type to store appropriate date, time, and time stamp duration values. Alternatively, replace the INTERVAL data type with the INTEGER or BIGINT data type, and store the value in units of seconds. Additional work is required to convert the INTEGER value to days, minutes, seconds, etc.
TIME WITH TIMEZONE (alias TIMETZ)	Store time zone information in a new separate column of the table. You can use an additional column to store the time zone component and include the value of this column in all time calculations. Alternatively, you can keep the database server on Coordinated Universal Time (UTC) and convert all time-zoned values to UTC.

IBM PureData System for Analytics (Netezza) and Db2 differences in non-internal data types

Some non-internal data types that are supported in both Netezza and Db2 products are supported differently, as described in this topic. For information about Db2 support for internal Netezza data types, such as ROWID, see "IBM PureData System for Analytics (Netezza) and Db2 differences in internal data types " on page 57.

The following table describes differences in support and recommendations for migrating to a Db2 product if the differences result in problems. IBM Database Harmony Profiler maps Netezza data types to Db2 data types and, in some cases, can perform the changes that are recommended in the table. Carefully review the Database Harmony Profiler assessment report to understand the conversions and to learn about differences that might require some changes to queries and reporting. Data truncation might occur as a result of conversion.

Table 25. Netezza and Db2 data type differences		
Data type	**Difference**	**Recommendations for migrating**
CHAR	The maximum length for the Db2 data type is 255 bytes. The maximum length for the Netezza data type is 64,000 bytes.	Check whether you can redefine the length of the data type to comply with the Db2 limit. If not, use the VARCHAR data type. Database Harmony Profiler converts CHAR(*n*) values where *n* is 255 - 32,592 to VARCHAR(*n*). Database Harmony Profiler also converts CHAR(*n*) values where *n* is greater than 32,592 to VARCHAR(32592).
VARCHAR	The maximum length for the Db2 data type is 32,592 bytes. The maximum length for the Netezza data type is 64,000 bytes.	Check whether you can redefine the length of the data type to comply with the Db2 limit. If not, use the CLOB data type in a row-organized table. Database Harmony Profiler converts CHAR(*n*) values where *n* is greater than 32,592 to VARCHAR(32592).

Table 25. Netezza and Db2 data type differences (continued)		
Data type	**Difference**	**Recommendations for migrating**
NCHAR	The maximum length for the Db2 data type is 63 bytes. The maximum length for the Netezza data type is 16,000 bytes.	Check whether you can redefine the length of the data type to comply with the Db2 limit. If not, use the NVARCHAR data type. Database Harmony Profiler converts NCHAR(*n*) values where *n* is 64 - 8148 to NVARCHAR(*n*). Database Harmony Profiler also converts NCHAR(*n*) values where *n* is greater than 8148 to NVARCHAR(8148).
NVARCHAR	The maximum length for the Db2 data type is 8148 bytes. The maximum length for the Netezza data type is 16,000 bytes.	Check whether you can redefine the length of the data type to comply with the Db2 limit. If not, use an NCLOB data type in a row-organized table. Database Harmony Profiler converts NVARCHAR(*n*) values where *n* is greater than 8148 to NVARCHAR(8148).
VARBINARY	The maximum length for the Db2 data type is 32,592 bytes. The maximum length for the Netezza VARBINARY and ST_GEOMETRY data types is 64,000 bytes.	Check whether you can redefine the length of the data type to comply with the Db2 limit. If not, use a BLOB data type in a row-organized table. Database Harmony Profiler converts VARBINARY(*n*) values where *n* is greater than 32592 to VARBINARY(32592).
DECIMAL (NUMERIC)	The maximum precision of the Db2 data type is 31 digits. The maximum precision of the Netezza data type is 38 digits.	Check whether you can redefine the precision of the data type to comply with the Db2 limit. If not, consider using the DECFLOAT data type, which has a maximum precision of 34 digits. Database Harmony Profiler converts DECIMAL(*p*,*s*) values where *p* is greater than 31 to DECIMAL(31,*s*). Similarly, Database Harmony Profiler converts NUMERIC(*p*,*s*) values where *p* is greater than 31 to NUMERIC(31,*s*).
Floating-point types	If you specify a precision (n) for a Db2 floating-point type, SQL standard precision ranges apply. Netezza does not use the standard precision ranges.	If you specify a precision, ensure that you are using the appropriate type (REAL or DOUBLE). It is preferable to change Netezza types with explicit precisions, as follows: • If you were using the Netezza DOUBLE PRECISION or FLOAT(15) type, change to the Db2 FLOAT type. • If you were using the Netezza REAL or FLOAT(6) type, change to the Db2 REAL type.

Table 25. Netezza and Db2 data type differences (continued)		
Data type	**Difference**	**Recommendations for migrating**
TIME	The Db2 TIME data type does not support the microseconds portion of the Netezza TIME data type (for example, 999999 in the value 23:59:59.999999).	Check whether you require microsecond-level precision and can therefore use the TIME data type. If not, use the TIMESTAMP data type.

IBM PureData System for Analytics (Netezza) and Db2 differences in internal data types
Netezza internal data types, such as ROWID, are supported differently or are unsupported in Db2.

Netezza data type	Netezza column name	Db2 support
rowid	rowid	You can use the ROWID pseudocolumn, which has a VARCHAR (16) FOR BIT DATA data type. ROWID is an alternative to the RID_BIT() function.
dataslice	datasliceid	You can use the DATASLICEID pseudocolumn. This column contains the database partition number for a row. The value can be 0.
transaction ID	createxid, deletexid	If required, you can use an identity column (hidden, if need be) in your table to provide a unique row identifier.
Other types		Other internal data types are not supported. If you need help migrating these to Db2, contact IBM.

IBM PureData System for Analytics (Netezza) and Db2 backup and restore compatibility
You cannot use an IBM PureData System for Analytics (Netezza) backup to restore to a Db2 database.

IBM PureData System for Analytics (Netezza) and Db2 CLI command compatibility
Most PureData System for Analytics (Netezza) CLI commands (**nz*** commands) do not have corresponding Db2 commands. In some cases, there's a different mechanism. In other cases, no equivalent is necessary.

Table 26. PureData System for Analytics (Netezza) command support	
Netezza CLI command	**Db2 support**
nzbackup	
nzcontents	There is no Db2 equivalent.
nzconvert	For information about how to load data in a Db2 environment, see Loading your data.
nzds	The Db2 environment does not use data slices.
nzevent	Use a Db2 web console.

Table 26. PureData System for Analytics (Netezza) command support (continued)	
Netezza CLI command	**Db2 support**
nzhistcleanupdb	The Db2 environment does not use a history database. To collect and view historical data about database usage, use the Db2 monitoring capabilities.
nzhistcreatedb	The Db2 environment does not use a history database. To collect and view historical data about database usage, use the Db2 monitoring capabilities.
nzhostbackup	
nzhostrestore	
nzhw	There is no Db2 equivalent. In the Db2 managed service environment, hardware is managed for you.
nzload	For information about how to load data in a Db2 environment, see Loading your data.
nzodbcsql	For information about defining and testing ODBC connections, see Connecting programmatically with ODBC or CLI.
nzpassword	Use a Db2 web console.
nzreclaim	For column-organized tables, reorganization is done automatically. For row-organized tables, use the REORG TABLE statement.
nzrestore	
nzrev	Use a Db2 web console or CLPPlus.
nzsession	Use a Db2 web console to obtain a list of connected applications.
nzspupart	The Db2 environment does not use SPUs.
nzsql	Use the **dbsql** command. For more information, see Overview of the support tools.
nzstart	
nzstate	Use a Db2 web console.
nzstats	Use a Db2 web console.
nzstop	
nzsystem	Use a Db2 web console.

IBM PureData System for Analytics (Netezza) and IBM ODBC, JDBC, OLE DB, and .NET compatibility
The driver package for the Db2 instance contains multiple drivers that you can use to connect client applications to the Db2 database. The package includes drivers for ODBC, JDBC, .NET, and OLE DB. You can also connect programmatically by using ODBC, JDBC, or .NET.For information about connecting an application to a Db2 database, see Connecting to your Db2 database.

Query history and workload monitoring

You can enable query history for monitoring and analysis of workloads by using the IBM Db2 Warehouse console.

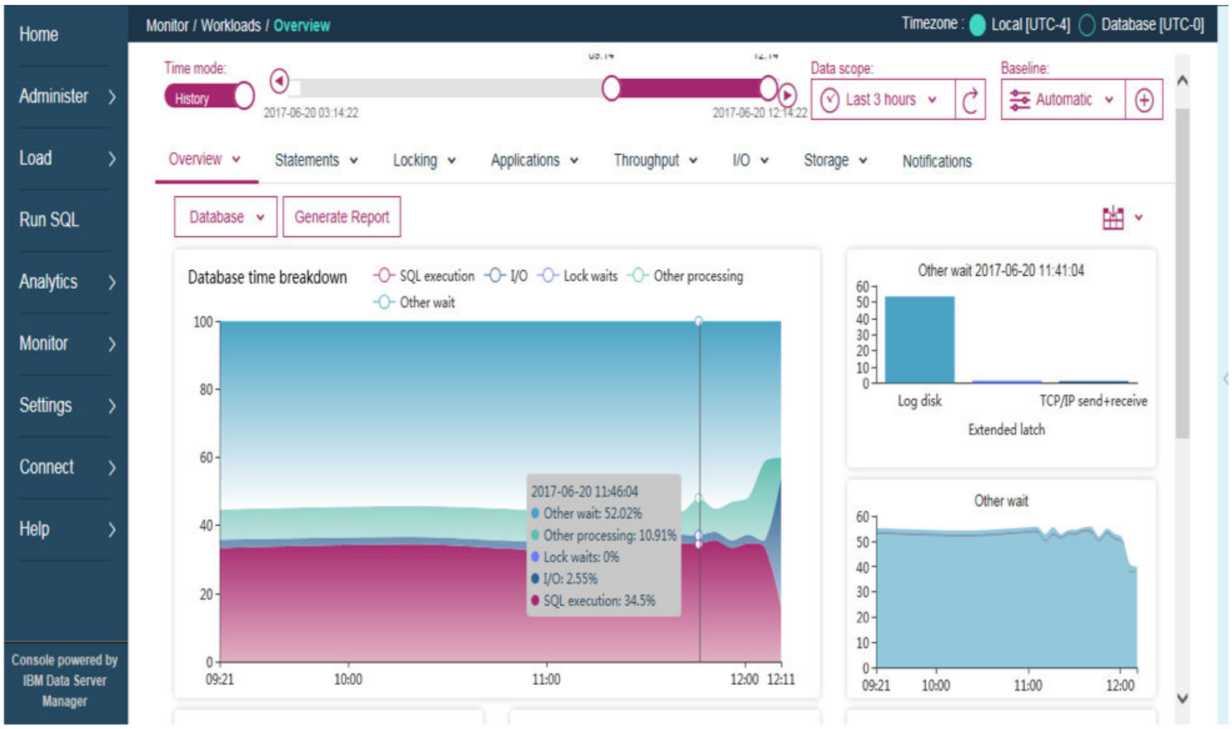
Procedure

1. Log in to the Db2 Warehouse web console as `bluadmin`.

   ```
   https://<virtual_ip>:8443/console
   ```

2. Select **Monitor** > **Workloads**
3. For **Time mode:**, select **History**.

   This starts the collection of query history.

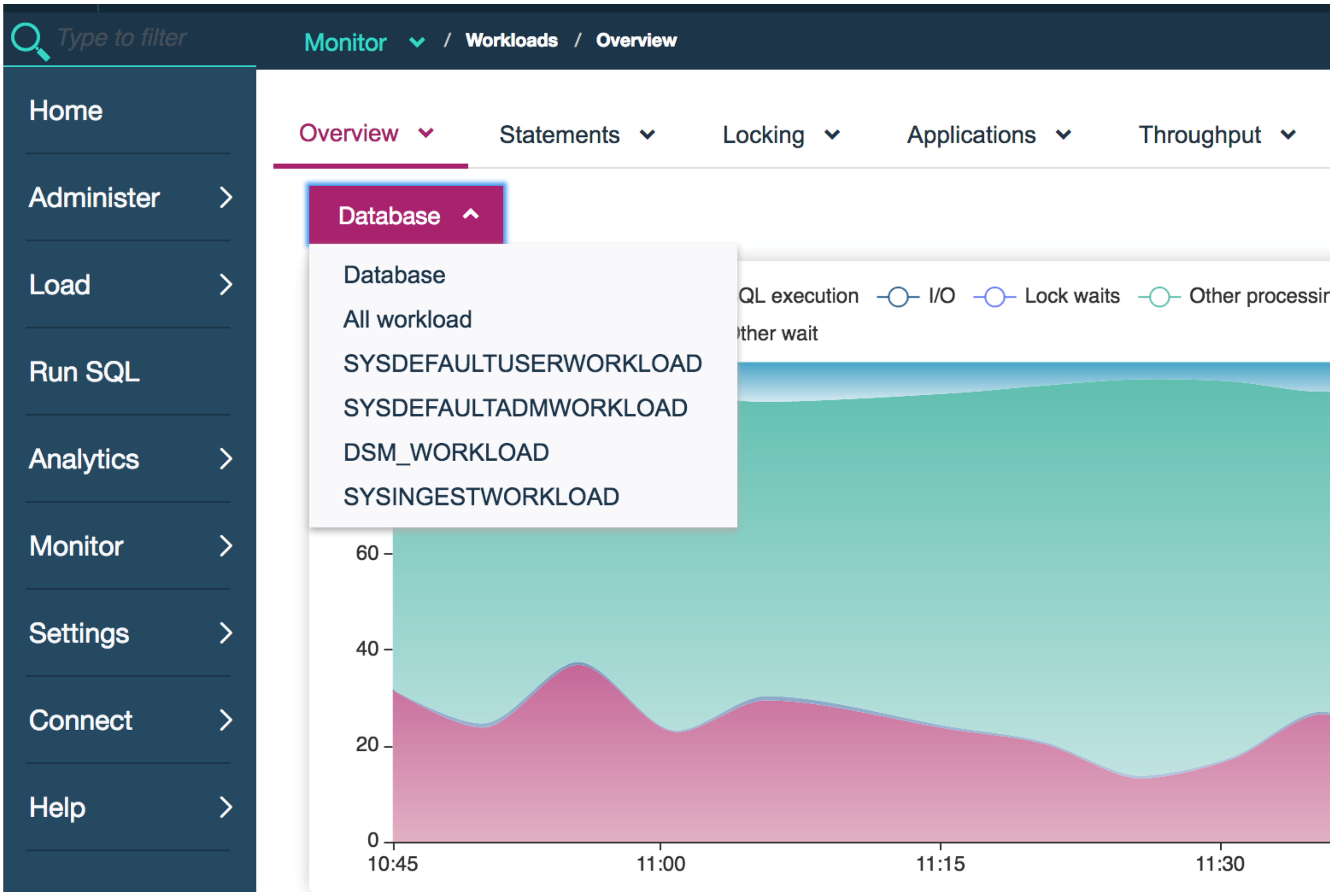
   When query history monitoring is enabled in the Db2 Warehouse web console, default monitoring profile properties are pre-defined so that you don't have to configure the database activity monitoring.

   Note: Query history is stored in the IBMOTS.SQL_DIM table. The IBM_RTMON_EVMON.EVENT_ACTIVITY table stores the following additional information:

   - `planid`
   - `user id`
   - `application name`
   - `service class`
   - `workload id`
   - `estimated cost`
   - and more.

   You can join these tables to get detailed information about a query.

4. To monitor a specific workload, click the **Database** drop-down menu and choose the workload.



IBM PureData System for Analytics (Netezza) configuration choices that impact migration to Db2products

Some PureData System for Analytics (Netezza) configuration choices can result in compatibility issues when you migrate to a Db2 product.

Some of these choices are as follows:

Multiple schemas

If you configured your Netezza system to support multiple schemas in multiple databases, migrate by using one of the following approaches. For each approach, use the **`db_migrate`** command or Database Harmony Profile tool.

- If there are no conflicting schemas across the databases, drop the database identifiers, and then migrate to a single BLUDB database.
- If there are conflicting schemas across the databases, merge each database identifier and each schema into a single schema, and then migrate to a single BLUDB database.

In each case, the schemas are moved to the BLUDB database when you migrate. You can perform the above with the **`-prefixSchema`** parameter of the **`db_migrate`** utility.

Case setting for identifiers

In a Db2 system, lowercase identifiers, such as for table names, are converted to uppercase. This is also the default behavior in a Netezza system. However, if you used the **`nzinitsystem -lowercase`** or **`nzconvertsyscase`** command to force the Netezza system to use lowercase ...

Time zone

Netezza uses UTC by default, but you might have configured Netezza to use a different time zone. The Db2 products also use UTC by default. If you require time stamps, times, or dates to use a different time zone, use the TO_UTC_TIMESTAMP or TIMEZONE function.

Encoding and collation

A Db2 system uses UTF-8 encoding with IDENTITY collation. A Netezza system uses Latin-9 encoding for single-byte characters and UTF-8 encoding for multibyte characters, with a default collation of BINARY. The differences between systems can result in data expansion when you load data and differences in how character data is ordered.

In-database analytics compatibility

In most cases, provide the same analytic functions that are available in IBM PureData System for Analytics (Netezza). For example, in most cases, a particular Netezza analytic function has an equivalent analytic function. In some cases, changes are required.

Compatibility matrix for IBM PureData System for Analytics (Netezza) built-in and analytic functions

The following tables identify the Netezza functions of the corresponding category and, if ported to , the corresponding functions. The tables also show changes and limitations.

The categories and the corresponding functions are sorted alphabetically.

- “Association rules” on page 61
- “Classification” on page 62
- “Clustering” on page 62
- “Column properties” on page 63
- “Data transformation” on page 63
- “Diagnostic measures” on page 63
- “Discretization” on page 64
- “Model management” on page 64
- “Probability distributions” on page 65
- “Quantiles and outliers” on page 69
- “Regression” on page 69
- “Sampling” on page 70
- “Sequential patterns” on page 70
- “Statistics” on page 70
- “Timeseries” on page 72
- “Utilities” on page 72

Association rules

Table 27. Functions for association rules

Netezza function	function	Changes and limitations
ARULE	ASSOCRULES	New routine with new parameters. Migration required.
PREDICT_ARULE	PREDICT_ASSOCRULES	New routine with new parameters. Migration required.
PRINT_ARULE	PRINT_MODEL	New routine with new parameters. Migration required.
VERIFY_ARULE	No alternative is available.	Not applicable.

Back to top ↑

Classification

Table 28. Functions for classification

Netezza function	function	Changes and limitations
CROSS_VALIDATION	No alternative is available.	Not applicable.
DECTREE	DECTREE	None.
GROW_DECTREE	GROW_DECTREE	None.
KNN	KNN	New sampling: 10000 by default instead of unlimited sampling. Additional parameters: **maxsize** and **randseed**.
NAIVEBAYES	NAIVEBAYES	None.
PERCENTAGE_SPLIT	No alternative is available.	Use classification algorithms and quality measures instead.
PMML_DECTREE	PMML_MODEL	None.
PMML_NAIVEBAYES	No alternative is available.	Not applicable.
PREDICT_DECTREE	PREDICT_DECTREE	None.
PREDICT_KNN	PREDICT_KNN	None.
PREDICT_NAIVEBAYES	PREDICT_NAIVEBAYES	None.
PRINT_DECTREE	PRINT_MODEL	Use PRINT_MODEL instead.
PRUNE_DECTREE	PRUNE_DECTREE	None.
TRAIN_TEST	No alternative is available.	Use classification algorithms and quality measures instead.

Back to top ↑

Clustering

Table 29. Functions for clustering

Netezza function	function	Changes and limitations
DIVCLUSTER	No alternative is available.	Not applicable.
KMEANS	KMEANS	The **mahalanobis** parameter values is not available.
PREDICT_DIVCLUSTER	No alternative is available.	Not applicable.
PREDICT_KMEANS	PREDICT_KMEANS	Not all parameter values available.
PREDICT_TWOSTEP	No alternative is available.	This stored procedure is implemented using Apache Spark, and can be used only on a system for which Spark capability is enabled.
PRINT_KMEANS	PRINT_MODEL	Use PRINT_MODEL and the respective parameters for KMEANS instead.

Table 29. Functions for clustering (continued)		
Netezza function	**function**	**Changes and limitations**
PRINT_TWOSTEP	PRINT_MODEL	New routine with new parameters. Migration required.
SET_CLUSTERNAME	No alternative is available.	Not applicable.
TWOSTEP	No alternative is available.	None.

Back to top ↑

Column properties

Table 30. Functions for column properties		
Netezza function	**function**	**Changes and limitations**
COLUMN_PROPERTIES	COLUMN_PROPERTIES	None.
GET_COLUMN_LIST	GET_COLUMN_LIST	None.
SET_COLUMN_PROPERTIES	SET_COLUMN_PROPERTIES	None.

Back to top ↑

Data transformation

Table 31. Functions for data transformation		
Netezza function	**function**	**Changes and limitations**
IMPUTE_DATA	IMPUTE_DATA	None.
PCA	No alternative is available.	Not applicable.
PROJECT_PCA	No alternative is available.	Not applicable.
SPLIT_DATA	SPLIT_DATA	None.
STD_NORM	STD_NORM	None.

Back to top ↑

Diagnostic measures

Table 32. Functions for diagnostic measures		
Netezza function	**function**	**Changes and limitations**
ACC	ACC	None.
CERROR	CERROR	None.
CMATRIX_ACC	CMATRIX_ACC	None.
CMATRIX_STATS	CMATRIX_STATS	None.
CMATRIX_WACC	CMATRIX_WACC	None.
CONFUSION_MATRIX	CONFUSION_MATRIX	None.
FMEASURE	FMEASURE	None.
FPR	FPR	None.

Table 32. Functions for diagnostic measures (continued)		
Netezza function	**function**	**Changes and limitations**
MAE	MAE	None.
MSE	MSE	None.
PPV	PPV	None.
RAE	RAE	None.
RSE	RSE	None.
TPR	TPR	None.
WACC	WACC	None.

Back to top ↑

Discretization

Table 33. Functions for discretization and moments		
Netezza function	**function**	**Changes and limitations**
No Netezza function is available.	AGGDISC	None.
APPLY_DISC	APPLY_DISC	None.
EFDISC	EFDISC	None.
EMDISC	EMDISC	None.
EWDISC	EWDISC	None.
EWDISC_NICE	EWDISC_NICE	None.

Back to top ↑

Model management

Table 34. Functions for model management		
Netezza function	**function**	**Changes and limitations**
ALTER_MODEL	ALTER_MODEL	None.
CLEANUP	CLEANUP	None.
COPY_MODEL	COPY_MODEL	None.
DROP_ALL_MODELS	DROP_ALL_MODELS	None.
DROP_MODEL	DROP_MODEL	None.
EXPORT_MODEL	EXPORT_MODEL	None.
EXPORT_PMML	EXPORT_PMML	None.
GRANT_MODEL	GRANT_MODEL	None.
IMPORT_MODEL	IMPORT_MODEL	None.
INITIALIZE	INITIALIZE	Only needed on a system.
IS_INITIALIZED	IS_INITIALIZED	Only needed on a system.

Table 34. Functions for model management (continued)		
Netezza function	**function**	**Changes and limitations**
LIST_COLPROPS	LIST_COLPROPS	None.
LIST_COMPONENTS	LIST_COMPONENTS	None.
LIST_MODELS	LIST_MODELS	None.
LIST_PARAMS	LIST_PARAMS	None.
LIST_PRIVILEGES	LIST_PRIVILEGES	None.
METADATA_ANALYZE	No alternative is available.	Not applicable.
MIGRATE_MODEL	No alternative is available.	Not required on Db2 systems.
MODEL_EXISTS	MODEL_EXISTS	None.
PMML_MODEL	PMML_MODEL	None.
PRINT_MODEL	PRINT_MODEL	None.
REGISTER_MODEL	No alternative is available.	Not required on Db2 systems.
REVOKE_MODEL	REVOKE_MODEL	None.

Back to top ↑

Probability distributions

Table 35. Functions for probability distributions		
Netezza function	**function**	**Changes and limitations**
CUMULATIVE	No alternative is available.	Not applicable.
DBERN	DBERN	None.
DBETA	DBETA	None.
DBINOM	DBINOM	None.
DCAUCHY	DCAUCHY	None.
DCHISQ	DCHISQ	None.
DENSITY	DENSITY	None.
DEXP	DEXP	None.
DF	DF	None.
DFISK	DFISK	None.
DGAMMA	DGAMMA	None.
DGEOM	DGEOM	None.
DHYPER	DHYPER	None.
DLNORM	DLNORM	None.
DLOGIS	DLOGIS	None.
DMWW	DMWW	None.
DNBINOM	DNBINOM	None.

Table 35. Functions for probability distributions (continued)

Netezza function	function	Changes and limitations
DNORM	DNORM	None.
DNORM3P	DNORM3P	None.
DPOIS	DPOIS	None.
DT	DT	None.
DUNIF	DUNIF	None.
DWALD	DWALD	None.
DWEIBULL	DWEIBULL	None.
DWILCOX	DWILCOX	None.
PBERN	PBERN	None.
PBERN_H	PBERN_H	None.
PBETA	PBETA	None.
PBETA_H	PBETA_H	None.
PBINOM	PBINOM	None.
PBINOM_H	PBINOM_H	None.
PCAUCHY	PCAUCHY	None.
PCAUCHY_H	PCAUCHY_H	None.
PCHISQ	PCHISQ	None.
PCHISQ_H	PCHISQ_H	None.
PCHISQ_S	PCHISQ_S	None.
PEXP	PEXP	None.
PEXP_H	PEXP_H	None.
PF	PF	None.
PF_H	PF_H	None.
PFISK	PFISK	None.
PFISK_H	PFISK_H	None.
PGAMMA	PGAMMA	None.
PGAMMA_H	PGAMMA_H	None.
PGEOM	PGEOM	None.
PGEOM_H	PGEOM_H	None.
PHYPER	PHYPER	None.
PHYPER_H	PHYPER_H	None.
PLNORM	PLNORM	None.
PLNORM_H	PLNORM_H	None.
PLOGIS	PLOGIS	None.

Table 35. Functions for probability distributions (continued)		
Netezza function	**function**	**Changes and limitations**
PLOGIS_H	PLOGIS_H	None.
PMWW	PMWW	None.
PMWW_H	PMWW_H	None.
PNBINOM	PNBINOM	None.
PNBINOM_H	PNBINOM_H	None.
PNORM	PNORM	None.
PNORM3P	PNORM3P	None.
PNORM_H	PNORM_H	None.
PPOINT	PPOINT	None.
PPOIS	PPOIS	None.
PPOIS_H	PPOIS_H	None.
PT	PT	None.
PT_H	PT_H	None.
PUNIF	PUNIF	None.
PUNIF_H	PUNIF_H	None.
PWALD	PWALD	None.
PWALD_H	PWALD_H	None.
PWEIBULL	PWEIBULL	None.
PWEIBULL_H	PWEIBULL_H	None.
PWILCOX	PWILCOX	None.
PWILCOX_H	PWILCOX_H	None.
QBERN	QBERN	None.
QBERN_H	QBERN_H	None.
QBETA	QBETA	None.
QBETA_H	QBETA_H	None.
QBINOM	QBINOM	None.
QBINOM_H	QBINOM_H	None.
QCAUCHY	QCAUCHY	None.
QCAUCHY_H	QCAUCHY_H	None.
QCHISQ	QCHISQ	None.
QCHISQ_H	QCHISQ_H	None.
QEXP	QEXP	None.
QEXP_H	QEXP_H	None.
QF	QF	None.

Table 35. Functions for probability distributions (continued)		
Netezza function	**function**	**Changes and limitations**
QF_H	QF_H	None.
QFISK	QFISK	None.
QFISK_H	QFISK_H	None.
QGAMMA	QGAMMA	None.
QGAMMA_H	QGAMMA_H	None.
QGEOM	QGEOM	None.
QGEOM_H	QGEOM_H	None.
QHYPER	QHYPER	None.
QHYPER_H	QHYPER_H	None.
QLNORM	QLNORM	None.
QLNORM_H	QLNORM_H	None.
QLOGIS	QLOGIS	None.
QLOGIS_H	QLOGIS_H	None.
QMWW	QMWW	None.
QMWW_H	QMWW_H	None.
QNBINOM	QNBINOM	None.
QNBINOM_H	QNBINOM_H	None.
QNORM	QNORM	None.
QNORM3P	QNORM3P	None.
QNORM_H	QNORM_H	None.
QPOIS	QPOIS	None.
QPOIS_H	QPOIS_H	None.
QT	QT	None.
QT_H	QT_H	None.
QUNIF	QUNIF	None.
QUNIF_H	QUNIF_H	None.
QWALD	QWALD	None.
QWALD_H	QWALD_H	None.
QWEIBULL	QWEIBULL	None.
QWEIBULL_H	QWEIBULL_H	None.
QWILCOX	QWILCOX	None.
QWILCOX_H	QWILCOX_H	None.

Back to top ↑

Quantiles and outliers

Table 36. Functions for quantiles and outliers		
Netezza function	**function**	**Changes and limitations**
IQR	No alternative is available.	Not applicable.
MEDIAN	No alternative is available.	Not applicable.
MEDIAN_DISC	No alternative is available.	Not applicable.
OUTLIERS	No alternative is available.	Not applicable.
QUANTILE	No alternative is available.	Not applicable.
QUANTILE_DISC	No alternative is available.	Not applicable.
QUARTILE	No alternative is available.	Not applicable.
QUARTILE_DISC	No alternative is available.	Not applicable.

Back to top ↑

Regression

Table 37. Functions for regression		
Netezza function	**function**	**Changes and limitations**
BTBNET_GROW	No alternative is available.	Not applicable.
GLM	No alternative is available.	None.
GROW_REGTREE	GROW_REGTREE	None.
LINEAR_REGRESSION	LINEAR_REGRESSION	Not available on Linux on IBM z Systems.
MTBNET_DIFF	No alternative is available.	Not applicable.
MTBNET_GROW	No alternative is available.	Not applicable.
PREDICT_GLM	No alternative is available.	None.
PREDICT_LINEAR_REGRESSION	PREDICT_LINEAR_REGRESSION	Not available on Linux on IBM z Systems.
PREDICT_REGTREE	PREDICT_REGTREE	None.
PRINT_GLM	No alternative is available.	Not applicable.
PRINT_REGTREE	PRINT_MODEL	Use PRINT_MODEL instead.
PRUNE_REGTREE	PRUNE_REGTREE	None.
REGTREE	REGTREE	None.
TANET_APPLY	No alternative is available.	Not applicable.
TANET_CLASSAPPLY	No alternative is available.	Not applicable.
TANET_GROW	No alternative is available.	Not applicable.
TBNET1G	No alternative is available.	Not applicable.
TBNET1G2P	No alternative is available.	Not applicable.
TBNET2G	No alternative is available.	Not applicable.

Table 37. Functions for regression (continued)

Netezza function	function	Changes and limitations
TBNET_APPLY	No alternative is available.	Not applicable.
TBNET_GROW	No alternative is available.	Not applicable.

Back to top ↑

Sampling

Table 38. Functions for sampling

Netezza function	function	Changes and limitations
RANDOM_SAMPLE	RANDOM_SAMPLE	None.

Back to top ↑

Sequential patterns

Table 39. Functions for sequential patterns

Netezza function	function	Changes and limitations
PREDICT_SEQRULES	PREDICT_SEQRULES	None.
PRINT_SEQRULES	PRINT_MODEL	Use PRINT_MODEL instead.
PRUNE_SEQRULES	PRUNE_SEQRULES	None.
SEQRULES	SEQRULES	None.

Back to top ↑

Statistics

Table 40. Functions for statistics

Netezza function	function	Changes and limitations
ANOVA_CRD_TEST	ANOVA_CRD_TEST	None.
ANOVA_RBD_TEST	ANOVA_RBD_TEST	None.
BITABLE	No alternative is available.	Not applicable.
CHISQ_TEST	CHISQ_TEST	None.
CHISQ_TEST_AGG	CHISQ_TEST_AGG	None.
CHISQ_TEST_S_AGG	CHISQ_TEST_S_AGG	None.
COL2TRCV_MANOVA_ONE_WAY_TEST	No alternative is available.	Not applicable.
COL2TRCV_MANOVA_TWO_WAY_TEST	No alternative is available.	Not applicable.
DROP_SUMMARY1000	DROP_SUMMARY1000	None.
HIST	No alternative is available.	Not applicable.
JOINT_ENTROPY	JOINT_ENTROPY	None.

Table 40. Functions for statistics (continued)		
Netezza function	**function**	**Changes and limitations**
KURTOSIS_AGG	KURTOSIS_AGG	None.
LDF_MANOVA_ONE_WAY_TEST	No alternative is available.	Not applicable.
LDF_MANOVA_TWO_WAY_TEST	No alternative is available.	Not applicable.
MANOVA_ONE_WAY_TEST	No alternative is available.	Not applicable.
MANOVA_TWO_WAY_TEST	No alternative is available.	Not applicable.
MOMENTS	MOMENTS	None.
MUTUALINFO	MUTUALINFO	None.
MUTUALINFO_AGG	MUTUALINFO_AGG	None.
MWW_TEST	MWW_TEST	None.
No Netezza function is available.	NUMERIC_SUMMARY	None.
PRINT_MANOVA_ONE_WAY_TEST	No alternative is available.	Not applicable.
PRINT_MANOVA_TWO_WAY_TEST	No alternative is available.	Not applicable.
SKEWNESS_AGG	SKEWNESS_AGG	None.
SPEARMAN_CORR	SPEARMAN_CORR	None.
SPEARMAN_CORR_S	No alternative is available.	Not applicable.
SUMMARY1000	SUMMARY1000	The **talk** parameter is deprecated and has no action.
SUMMARY1000CHAR	SUMMARY1000CHAR	The **talk** parameter is deprecated and has no action.
SUMMARY1000DATE	SUMMARY1000DATE	The **talk** parameter is deprecated and has no action.
SUMMARY1000INTERVAL	No alternative is available.	Not applicable.
SUMMARY1000NUM	SUMMARY1000NUM	The **talk** parameter is deprecated and has no action.
SUMMARY1000TIME	SUMMARY1000TIME	The **talk** parameter is deprecated and has no action.
SUMMARY1000TIMESTAMP	SUMMARY1000TIMESTAMP	The **talk** parameter is deprecated and has no action.
T_LS_TEST	T_LS_TEST	The output table has a different format.
T_LS_TEST_S_AGG	T_LS_TEST_S_AGG	None.
T_ME_TEST	T_ME_TEST	The output table has a different format.
T_ME_TEST_S_AGG	T_ME_TEST_S_AGG	None.
T_PMD_TEST	T_PMD_TEST	The output table has a different format.

Table 40. Functions for statistics (continued)		
Netezza function	**function**	**Changes and limitations**
T_PMD_TEST_S_AGG	No alternative is available.	Not applicable.
T_TEST_AGG	T_TEST_AGG	The output table has a different format.
T_TEST_S_AGG	T_TEST_S_AGG	None.
No Netezza function is available.	SPLIT_TEST_S	None.
T_UMD_TEST	T_UMD_TEST	The output table has a different format.
UNITABLE	No alternative is available.	Not applicable.
WILCOXON_TEST	WILCOXON_TEST	None.

Back to top ↑

Timeseries

Table 41. Functions for timeseries		
Netezza function	**function**	**Changes and limitations**
PRINT_TIMESERIES	No alternative is available.	Not applicable.
TIMESERIES	No alternative is available.	Not applicable.

Back to top ↑

Utilities

Table 42. Functions for utilities		
Netezza function	**function**	**Changes and limitations**
msghelp	No alternative is available.	Not applicable.
_sp_utl_dropAllAggregates	No alternative is available.	Not applicable.
_sp_utl_dropAllFunctions	No alternative is available.	Not applicable.
_sp_utl_dropAllLike	No alternative is available.	Not applicable.
_sp_utl_dropAllProcedures	No alternative is available.	Not applicable.
_sp_utl_dropAllUDX	No alternative is available.	Not applicable.
_sp_utl_justExecute	No alternative is available.	Not applicable.
DROP_TABLE	No alternative is available.	Not applicable.
_sp_utl_aggregateExists	No alternative is available.	Not applicable.
_sp_utl_columnContainsNulls	No alternative is available.	Not applicable.
_sp_utl_columnExists	No alternative is available.	Not applicable.
_sp_utl_columnIsId	No alternative is available.	Not applicable.
_sp_utl_columnIsNumeric	No alternative is available.	Not applicable.
_sp_utl_columnListExists	No alternative is available.	Not applicable.

Table 42. Functions for utilities (continued)		
Netezza function	**function**	**Changes and limitations**
_sp_utl_columnsEqualTypes	No alternative is available.	Not applicable.
_sp_utl_functionExists	No alternative is available.	Not applicable.
_sp_utl_isTempTable	No alternative is available.	Not applicable.
_sp_utl_procedureExists	No alternative is available.	Not applicable.
_sp_utl_relationExists	No alternative is available.	Not applicable.
_sp_utl_sequenceExists	No alternative is available.	Not applicable.
_sp_utl_tableExists	No alternative is available.	Not applicable.
_sp_utl_viewExists	No alternative is available.	Not applicable.
ISDATE_TINY	No alternative is available.	Not applicable.
_sp_utl_getColumnType	No alternative is available.	Not applicable.
_sp_utl_getTableSize	No alternative is available.	Not applicable.
drand64	No alternative is available.	Not applicable.

Back to top ↑

Related information

Analytic stored procedures

IBM PureData System for Analytics (Netezza) and Db2 SQL compatibility

In most cases, Db2 and Netezza products provides equivalent SQL support. For example, in most cases, a particular Netezza SQL command has an equivalent Db2 SQL statement, which you can use without modifying your code. In a limited number of cases, changes are required.

The following tables identify areas where Netezza SQL support differs from Db2 SQL support. For SQL restrictions that are related to stored procedures, see Routines written in NZPLSQL.

Table 43. SQL compatibility: commands	
Netezza SQL command	**Db2 support**
ALTER AGGREGATE	No Db2 ALTER AGGREGATE statement is available. Change the definitions of aggregates by using the Db2 ALTER FUNCTION statement. The IBM Database Harmony Profiler tool converts the ALTER AGGREGATE command to the ALTER FUNCTION statement.
ALTER FUNCTION	The only Netezza ALTER FUNCTION command parameters that are supported by the Db2 ALTER FUNCTION statement are FENCED and NOFENCED. Change the definitions of functions by using the Db2 CREATE [OR REPLACE] FUNCTION statement.
ALTER GROUP	No Db2 ALTER GROUP statement is available. Use roles instead of groups.
ALTER HISTORY CONFIGURATION	A Db2 ALTER HISTORY CONFIGURATION statement is not available. To collect and view historical data about database usage, use the Db2 monitoring capabilities.

Table 43. SQL compatibility: commands (continued)	
Netezza SQL command	**Db2 support**
ALTER LIBRARY	No Db2 ALTER LIBRARY statement is available. The Database Harmony Profiler tool comments out the ALTER LIBRARY command. To ensure that dependent libraries are accessible to your Db2 product, place them in the directory that is specified by the **LIBPATH** or **PATH** environment variable on Windows operating systems or in the instance-level function directory (`$inst_home_dir/sqllib/function/`) on Linux® and UNIX operating systems.
ALTER SCHEMA	The Db2 ALTER SCHEMA statement does not support some clauses of the Netezza ALTER SCHEMA command: • The Db2 ALTER SCHEMA statement does not support the AUTHORIZATION TO clause. Instead, reassign ownership of the schema by using the Db2 TRANSFER OWNERSHIP statement. • The Db2 ALTER SCHEMA statement does not support the SET PATH clause. For dynamic SQL statements, the SQL path is the value of the Db2 CURRENT PATH special register, which you can change by using the **SET PATH** command. For static SQL statements, specify the SQL path by using the **FUNCPATH** bind option.
ALTER SESSION	No Db2 ALTER SESSION statement is available. Consider using the WLM_CANCEL_ACTIVITY or WLM_SET_CONN_ENV procedure.

Table 43. SQL compatibility: commands (continued)

Netezza SQL command	**Db2 support**
ALTER TABLE	The Db2 ALTER TABLE statement does not support some of the functionality of the Netezza ALTER TABLE command: • The Db2 ALTER TABLE statement does not support the RENAME TO clause. Instead, use one of the following approaches: – Rename the table by using the RENAME statement. – Drop the table and re-create it with the new name. – Create an alias with the new name. • The Db2 ALTER TABLE statement does not support the SET PRIVILEGES TO clause. • The row size limit in the Db2 ALTER TABLE statement is 32677. • The Db2 ALTER TABLE statement does not support the DEFERRABLE, NOT DEFERRABLE, or INITIALLY DEFERRED clause. For alternative solutions, consult your IBM representative. The IBM Database Harmony Profiler tool comments out the clause. • The Db2 ALTER TABLE statement does not support the ORGANIZE ON clause. This clause is used in Netezza for clustered base tables (CBTs), which are not supported in Db2 products.
ALTER USER	No Db2 ALTER USER statement is available. To alter users, you can use the web console.
ALTER VIEW	The Db2 ALTER VIEW statement does not support all clauses of the Netezza ALTER VIEW command: • The Db2 ALTER VIEW statement does not support the SET PRIVILEGES TO clause. • The Db2 ALTER VIEW statement does not support the MATERIALIZE SUSPEND clause, which is used in Netezza for materialized views. Materialized query tables (MQTs) are a possible replacement for Netezza materialized views. The Database Harmony Profiler tool converts materialized views to MQTs. To suspend the use of an MQT by the query optimizer, use one of the following approaches: – Issue the ALTER TABLE ... DISABLE QUERY OPTIMIZATION statement. – Convert the MQT to a regular table by issuing the ALTER TABLE ... DROP MATERIALIZED QUERY statement.

Table 43. SQL compatibility: commands (continued)	
Netezza SQL command	**Db2 support**
ALTER VIEWS ON	No Db2 ALTER VIEWS ON statement is available. The ALTER VIEWS ON command is used in Netezza for materialized views. Materialized query tables (MQTs) are a possible replacement for Netezza materialized views. The Database Harmony Profiler tool converts materialized views to MQTs. Modify the MQTs that are associated with the base table in the ALTER VIEWS ON command.
BEGIN	No Db2 BEGIN TRANSACTION statement is available. The Db2 products automatically begin transactions (units of work). Remove the BEGIN TRANSACTION command and, if necessary, adjust the application code to disable the autocommit feature.
COMMENT ON	The Db2 COMMENT ON statement does not support the LIBRARY and DATABASE object types. Instead on commenting on a database, comment on a schema.
COMMIT [WORK|TRANSACTION]	The TRANSACTION parameter is not supported. Remove it or use the WORK parameter. (The ROLLBACK TRANSACTION statement behaves the same way regardless of whether you specify the WORK parameter.)
CREATE AGGREGATE	No Db2 CREATE AGGREGATE statement is available. Define aggregates by using the Db2 CREATE FUNCTION statement. The Database Harmony Profiler tool converts the CREATE AGGREGATE command to the CREATE FUNCTION statement.
CREATE DATABASE	A Db2 CREATE DATABASE statement is not available. The Database Harmony Profiler tool assumes that the database has only one schema and converts the CREATE DATABASE command to a Db2 SET CURRENT SCHEMA statement so that any subsequently created objects are placed under the same schema.
CREATE GROUP	No Db2 CREATE GROUP statement is available. To create groups, you can use the web console.
CREATE HISTORY CONFIGURATION	A Db2 CREATE HISTORY CONFIGURATION statement is not available. To collect and view historical data about database usage, use the Db2 monitoring capabilities.

Table 43. SQL compatibility: commands (continued)	
Netezza SQL command	**Db2 support**
CREATE [OR REPLACE] FUNCTION	The Db2 CREATE FUNCTION statement does not support all of the clauses of the Netezza CREATE [OR REPLACE] FUNCTION command: • The TABLE ALLOWED and TABLE FINAL ALLOWED clauses have no effect in a Db2 CREATE FUNCTION statement. The Database Harmony Profiler tool comments out the TABLE ALLOWED and TABLE FINAL ALLOWED clauses. • In Db2 products, the DEPENDENCIES clause is not used. The Database Harmony Profiler tool comments out the DEPENDENCIES clause. To ensure that dependent libraries are accessible, place them in the directory that is specified by the **LIBPATH** or **PATH** environment variable on Windows operating systems or in the instance-level function directory (`$inst_home_dir/sqllib/function/`) on Linux and UNIX operating systems.
CREATE [OR REPLACE] LIBRARY	No Db2 CREATE [OR REPLACE] LIBRARY statement is available. The Database Harmony Profiler tool comments out the CREATE [OR REPLACE] LIBRARY command. To ensure that dependent libraries are accessible, place them in the directory that is specified by the **LIBPATH** or **PATH** environment variable on Windows operating systems or in the instance-level function directory (`$inst_home_dir/sqllib/function/`) on Linux and UNIX operating systems.
CREATE SCHEDULER RULE	No Db2 CREATE SCHEDULER RULE statement is available. Consider using Db2 WLM capabilities.
CREATE SCHEMA	The Db2 CREATE SCHEMA statement does not support the PATH clause. For dynamic SQL statements, the SQL path is the value of the Db2 CURRENT PATH special register, which you can change by using the **SET PATH** command. For static SQL statements, specify the SQL path by using the **FUNCPATH** bind option.

Table 43. SQL compatibility: commands (continued)	
Netezza SQL command	**Db2 support**
CREATE TABLE	There are several differences between the Db2 CREATE TABLE statement and the Netezza CREATE TABLE command: • The Db2 CREATE TABLE statement does not support the ROW SECURITY clause. Consider using row and column access control (RCAC) instead. • The NOT NULL constraint must be explicitly specified for unique and primary key columns in the Db2 CREATE TABLE statement. • The Db2 CREATE TABLE statement does not support the DEFERRABLE, NOT DEFERRABLE, or INITIALLY DEFERRED clause. For alternative solutions, consult your IBM representative. The Database Harmony Profiler tool comments out the clause. • The Db2 CREATE TABLE statement does not support the ORGANIZE ON clause. This clause is used in Netezza for clustered base tables (CBTs), which are not supported in Db2 products. The Db2 ORGANIZE BY clause is not equivalent to the Netezza ORGANIZE ON clause; substituting the ORGANIZE BY clause for the ORGANIZE ON clause will likely cause problems.
CREATE USER	No Db2 CREATE USER statement is available. To create users, you can use the web console.
DROP GROUP	No Db2 DROP GROUP statement is available. Use roles instead of groups.
DROP HISTORY CONFIGURATION	A Db2 DROP HISTORY CONFIGURATION statement is not available. To collect and view historical data about database usage, use the Db2 monitoring capabilities.
DROP USER	No Db2 DROP USER statement is available. To delete users, you can use the web console.
GENERATE [EXPRESS] STATISTICS	A Db2 GENERATE [EXPRESS] STATISTICS statement is not available. Use the Db2 **RUNSTATS** command or automatic statistics collection. The IBM Database Harmony Profiler tool converts the GENERATE STATISTICS command to the **RUNSTATS** command. The **RUNSTATS** command does not support the column range syntax as used in the GENERATE STATISTICS command, so the Database Harmony Profiler tool does not preserve that syntax.

Table 43. SQL compatibility: commands (continued)	
Netezza SQL command	**Db2 support**
GRANT	The Db2 GRANT statement does not support all the object types and privileges that the Netezza GRANT command supports: • The Db2 GRANT statement does not support the following object types: AGGREGATE, DATABASE, EXTERNAL, GROUP, MANAGEMENT TABLE, MANAGEMENT VIEW, SYNONYM, SYSTEM TABLE, SYSTEM VIEW, and USER. For a suitable alternative, review the Db2 documentation. • Some administration privileges are not supported. For a suitable alternative, review the Db2 documentation.
GROOM TABLE	A Db2 GROOM TABLE statement is not available. For column-organized tables, reorganization is done automatically. For row-organized tables, use the REORG TABLE statement.
INSERT	The Db2 INSERT statement does not support the DEFAULT VALUES clause. Replace the DEFAULT VALUES clause with the DEFAULT keyword for each of the columns. For example, change `INSERT INTO tb1(c1, c2) DEFAULT VALUES` to `INSERT INTO tb1(c1, c2) VALUES (DEFAULT, DEFAULT)`.
LOCK TABLE	The Db2 LOCK TABLE statement does not support the NOWAIT parameter of the Netezza LOCK TABLE command. Also, you must use either the SHARE or the EXCLUSIVE lock mode parameter in the Db2 statement. Consider replacing the Netezza ACCESS EXCLUSIVE mode parameter with the Db2 EXCLUSIVE mode parameter, and consider replacing the Netezza SHARE, SHARE ROW EXCLUSIVE, and EXCLUSIVE mode parameters with the Db2 SHARE mode parameter. The Database Harmony Profiler tool performs the following actions: • Removes the NOWAIT parameter • If you did not specify a mode parameter, adds the IN EXCLUSIVE MODE clause • Replaces the ACCESS EXCLUSIVE mode parameter with the EXCLUSIVE mode parameter • Replaces the SHARE, SHARE ROW EXCLUSIVE, and EXCLUSIVE mode parameters with the SHARE mode parameter If you use the Database Harmony Profiler tool, you must manually convert all other mode parameters to SHARE or EXCLUSIVE. Also, review the LOCK TABLE statements to ensure that your concurrency requirements are still met.

Table 43. SQL compatibility: commands (continued)	
Netezza SQL command	**Db2 support**
REVOKE	The Db2 REVOKE statement does not support all the object types and privileges that the Netezza REVOKE command supports: • The Db2 REVOKE statement does not support the following object types: AGGREGATE, DATABASE, EXTERNAL, GROUP, MANAGEMENT TABLE, MANAGEMENT VIEW, SYNONYM, SYSTEM TABLE, SYSTEM VIEW, and USER. For a suitable alternative, review the Db2 documentation. • Some administration privileges are not supported. For a suitable alternative, review the Db2 documentation.
ROLLBACK [WORK|TRANSACTION]	The TRANSACTION parameter is not supported. Remove it or use the WORK parameter. (The ROLLBACK TRANSACTION statement behaves the same way regardless of whether you specify the WORK parameter.)

Table 43. SQL compatibility: commands (continued)

Netezza SQL command	Db2 support

SELECT	There are multiple differences between the Db2 SELECT statement and the Netezza SELECT command, some of which you can resolve by setting the **SQL_COMPAT** global variable to `'NPS'`:
	• In Db2 products, expressions on column aliases in the SELECT list, as shown in the following example, are not supported by default:
	`SELECT c1 AS a, a+3 AS`
	`b FROM t1;`
	Enable support by setting the **SQL_COMPAT** global variable to `'NPS'`.
	• Referencing SELECT list elements by ordinal positions in the GROUP BY clause, as shown in the following example, is not supported by default:
	`SELECT c1 AS a, c2+c3 AS b,`
	`COUNT(*) AS c FROM t1`
	`GROUP BY 1, 2;`
	Enable support by setting the **SQL_COMPAT** global variable to `'NPS'`.
	Referencing SELECT list elements by ordinal positions in the GROUP BY clause is also not supported by default, but you can enable support by setting the **SQL_COMPAT** global variable to `'NPS'`.
	• Referencing SELECT list elements by aliases in the GROUP BY clause, as shown in the following example, is not supported by default:
	`SELECT c1 as a, count(*)`
	`FROM t1 GROUP BY a`
	`ORDER BY a;`
	Enable support by setting the **SQL_COMPAT** global variable to `'NPS'`.
	Referencing SELECT list elements by aliases in the ORDER BY clause is also not supported by default, but you can enable support by setting the **SQL_COMPAT** global variable to `'NPS'`.
	• The syntax in the following example is not supported by default:
	`SELECT A + B as C , C+2 FROM T1`
	Enable support by setting the **SQL_COMPAT** global variable to `'NPS'`.
	• A column alias in a HAVING or WHERE clause of the Db2 SELECT statement is not supported. Replace the alias in the HAVING or WHERE clause with the corresponding column or expression.
	• The ORDER BY clause is not supported in the definition of a materialized query table (MQT), which is a possible replacement for a Netezza materialized view. The IBM Database Harmony Profiler tool converts materialized views to MQTs.
	Rewrite the query to move the ORDER BY clause to an

Table 43. SQL compatibility: commands (continued)	
Netezza SQL command	**Db2 support**
SET	Db2 products do not support most Netezza session variables. Use other techniques to tune and configure your Db2 product.
SHOW LIBRARY	No Db2 SHOW LIBRARY statement is available. The Database Harmony Profiler tool comments out the SHOW LIBRARY command.
TRUNCATE TABLE	The Db2 TRUNCATE TABLE statement requires the IMMEDIATE parameter. This parameter specifies that the truncate operation is processed immediately and cannot be undone.

Table 44. SQL compatibility: operators	
Netezza operator	**Db2 support**
! factorial operator	This operator is not supported. Instead, you can create a factorial function.
^ and ** exponential operators	These operators are not supported by default. Enable support by setting the **`SQL_COMPAT`** global variable to `'NPS'`.
# bitwise XOR operator	This operator is not supported by default. Enable support by setting the **`SQL_COMPAT`** global variable to `'NPS'`.
<< and >> bitwise left shift and right shift operators	These operators are not supported. You can replace the operators with multiplication or division by powers of 2. For example, `col1 << 4` is equivalent to `col1 * power(2, 4)`.

Table 45. SQL compatibility: miscellaneous items	
Netezza language construct	**Db2 support**
System views (which have names of _V_*viewname* or _VT_*viewname*) and system tables (which have names of _T_*tablename*)	System views and system tables are not supported. Use the Db2 SYSCAT views, where possible. Otherwise, create your own views.
Ampersand (&) in a column name	This syntax is not supported unless you use delimiters, for example, "SALES&".
Underscore (_) as the starting character for an identifier	This syntax is not supported unless you use delimiters, for example, "_SALES".

Compatibility features for Netezza Platform Software (NPS)

Db2 provides features that enable applications that were written for a Netezza Platform Software (NPS) database to use a Db2 database without having to be rewritten.

Some NPS compatibility features (such as equivalent data type names and the DATASLICEID pseudocolumn) are always active; others are active only if the SQL_COMPAT global variable is set to 'NPS'.

Data type aliases

The DATETIME, INT2, INT4, INT8, FLOAT4, FLOAT8, and BPCHAR built-in data types correspond to the identically named Netezza data types.

- DATETIME is an alias for the TIMESTAMP data type.
- INT2 is an alias for the SMALLINT data type.
- INT4 is an alias for the INTEGER data type.
- INT8 is an alias for the BIGINT data type.
- FLOAT4 is an alias for the REAL data type.
- FLOAT8 is an alias for the DOUBLE data type.
- NUMERIC is an alias for the DECIMAL data type.
- BPCHAR is an alias for VARCHAR data type.

If you have a user-defined data type that uses any of these names, you must use a fully-qualified reference to ensure that the user-defined data type is not overridden by the built-in data type alias.

DATASLICEID pseudocolumn

Any unresolved and unqualified column reference to the DATASLICEID pseudocolumn is converted to NODENUMBER function and returns the database partition number for a row. For example, if DATASLICEID is used in a SELECT clause, the database partition number for each row is returned in the result set.

The specific row (and table) for which the database partition number is returned by DATASLICEID is determined from the context of the SQL statement that uses it.

The database partition number returned on transition variables and tables is derived from the current transition values of the distribution key columns.

Before an unqualified reference to 'DATASLICEID' is translated as a NODENUMBER() function, an attempt is made to resolve the reference to one of the following items:

- A column within the current SQL query
- A local variable
- A routine parameter
- A global variable

Avoid using 'DATASLICEID' as a column name or a variable name while DATASLICEID pseudocolumn is needed. All limitations of the DBPARTITIONNUM function (and its alternative, the NODENUMBER function) apply to the DATASLICEID pseusocolumn.

Examples

Example 1

The following example counts the number of instances in which the row for a given employee in the EMPLOYEE table is on a different database partition from the description of the employee's department in the DEPARTMENT table:

```
SELECT COUNT(*) FROM DEPARTMENT D, EMPLOYEE E WHERE D.DEPTNO=E.WORKDEPT AND E.DATASLICEID
<> D.DATASLICEID
```

Example 2

The following example joins the EMPLOYEE and DEPARTMENT tables so that the rows of the two tables are on the same database partition:

```
SELECT * FROM DEPARTMENT D, EMPLOYEE E WHERE E.DATASLICEID = D.DATASLICEID
```

Routines written in NZPLSQL

The NZPLSQL language can be used in addition to the SQL PL language.

SQL PL is a procedural programming language that can be used to write routines. NZPLSQL is a different procedural programming language that is similar to Postgres PL/pgSQL and is used by Netezza Platform Software (NPS). NZPLSQL statements and grammar describes NZPLSQL and its structure.

A NZPLSQL routine is a sent to the NZPLSQL cross-compiler, which converts it from NZPLSQL to SQL PL before sending it on to the SQL PL compiler. If a routine is written in SQL PL, the cross-compiler usually will realize this and will not attempt to convert it before sending it on to the SQL PL compiler. However, because the cross-compiler might mistake SQL PL code for NZPLSQL code, try to convert it, and fail, it is recommended that you do not submit routines written in SQL PL.

Limitations for routines written in NZPLSQL

For a routine written in NZPLSQL, the following limitations apply.

- The routine must define exactly one alias for each of its parameters.
- The routine cannot contain argument lists or variable arguments (varargs).
- If the routine returns a non-null value that has a data type other than binary integer or Boolean, the routine will automatically be converted to a MODIFIES SQL DATA compiled SQL function. This function can then be invoked in the same way as any other compiled SQL function.

Also, use of the following syntax keywords is restricted:

Syntax Keyword	Description
IN EXECUTE	If the routine uses a FOR...IN EXECUTE statement to iterate through the results of a query, it must use a structured record (see "Records in a FOR...IN EXECUTE statement" on page 87).
INTERVAL	The routine cannot use the INTERVAL data type.
LAST_OID	The routine cannot use the LAST_OID clause.
RAISE	The routine cannot use the DEBUG level for RAISE. It can use the NOTICE level, but SERVER OUTPUT must be set to ON for you to be able to see the output.
SELECT	The routine can contain a SELECT statement only if that statement contains a FROM clause. It cannot use a SELECT statement to call another procedure or to retrieve a scalar value. To call another procedure, the routine must use a CALL statement.
TIMETZ	The routine cannot use the TIMETZ data type.
TRANSACTION_ABORTED	The routine cannot issue TRANSACTION_ABORTED exceptions. All exceptions it issues are routed to OTHERS.

NZPLSQL procedures as functions

A routine that returns a non-null value of data type other than binary integer or Boolean is automatically converted to a MODIFIES SQL DATA compiled SQL function. You can then invoke this function in the same way as any other compiled SQL function. A routine that returns a null value, or a binary integer, or a Boolean can also be created as a function by explicitly replacing the PROCEDURE keyword with the FUNCTION keyword.

These procedures that are created either implicitly or explicitly as functions must be dropped by using DROP FUNCTION instead of DROP PROCEDURE.

You can't call these created functions by using the CALL statement but you must use VALUES instead. If you must use a CALL statement, create an SQL PL procedure that calls the NZPLSQL function. The SQL PL procedure must match names and arguments of the function and define the return type as an output parameter.

Example

```
CREATE PROCEDURE MYPROC(INTEGER)
RETURNS VARCHAR(32)
LANGUAGE NZPLSQL
AS
BEGIN_PROC
BEGIN
RETURN 'ABC';
END;
END_PROC;
```

```
CREATE PROCEDURE MYPROC(IN myvar INTEGER, OUT output VARCHAR(32))
LANGUAGE SQL
BEGIN
SET output = MYPROC(myvar) ;
END
```

Records in NZPLSQL

In NZPLSQL, a record can be used in a SELECT operation or FOR statement to hold one database row.

A variable of type RECORD can be used for different selections. Accessing an empty record, or attempting to assign a value to a field of an empty record, results in a runtime error.

A record can be either structured or unstructured:

- An **unstructured record** does not specify a row type, and can be used when the structure of the row being read is unknown. Issue the following statement to declare an unstructured record:

  ```
  recordname RECORD;
  ```

  where `recordname` represents the name of the record.
- A **structured record** explicitly specifies a row type by means of either an AS keyword or the %ROWTYPE attribute. Issue one of the following statements to declare a structured record:

  ```
  recordname RECORD AS row_type;
  recordname tablename%ROWTYPE;
  ```

  where `row_type` represents the name of a predefined row type and `tablename` represents the name of a table whose rows structure corresponds to the desired row type.

To define a row type, issue a CREATE TYPE statement, for example:

```
CREATE TYPE DEPTROW AS ROW (DEPTNO   VARCHAR(3),
                            DEPTNAME VARCHAR(29),
                            MGRNO    CHAR(6),
                            ADMRDEPT CHAR(3),
                            LOCATION CHAR(16))
```

Assigning a complete selection into a record or row

You can use the following query to assign a complete selection into a record or row:

```
SELECT expressions INTO target FROM ...;
```

The target value can be a record, a row variable, or a comma-separated list of variables and record fields or row fields. If the query returns several rows, only the first row is moved into the target fields; all others are discarded.

Note: This interpretation of SELECT INTO is different from that of SQL, in which the INTO target is a newly created table. To create a table from a SELECT result from within an NZPLSQL procedure, use a CREATE TABLE AS SELECT statement.

If the target is a row, record, or variable list, the selected values must exactly match the structure of the target. The FROM keyword can be followed by any valid qualification, grouping, or sorting clauses that can be specified for a SELECT statement.

After a record or row is assigned to a record, you can use dot notation to access the fields of that record. For example, to access the first_name and last_name fields in the record users_rec:

```
DECLARE
    users_rec RECORD AS users_rowtype;
    full_name varchar;
BEGIN
    SELECT * INTO users_rec FROM users WHERE user_id=3;
  full_name := users_rec.first_name || ' ' || users_rec.last_name;
```

Checking whether a value was assigned to a record

There are several ways to check whether a value was assigned to a record by a SELECT INTO statement:

- Use the special variable named FOUND of type Boolean immediately after the SELECT INTO statement. For example:

```
SELECT * INTO myrec FROM EMP WHERE empname = myname;
IF NOT FOUND THEN
    RAISE EXCEPTION 'employee % not found', myname;
END IF;
```

- Use `ROW_COUNT >= 1` instead of `FOUND`.
- Use IS NULL or ISNULL conditionals to test whether a record or row is NULL. For example:

```
DECLARE
    users_rec RECORD AS users_rowtype;
    full_name varchar;
BEGIN
    SELECT * INTO users_rec FROM users WHERE user_id=3;
    IF users_rec.homepage IS NULL THEN
        -- user entered no homepage, return "http://"
        return 'http://';
    END IF;
END;
```

Iterating through the results of a query

There are two methods for iterating through the results of a query and manipulating the result data. In both methods, the record or row is assigned all the rows that are returned by the select clause and the loop body runs for each row.

Use a FOR...IN statement

The syntax of a FOR...IN statement is:

```
[<<label>>]
FOR record_or_row IN select_clause
LOOP
  statements
END LOOP;
```

A FOR...IN statement can use either a structured or an unstructured record type. If the loop is terminated with an EXIT statement, the last assigned row is still accessible.

For example:

```
DECLARE
    mviews RECORD AS cs_materialized_views_rowtype;
    -- this record is usable ONLY for the cs_materialized_views table
BEGIN
    CALL cs_log('Refreshing materialized views...');
    FOR mviews IN SELECT * FROM cs_materialized_views ORDER BY sort_key
    LOOP
    -- Now "mviews" has one record from cs_materialized_views
        RAISE EXCEPTION, 'Can't execute SQL while processing SQL for %',
        mview.my_name;
    END LOOP;
    CALL cs_log('Done refreshing materialized views.');
    return 1;
end;
```

Use a FOR...IN EXECUTE statement

The syntax of a FOR...IN EXECUTE statement is similar to that of a FOR...IN statement, except that the source SELECT statement is specified as a string expression:

```
[<<label>>]
FOR record_or_row IN EXECUTE 'select_clause'
LOOP
  statements
END LOOP;
```

A FOR...IN EXECUTE statement must use a structured record type. A FOR...IN EXECUTE statement lets you create a dynamic statement. For example, you can pass in the SELECT statement as a VARCHAR string into the NZPLSQL procedure.

Records in a FOR...IN EXECUTE statement

A record in a FOR...IN EXECUTE statement must be structured, that is, its row type must be explicitly specified.

Here is an example of an NZPLSQL procedure written for use with Netezza Platform Software (NPS):

```
CREATE or replace PROCEDURE  myproc(varchar(256))
    RETURNS INT4
    LANGUAGE NZPLSQL
AS
BEGIN_PROC
 declare
  sqlstr alias for $1;
  r1 record;
 begin
  FOR r1 IN EXECUTE sqlstr
  loop
   insert into t1 values r1.c1;
  end loop;
 end;
END_PROC@
```

Because the input SQL statement is unknown, the procedure uses a record with an unstructured (that is, generic) row type to retrieve the result. However, Db2 requires that such records use a structured data type. To work around this limitation:

- If you know the structure of the record, define a new row type, then use the AS keyword to specify that type for the record that retrieves the result. For example, define a new row type called myrecord and use it to define the type of record r1:

```
CREATE TYPE myrecord AS ROW (c1 INTEGER)@

CREATE or replace PROCEDURE  myproc(varchar(256))
    RETURNS INT4
    LANGUAGE NZPLSQL
AS
BEGIN_PROC
 declare
  sqlstr alias for $1;
  r1 record AS myrecord;
 begin
  FOR r1 IN EXECUTE sqlstr
  loop
   insert into t1 values r1.c1;
  end loop;
 end;
END_PROC@
```

- If you know that the procedure will select from a particular table (or a table with an identical structure), use an anchored row type for the record that retrieves the result. For example, if the table being read from is table t2, use that table to define the row type of record r1:

```
CREATE or replace PROCEDURE  myproc(varchar(256))
    RETURNS INT4
    LANGUAGE NZPLSQL
AS
BEGIN_PROC
 declare
```

```
   sqlstr alias for $1;
   r1 t2%ROWTYPE;
  begin
   FOR r1 IN EXECUTE sqlstr
   loop
    insert into t1 values r1.c1;
   end loop;
  end;
 END_PROC@
```

Returning a result set

Typically, an NZPLSQL procedure returns a simple return value. However, a NZPLSQL procedure can also be made to return a result set, which has the form of a table.

To create a NZPLSQL procedure that returns a result set:

- Define the procedure with a return clause of the form **RETURNS REFTABLE (<table-name>)**. This indicates that the procedure is to return a result set with the same layout as the specified table. The specified table must exist at the time that the procedure is created, although the table can be empty.
- Within the body of the procedure, use the variable **REFTABLENAME** to refer to the result table.

Example

A 2-column table with the name **tbl1** was defined by the following command:

```
CREATE TABLE tbl1 (i INT4, i2 bigint);
```

The layout of tbl is:

```

                                Data type                     Column
Column name                     schema    Data type name      Length     Scale Nulls
------------------------------- --------- ------------------- ---------- ----- ------
I                               SYSIBM    INTEGER                      4     0 Yes
I2                              SYSIBM    BIGINT                       8     0 Yes
```

The following command defines a procedure with the name **returntwo** that returns a result set that uses **tbl1** as its reference table:

```
DEV.SCH1(ADMIN)=> CREATE OR REPLACE PROCEDURE returntwo(timestamp) RETURNS
REFTABLE(tbl1) LANGUAGE NZPLSQL AS
BEGIN_PROC
  BEGIN
    EXECUTE IMMEDIATE 'INSERT INTO ' || REFTABLENAME ||' values (1,1)';
    EXECUTE IMMEDIATE 'INSERT INTO ' || REFTABLENAME ||' values (2,2)';
    RETURN REFTABLE;
  END;
  END_PROC;
```

Call **returntwo** by issuing the following statement:

```

DEV.SCH1(ADMIN)=> CALL PROCEDURE returntwo(now());
```

This produces the following result:

```
  Result set 1
  --------------
  I          I2
  ----------- --------------------
           1                    1
           2                    2
  2 record(s) selected.
  Return Status = 0
```

Restrictions

An NZPLSQL procedure that returns a result set is subject to the following restrictions:

- The procedure can be invoked only via the CALL statement. When invoked, the database:
  - Generates a table name of the form SESSION.<routinename>, where <routinename> represents the object ID of the procedure that was invoked
  - Issues the following statement to create a temporary table, with no initial contents, for the result set:

    ```
    DECLARE GLOBAL TEMPORARY TABLE <temp-table-name>
      LIKE <table-name> WITH REPLACE ON COMMIT PRESERVE ROWS
    ```

  - Opens a cursor on SELECT * FROM <temp-table-name> and returns the result set

  To use this in a procedure, you must use the REFTABLENAME variable to obtain the name the of temporary table indicated by <temp-table-name>, and insert your results into that table. This SQL command must be invoked dynamically to use the REFTABLENAME variable.
- You must return NULL in your procedure by one of the following methods:
  - RETURN REFTABLE; (this is the recommended method)
  - RETURN NULL;
  - RETURN;
  - By not specifying a RETURN clause

  If you do not return NULL, the procedure returns an error.
- One NZPLSQL procedure that returns a result set can call another. However, due to the temporary-table logic that it employs, if a NZPLSQL procedure that returns a result set calls itself (either directly or recursively), the results are unpredictable. Therefore, avoid designing such procedures.
- Do not issue a ROLLBACK command inside the procedure body unless you first issue a COMMIT command to create the temporary table in which the result is to be stored. Otherwise, the procedure will fail.
- If you run a stored procedure that runs a SELECT statement on a large data set, there might not be enough memory to hold the result. For example, the following stored procedure reads each record from the table with the name table1 and carries out an action on each record:

  ```
  FOR rec in SELECT * from table1 LOOP
  --perform processing steps
  END LOOP:
  ```

  The SELECT operation caches its results in memory or as a temporary file on disk, depending upon the size of the result set. The procedure then applies the steps in the inner processing loop. If the input table is very large, the temporary file might use up all the free disk space on the host.

Using column aliases in a HAVING clause

When operating in NPS compatibility mode, you can specify the exposed name of a SELECT clause column in the HAVING clause of a query.

Whether NPS compatibility mode is being used depends on the setting of the SQL_COMPAT global variable:

- When **SQL_COMPAT='NPS'**, a HAVING clause can refer to a column of a SELECT clause by either its name or its exposed name.
- Otherwise, a HAVING clause can refer to a column of a SELECT clause only by its name, not by its exposed name.

Examples

The following examples illustrate the use of exposed names of SELECT clause columns in having clauses when SQL_COMPAT='NPS':

```
SELECT c1 as a, COUNT(*) as c
FROM t1
GROUP BY c1 having a > 20 and c > 10;
```

```
SELECT t1.c1 as a, t1.c2+t2.c3 as b, COUNT(*) as c
FROM t1 JOIN t2 ON t1.c1 = t2.c1
GROUP BY t1.c1, b having a+5 = 10 ;

SELECT var(c1) as a
FROM t1
GROUP BY c1 having a > 200
```

Using column aliases in a WHERE clause

When operating in NPS compatibility mode, you can reference an expression in the WHERE clause by its alias in the select list.

Whether NPS compatibility mode is being used depends on the setting of the SQL_COMPAT global variable.

When the **SQL_COMPAT='NPS'** parameter is set, an expression in the WHERE clause can be referenced by its alias in the select list.

Attention: The SQL_COMPAT='NPS' feature is only available in Db2 Version 11.5 Mod Pack 1 or later versions.

The WHERE clause can contain non-correlated aliases and correlated aliases.

Examples

The following examples illustrate the use of non-correlated column aliases in the WHERE clause:

```
SELECT c1 as a
FROM t1
WHERE a = 5;

SELECT t1.c1 as a, t1.c2+t2.c3 as b
FROM t1 , t2
WHERE a = t2.c2;

SELECT abs(c1) as a
FROM t1
WHERE a = 4;

SELECT length(c1) as a
FROM t1
WHERE a = 5
GROUP BY c1;
```

The following examples illustrate the use of correlated column aliases in the WHERE clause:

```
select c1 as a1
FROM t1
WHERE c2 in (select c3 from t2 where c3 = a1);

select abs(c1) as a1
FROM t1
WHERE c2 in (select c3 as a3 from t2 where a3 = a1);
```

Double-dot notation

When operating in NPS compatibility mode, you can use double-dot notation to specify a database object.

Double-dot notation follows this format:

```
<NPS_DatabaseName>..<NPS_ObjectName>
```

The two dots indicate that the schema name is not specified. How such a statement is interpreted depends on the setting of the SQL_COMPAT global variable:

- When **SQL_COMPAT='NPS'**, the statement is interpreted as:

  ```
  <SchemaName>.<ObjectName>
  ```

Provided that you moved your NPS database to a schema that has the same name as the database, you can use existing SQL scripts that were written for the NPS database without having to adjust their syntax.

- Otherwise, because two dots refer to a method invocation of an abstract data type, a statement with two dots would result in a syntax error.

BETWEEN scalar functions syntax

In NPS compatibility mode, the DAYS_BETWEEN, HOURS_BETWEEN, MINUTES_BETWEEN, MONTHS_BETWEEN, SECONDS_BETWEEN, and WEEK_BETWEEN scalar functions always return a positive number.

Whether NPS compatibility mode is being used depends on the setting of the SQL_COMPAT global variable.

When **`SQL_COMPAT='NPS'`**, the DAYS_BETWEEN, HOURS_BETWEEN, MINUTES_BETWEEN, MONTHS_BETWEEN, SECONDS_BETWEEN, and WEEK_BETWEEN scalar functions always return a positive number.

For information about the syntax of these scalar functions, see the following descriptions:

- DAYS_BETWEEN scalar function
- HOURS_BETWEEN scalar function
- MINUTES_BETWEEN scalar function
- MONTHS_BETWEEN scalar function
- SECONDS_BETWEEN scalar function
- WEEKS_BETWEEN scalar function

TRANSLATE scalar function syntax

The syntax of the TRANSLATE scalar function depends on whether NPS compatibility mode is being used.

Whether NPS compatibility mode is being used depends on the setting of the SQL_COMPAT global variable:

- When **`SQL_COMPAT='NPS'`**, the syntax of the TRANSLATE scalar function is as described in "Syntax of the TRANSLATE scalar function when SQL_COMPAT='NPS'" on page 92. For example:

  ```
  translate('12345', '143', 'ax')
  ```

  returns:

  ```
  a2x5
  ```

  In the string `'12345'`:

  – The character 1 is translated to a.

  – The character 4 is translated to x.

  – The character 3 does not have a corresponding character in the "to" string, so it is removed.

- Otherwise, the syntax of the TRANSLATE scalar function is as described in TRANSLATE scalar function. For example:

  ```
  translate('12345', 'ax', '143')
  ```

  returns:

  ```
  a2 x5
  ```

  In the string `'12345'`:

  – The character 1 is translated to a.

– The character 4 is translated to x.
– The character 3 does not have a corresponding character in the "to" string, so it is replaced with a padding character. The default padding character is a blank.

Syntax of the TRANSLATE scalar function when SQL_COMPAT='NPS'

If **SQL_COMPAT='NPS'**, the syntax of the TRANSLATE scalar function is:

```
►► TRANSLATE ── ( ── char-string-exp ── , ── from-string-exp ── , ── to-string-exp ── ) ►◄
```

This function converts all the characters in *char-string-exp* that also occur in *from-string-exp* to the corresponding characters in *to-string-exp*. If *from-string-exp* is longer than *to-string-exp*, occurrences of the extra characters in *from-string-exp* are removed from *char-string-exp*.

char-string-exp
The string that is to be converted. The expression must return a value that is a built-in CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, numeric, or datetime data type. If the value is not a CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC data type, it is implicitly cast to VARCHAR before evaluating the function.

from-string-exp
A string of characters that, if found in *char-string-exp*, are to be converted to the corresponding character in *to-string-exp*.

The expression must return a value that is a built-in CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, numeric, or datetime data type. If the value is not a CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC data type, it is implicitly cast to VARCHAR before evaluating the function. If *from-string-exp* contains duplicate characters, the first one found will be used, and the duplicates will be ignored. If *to-string-exp* is longer than *from-string-exp*, the surplus characters will be ignored. If *to-string-exp* is specified, *from-string-exp* must also be specified.

to-string-exp
A string of characters to which certain characters in *char-string-exp* arre to be converted.

The expression must return a value that is a built-in CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, numeric, or datetime data type. If the value is not a CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC data type, it is implicitly cast to VARCHAR before evaluating the function. If a value for *to-string-exp* is not specified, and the data type is not graphic, all characters in *char-string-exp* will be in monocase; that is, the characters a-z will be converted to the characters A-Z, and other characters will be converted to their uppercase equivalents, if they exist. For example, in code page 850, é maps to É, but ÿ is not mapped, because code page 850 does not include Ÿ. If the code point length of the result character is not the same as the code point length of the source character, the source character is not converted.

Operators

Which symbols are used to represent operators in expressions depends on whether NPS compatibility mode is being used.

Whether NPS compatibility mode is being used depends on the setting of the SQL_COMPAT global variable:

- When **SQL_COMPAT='NPS'**, the operators ^ and ** are both interpreted as the exponential operator, and the operator # is interpreted as bitwise XOR.
- Otherwise, the operator ^ is interpreted as bitwise XOR, the operator ** is interpreted as the exponential operator, and the operator # has no meaning (see Expressions).

Grouping by SELECT clause columns

When operating in NPS compatibility mode, you can specify the ordinal position or exposed name of a SELECT clause column when grouping the results of a query.

A GROUP BY clause groups the results of a query that have matching values for one or more grouping expressions. Each column included in a grouping expression must unambiguously identify a column of the query's SELECT clause or an exposed column of the query's intermediate result. If a SELECT clause contains column expressions that are not aggregate expressions, and if a GROUP BY clause is specified, those column expressions must be in the GROUP BY clause. For example:

```
SELECT c1 as a, c2+c3 as b, COUNT(*) as c
FROM t1
GROUP BY c1, c2+c3;
```

The syntax of a GROUP BY clause is described in group-by-clause.

Whether NPS compatibility mode is being used depends on the setting of the SQL_COMPAT global variable:

- When **`SQL_COMPAT='NPS'`**, a grouping expression can refer to a SELECT clause column not only by its name, but also by its ordinal position in the SELECT clause or by its exposed name. This applies to simple grouping expressions, grouping sets, and super groups. The following restrictions apply:
  - An expression is not allowed on an ordinal position or exposed name of a SELECT clause.
  - An integer expression is not treated as an ordinal position of a SELECT clause. Instead, it is treated as a constant and results are grouped by the constant value.
  - The ordinal position of a SELECT clause column cannot be less than one or greater than the number of columns in the result.
- Otherwise, a grouping expression can refer to a SELECT clause column only by its name, not by its ordinal position in the SELECT clause or by its exposed name.

Examples

The following examples illustrate the use of ordinal positions and exposed names of SELECT clause columns in GROUP BY clauses when **`SQL_COMPAT='NPS'`**:

- A simple grouping expression in which columns c1 and c2+c3 are referred to by their ordinal positions in the SELECT clause (1 and 2):

  ```
  SELECT c1 AS a, c2+c3 AS b, COUNT(*) AS c
  FROM t1
  GROUP BY 1, 2;
  ```

- A simple grouping expression in which columns c1 and c2+c3 are referred to by their exposed names (a and b):

  ```
  SELECT c1 AS a, c2+c3 AS b, COUNT(*) AS c
  FROM t1
  GROUP BY a, b;
  ```

- A simple grouping expression in which the GROUP BY clause also references an exposed column of the query's intermediate result (c6):

  ```
  SELECT c1 as a, c2+c3 as b, c4 || c5 as c, COUNT(*) as d
  FROM t1
  GROUP BY 1, b, c4, c5, c6;
  ```

- A grouping set in which columns c1 and c2+c3 are referred to by their ordinal positions in the SELECT clause (1 and 2):

  ```
  SELECT c1 AS a, c2+c3 AS b, COUNT(*) AS c
  FROM t1
  GROUP BY GROUPING SETS ((1, 2), (1), (2));
  ```

- A grouping set in which columns c1 and c2+c3 are referred to by their exposed names (a and b):

```
SELECT c1 AS a, c2+c3 AS b, COUNT(*) AS c
FROM t1
GROUP BY GROUPING SETS ((a, b), (a), (b));
```

- A grouping set in which columns c1 and c2+c3 are referred to by both their ordinal positions in the SELECT clause (1 and 2) and their exposed names (a and b):

```
SELECT c1 AS a, c2+c3 AS b, COUNT(*) AS c
FROM t1
GROUP BY GROUPING SETS ((1, b), (a), (2));
```

- A super group in which columns c1 and c2+c3 are referred to by their ordinal positions in the SELECT clause (1 and 2):

```
SELECT c1 AS a, c2+c3 AS b, COUNT(*) AS c
FROM t1
GROUP BY ROLLUP (1), CUBE (1, 2);
```

- A super group in which columns c1 and c2+c3 are referred to by their exposed names (a and b):

```
SELECT c1 AS a, c2+c3 AS b, COUNT(*) AS c
FROM t1
GROUP BY ROLLUP (a), CUBE (a, b);
```

- A super group in which columns c1 and c2+c3 are referred to by both their ordinal positions in the SELECT clause (1 and 2) and their exposed names (a and b):

```
SELECT c1 AS a, c2+c3 AS b, COUNT(*) AS c
FROM t1
GROUP BY a, ROLLUP (a, 2), CUBE (b);
```

- Each of the following statements groups by the column t.c1:

```
SELECT c1 FROM t GROUP BY 1
    SELECT c1 FROM t GROUP BY c1
    SELECT c1 AS c1_alias FROM t GROUP BY c1_alias
```

However, in the following statement, the 1 in c2+1 is interpreted as a number, not as an ordinal position, so c2+1 is not equivalent to c2+c1:

```
SELECT c1,c2 FROM t GROUP BY c2+1
```

Expressions refer to column aliases

When operating in NPS compatibility mode, an expression can refer to column aliases that are set in the select list.

Whether you are operating in NPS compatibility mode depends on the setting of the SQL_COMPAT global variable:

- When **SQL_COMPAT='NPS'**, an expression can refer to either a column name or a column alias that is set in the select list. The column resolution order for an unqualified column reference in a select list is:
  1. Resolve as a procedure argument or variable references.
  2. Resolve in the input tables of the current operation.
  3. Resolve against column aliases that appear before this reference in the select list.
  4. Resolve as a correlated column reference.
  5. Resolve as a trigger reference.
  6. Resolve in an external reference table (used by LOAD).
  7. Resolve as a global variable reference.
- Otherwise, a grouping expression can refer only to a column name.

Examples

The following examples illustrate the use, in an expression, of column aliases (a and b) that are set in the select list:

```
SELECT c1 AS a, a+3 AS b FROM t1;

SELECT c1 AS a, ABS(a) AS b FROM t1 GROUP BY a, b HAVING c1 < 0;

SELECT c1+c3 AS a, CASE WHEN a < 5 THEN a ELSE NULL END AS b FROM t1;
```

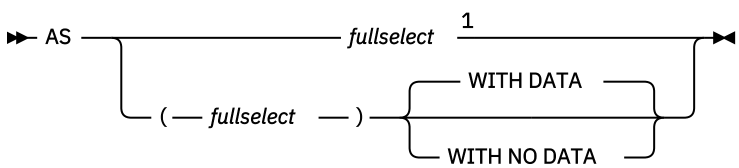
CREATE TABLE statement can use CTAS syntax

When operating in NPS compatibility mode, the AS clause of a CREATE TABLE statement can use the same syntax as the corresponding clause of a Netezza CREATE TABLE AS command (sometimes referred to as a CTAS command).

Note: When the EXPLAIN mode is ON and a CTAS statement is explained, the target table is created but no data is loaded into it.

Whether you are operating in NPS compatibility mode depends on the setting of the SQL_COMPAT global variable:

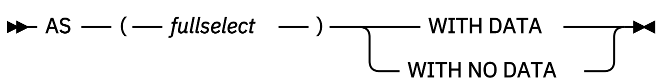
- When **`SQL_COMPAT='NPS'`**, the AS clause of a CREATE TABLE statement can use the following syntax:



  Notes:

  [1] The table is populated with the result of the query that is specified in the fullselect statement. This is equivalent to specifying `(`*`fullselect`*`) WITH DATA`.

- Otherwise, the AS clause of a CREATE TABLE statement must use the following syntax:



  If WITH DATA is specified, the table is populated with the result of the query that is specified in the fullselect statement. If WITH NO DATA is specified, the fullselect statement is used only to define the table, but the table is not populated with the results of the query.

SUBSTR allows non-positive start values

When operating in NPS compatibility mode, the SUBSTR scalar function allows the start value to be negative, zero, or positive.

The start argument of the SUBSTR scalar function specifies the position, relative to the beginning of the input expression, from which the substring is to be calculated.

For example:

- Position 1 is the first string unit of the input expression.
- Position 2 is one position to the right of position 1.
- Position 0 is one position to the left of position 1.
- Position -1 is two positions to the left of position 1.

Which values are valid for the start argument is determined by the SQL_COMPAT global variable:

- If `SQL_COMPAT='NPS'`, the value can be any positive, zero, or negative number:
- Otherwise, the value must be at least 1 and at most the maximum length of the input string (SQLSTATE 22011 if out of range). Also, it must be specified as a number of string units in the context of the database code page, not the application code page.

Examples

If `SQL_COMPAT='NPS'`:

- The statement SUBSTR`('abcd',1,2)` returns `'ab'`.
- The statement SUBSTR`('abcd',2,2)` returns `'bc'`.
- The statement SUBSTR`('abcd',0,2)` returns `'a'`.
- The statement SUBSTR`('abcd',-1,2)` returns a zero-length string.
- The statement SUBSTR`('BLUE JAY',0,4)` returns `'BLU'`.
- The statement SUBSTR`('BLUE JAY',-1,4)` returns `'BL'`.

AGE returns a decimal duration

When operating in NPS compatibility mode, the AGE scalar function returns a decimal duration instead of an integer value.

Whether NPS compatibility mode is being used depends on the setting of the SQL_COMPAT global variable:

- When **`SQL_COMPAT='NPS'`**, the AGE scalar function returns a decimal duration:
  - When a single argument is specified and the data type of the input argument is DATE, the AGE scalar function returns a **date** duration.
  - When a single argument is specified and the data type of the input argument is TIMESTAMP, the AGE scalar function returns a **timestamp** duration.
  - When two arguments are specified and the data type of both input arguments is DATE, the AGE scalar function returns a **date** duration.
  - When two arguments are specified and the data type of either input argument is TIMESTAMP, the AGE scalar function returns a **timestamp** duration.

  For more information about date and timestamp durations, see Datetime operations and durations.
- Otherwise, the AGE scalar function returns an integer value as described in AGE scalar function.

If **`SQL_COMPAT='NPS'`**...	...is equivalent to...
AGE(*date-expression*)	(CURRENT_DATE – *date-expression*)
AGE(*timestamp-expression*)	(CURRENT_TIMESTAMP – *timestamp-expression*)
AGE(*date-expression1*,*date-expression2*)	(*date-expression1* - *date-expression2*)
AGE(*timestamp-expression1*,*timestamp-expression2*)	(*timestamp-expression1* - *timestamp-expression2*)

Precision and scale for DECIMAL and NUMERIC scalar functions

The default precision and scale used by the DECIMAL and NUMERIC scalar functions depend on whether NPS compatibility mode is being used.

The default precision and scale used by the DECIMAL and NUMERIC scalar functions depend on the setting of the SQL_COMPAT global variable:

- When **`SQL_COMPAT='NPS'`** and the data type of the input expression is:
  - DECIMAL, the default precision and scale are the same as the precision and scale of the input data.
  - REAL or DOUBLE, the default precision is 15 and the default scale is 6.
- Otherwise, the default precision and scale are as described in DECIMAL or DEC scalar function.

Installing the DB SQL Extension Toolkit

You can install the DB SQL Extension Toolkit in your Db2 environment to add a variety of functions that perform XML processing, compression, hashing, and other analytics that you used in your Netezza SQL queries.

About this task

DB SQL Extension Toolkit is supported on Db2 Warehouse, Db2 Warehouse on Cloud, and IBM Integrated Analytics System. You can download the tool from IBM DeveloperWorks page.

For the list of functions that are implemented, see Release notes.

Refer to this link for details of available functions https://www.ibm.com/support/knowledgecenter/en/SSULQD_7.2.1/com.ibm.nz.sqltk.doc/c_sqltk_plg_overview.html

Prerequisites:

- Db2 client driver must be installed on the system. CLPPLUS is required to run the setup.
- Database user credentials which has the privileges to create schema/functions in database.

Usage:

```
./setup.py [-user USER -schema SCHEMA] {[-dsn DSN] | [-pw PASS -port PORT -db DATABASE -host
HOST_IP]} [-cleanup]

  -h, --help      show this help message and exit
  -user USER      Username with which you want to register udx. Default is bluadmin.
  -pw PW          Password of -u user; prompts user if -u is provided.
  -host HOST      Host IP if you're using Db2WoC. Default is current host's IP.
  -port PORT      Port number in case of SSL connections. Default is 50000.
  -db DB          Database name. Default is BLUDB.
  -dsn DSN        Used in case of SSL connection mostly.
  -schema SCHEMA  Schema name in which user wants to register the UDXes. Default is $USER.
  -cleanup        Cleanup the sqltoolkit funtions created.
```

Procedure

1. Download the package according to the architecture of your database server. For example, for x86_64 platform:

   ```
   [bluadmin@host - Db2wh ga_1.2]$ uname -i
           x86_64
           [bluadmin@host - Db2wh ga_1.2]$ ls
           db_sqltoolkit_x86_64_1.2.tgz
   ```

2. Extract the package to the location accessible to the user you logged in:

   ```
   -- tar -xzf db_sqltoolkit_arch.tgz
   ```

   Example:

   ```

           [bluadmin@host - Db2wh ga_1.2]$ ls
           db_sqltoolkit_x86_64  db_sqltoolkit_x86_64_1.2.tgz
   ```

3. Change the directory to `db_sqltoolkit_arch`.

   Example:

```
[bluadmin@host - Db2wh ga_1.2]$ cd db_sqltoolkit_x86_64
        [bluadmin@host - Db2wh db_sqltoolkit_x86_64]$ ll
        total 4100
        -rw-r--r-- 1 bluadmin bluadmin   40011 Sep  3 12:39 create_sql_function.base
        -rw-r--r-- 1 bluadmin bluadmin     182 Sep  5 13:09 deploy.base
        -rw-r--r-- 1 bluadmin bluadmin     536 Aug 14 14:19 deploy_toolkit.cpp
        -rw-r--r-- 1 bluadmin bluadmin    8556 Sep  5 13:33 drop_sql_function.base
        -rw-r--r-- 1 bluadmin bluadmin      22 Sep  5 10:10 get_string_units.base
        -rw-r--r-- 1 bluadmin bluadmin      80 Aug 29 14:05 get_upload_path.sql
        -rw-r--r-- 1 bluadmin bluadmin    2069 Aug 14 14:19 README
        -rwxr-xr-x 1 bluadmin bluadmin   11932 Sep  5 14:31 setup.py
        -rw-r--r-- 1 bluadmin bluadmin 4106240 Aug 14 14:19 toolkit_libs.tar
        -rw-r--r-- 1 bluadmin bluadmin      36 Aug 29 14:04 upload.sql
```

The directory contains a `setup.sh` file.

4. Run **`./setup.sh -h`** to display help for options.
5. When DSN option is used, only the user, schema, and cleanup options are valid. Other connection options are ignored.

   Note: When using DSN option, add the DSN as follows:

   a. In catalog, using **`db_catalog`** command.
   b. Add DSN and database in `db2dsdriver.cfg` using **`db2cli writecfg`**. This enables WebAPIs required for CLPPlus.
6. If the password is not provided in the command line argument, you will be prompted for it. In case of DSN, enter the password only when prompted. Even if the password is provided as command line argument, it will be ignored in case of DSN.

```
[bluadmin@host - Db2wh db_sqltoolkit_x86_64]$ ./setup.py -schema sqltk
        [INFO] : Checking required files ...
        [INFO] : Connecting user bluadmin with 172.16.176.74:50000/BLUDB
        Enter password for bluadmin :
        [INFO] : STRING_UNITS detected as : SYSTEM
        [INFO] : The files have been uploaded.
        [INFO] : Registering the functions ...
        [INFO] : Successfully operated ..
        [bluadmin@host - Db2wh db_sqltoolkit_x86_64]$
```

Note: It is advised to use the function with SCHEMA_NAME in which you've registered the toolkit. Otherwise, it is possible that you may get unexpected results due to an already existing function with same name. This function may exist in schema with higher precedence in CURRENT_PATH.

Example:

```
db2 => values CURRENT_PATH

1
----------------------------------------------------------------------------------------
-------
"SYSIBM","SYSFUN","SYSPROC","SYSIBMADM","BLUADMIN"

In the above path, SYSIBM schema has the highest precedence. If a function exists in both
SYSIBM and BLUADMIN schemas, and you run that function without mentioning the schema, the
version in SYSIBM will be executed.

db2 => values BLUADMIN.COMPRESS('some_string')
```

Known issues:

- For ARRAY_COMBINE function, more than one character delimiter does not work.
- ENCRYPT and COMPRESS results may vary as compared to Netezza. If you encrypt a string using ENCRYPT function and use DECRYPT function on result of former returns original string. Similar for COMPRESS and DECOMPRESS functions. For example, `VALUES(DECRYPT(ENCRYPT('string')) = 'string'`

Best practices

- NVARCHAR is internally handled in VARCHAR in Db2. Thus, registering the functions with VARCHAR variant only will internally handle NVARCHAR as well.
- ARRAY objects are fits in VARCHAR column of the table. It is user's responsibility to allocate appropriate memory in VARCHAR.

  For example, VARCHAR(100) might give error while adding multiple BIGINT values. However, changing it to VARCHAR(10000) will solve the issue.
- Since BYTEINT is converted to SMALLINT in db2, ARRAY(1) gives error for such query. User is advised to use ARRAY(2) , as BYTEINT values are handled in SMALLINT.
- User needs to use functions with schema name under which they are registered. Schema will be the user name with which the package was installed.
- Output representation is different for DOUBLE/BOOLEAN in Db2 compared to PDA. So, you might observe output difference in case you are using DOUBLE/BOOLEAN data types with the functions.
- If your workload/query involves "NCHAR", "NVARCHAR" data then your system must be set with appropriate STRING_UNITS configurations.

Uninstalling the DB SQL Extension Toolkit

The cleanup of the SQL Extension Toolkit requires the schema which was provided while installing. If not provided, the user with which the script is run will be treated as schema name.

Procedure

Use the following command to uninstall the tool:

```
[bluadmin@host - Db2wh db_sqltoolkit_x86_64]$ ./setup.py -schema sqltk -cleanup
        [INFO] : Checking required files ...
        [INFO] : Connecting user bluadmin with 172.16.176.74:50000/BLUDB
        Enter password for bluadmin :
        [INFO] : STRING_UNITS detected as : SYSTEM
        [INFO] : De-registering functions ...
        [INFO] : Successfully operated ..
        [bluadmin@host - Db2wh db_sqltoolkit_x86_64]$
```

Examples

Installing the toolkit:

- using connection identifiers:

  ```
  ./setup.py -user bluadmin -host myhost_ip -schema db_toolkit
  ```
- using DSN:

  ```
  ./setup.py -user bluadmin -dsn MY_DSN -schema db_toolkit
  ```

Uninstalling the toolkit:

- using connection identifiers:

  ```
  ./setup.py -user bluadmin -host myhost_ip -schema db_toolkit -cleanup
  ```
- using DSN:

  ```
  ./setup.py -user bluadmin -dsn MY_DSN -schema db_toolkit -cleanup
  ```

Using DB2_REVERSE_NULL_ORDER

In Db2, NULL values are considered higher than any other values. By enabling NULL ordering, NULLS are considered as the smallest values in sorting. You can enable this new option by setting the

DB2_REVERSE_NULL_ORDER registry variable to DB2_REVERSE_NULL_ORDER=TRUE. By default, the DB2_REVERSE_NULL_ORDER registry variable is set to FALSE.

To enable, set the registry variable DB2_REVERSE_NULL_ORDER to TRUE. By default, DB2_REVERSE_NULL_ORDER is set to FALSE.

The following examples illustrate the use of DB2_REVERSE_NULL_ORDER:

```
CREATE table t1 ( c1 integer, c2 char(1));
INSERT into t1 values
( 1, 'A'),( 2, NULL),( 3, 'C'),( 4, NULL),( 5, 'E');
```

With DB2_REVERSE_NULL_ORDER disabled, select * from t1 order by c2:

Table 46.

c1	c2
1	A
3	C
5	E
2	-
4	-

With DB2_REVERSE_NULL_ORDER enabled, the following table displays DB2_REVERSE_NULL_ORDER set to TRUE.

Table 47.

c1	c2
2	-
4	-
1	A
3	C
5	E

IBM Database Conversion Workbench (DCW)

IBM Database Conversion Workbench (DCW) is a no-charge plug-in that adds database migration capabilities to IBM Data Studio.

You can download both Data Studio and DCW from IBM developerWorks.

Related information

DCW home page

Index

W