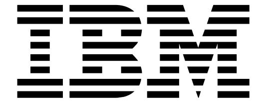
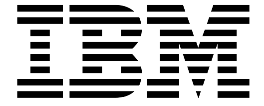
Db2 11.1 for Linux, UNIX, and Windows



Compatibility Features

Db2 11.1 for Linux, UNIX, and Windows



Compatibility Features

Notice regarding this document

This document in PDF form is provided as a courtesy to customers who have requested documentation in this format. It is provided As-Is without warranty or maintenance commitment.

© Copyright IBM Corp. 1994, 2017

Contents

Notice regarding this document iii

Figures vii

Tables ix

Compatibility features. 1

Compatibility features for Oracle 1
- Data types provided for Oracle compatability . . 1
- Implicit casting for character and graphic constants 11
- SQL data-access-level enforcement. 12
- Outer join operator 13
- Hierarchical queries 15
- Compatibility database configuration parameters 23
- ROWNUM pseudocolumn 24
- DUAL table 25
- Changed syntax for the TRUNCATE statement 25
- Insensitive cursor 25
- INOUT parameters 27
- Currently committed semantics. 27
- Oracle data dictionary-compatible views. . . . 29
- Oracle database link syntax 31
- Synonym usage 32

DB2_COMPATIBILITY_VECTOR registry variable 32
Setting up the Db2 environment for Oracle application enablement 36
Terminology mapping: Oracle to Db2 products . . 38
Compatibility features for Netezza Platform Software (NPS) 42
- Data type aliases 43
- DATASLICEID pseudocolumn 43
- Routines written in NZPLSQL 44
- Double-dot notation 45
- TRANSLATE scalar function syntax 45
- Operators 47
- Grouping by SELECT clause columns 47
- Expressions refer to column aliases 49

IBM Database Conversion Workbench (DCW) . . . 49

Index 51

© Copyright IBM Corp. 1994, 2017

Figures

© Copyright IBM Corp. 1994, 2017

Tables

1. TIMESTAMPDIFF computations 3
2. Rounding for numeric assignments and casts 6
3. Modified rules for result data types that involve character strings 8
4. Data Type and lengths of concatenated operands 8
5. The italicized variables in the previous table have the following values 9
6. Oracle data dictionary-compatible views 30
7. DB2_COMPATIBILITY_VECTOR bit positions 33
8. Mapping of common Oracle concepts to Db2 concepts 38

© Copyright IBM Corp. 1994, 2017

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.

In addition to the Oracle Compatibility features that are always active, you can activate the following optional Oracle compatibility features by setting the "DB2_COMPATIBILITY_VECTOR registry variable" on page 32 (which are features that are inactive by default):

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.

© Copyright IBM Corp. 1994, 2017

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

	Target data type			
Source 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. Non-Unicode code pages are not supported when setting the **DB2_COMPATIBILITY_VECTOR** registry variable to hexadecimal value 0x20 (bit position 6).

```
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*))

Table 4. Data Type and lengths of concatenated operands (continued).

Operands	Combined length attributes[1]	Result[1]
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*))
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.

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.

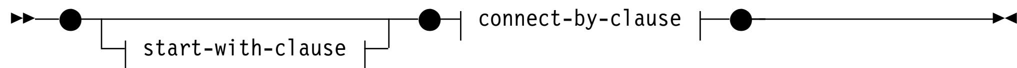
You can then use CONNECT BY syntax, including pseudocolumns, unary operators, and the SYS_CONNECT_BY_PATH scalar function.

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:

```
├──START WITH──search-condition──────────────────────────────────┤
```

connect-by-clause:

```
├──CONNECT BY──┬─────────┬──search-condition──────────────────────┤
               └─NOCYCLE─┘
```

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):

  $path_1$:= *string-expression2* || *string-expression1*

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

  path$_n$:= path$_{n-1}$ || *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).

Example

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 Oracle applications, the recommended setting for the DB2_COMPATIBILITY_VECTOR is ORA, 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.

Example

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.

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
  - 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
- In a Db2 pureScale® environment, currently committed semantics only apply to lock conflicts between applications running on the same member. If the lock conflict is with an application on a remote member, the requester of the lock waits for the lock to be released before processing the row.

Example

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.

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.

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 to support 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.

Example

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

Example

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.

Hexadecimal `00000000 - FFFFFFFF`
Each bit in the variable value enables an individual compatibility feature. For the meanings of each bit, see Table 7.

`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. (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. 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.

Table 7. DB2_COMPATIBILITY_VECTOR bit positions (continued)

Bit position	Hexadecimal value	ORA	SYB	Compatibility feature	Description
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.
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.

Table 7. DB2_COMPATIBILITY_VECTOR bit positions (continued)

Bit position	Hexadecimal value	ORA	SYB	Compatibility feature	Description
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 27	This bit enables the specification of DEFAULT for INOUT parameter declarations.
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 12	This bit enables routines to enforce SQL data-access levels at run time.
18	0x20000	•		"Oracle database link syntax" on page 31	This bit enables Oracle database link syntax for accessing objects in other databases.
19	0x40000	•		"Synonym usage" on page 32	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. However, it is recommended that you explicitly set the **DB2_DEFERRED_PREPARE_SEMANTICS** registry variable to YES.
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 (non-Unicode code pages are not supported when setting **DB2_COMPATIBILITY_VECTOR=ORA**). 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 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.

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

Oracle concept	Db2 concept	Notes
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.
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.

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

Oracle concept	Db2 concept	Notes
inactive log	online archive log	The concepts are the same.
`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.

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

Oracle concept	Db2 concept	Notes
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.
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.

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

Oracle concept	Db2 concept	Notes
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.
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.

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'.

Related information:

Analyzing with SQL

Data type aliases

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

- 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.
- BPCHAR is an alias for VARCHAR data type, but can be used only as a cast target data type. The BPCHAR data type does not accept a length argument.

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
```

Related information:

DBPARTITIONNUM scalar function

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.

When you create a routine, the process by which it is compiled depends on the setting of the SQL_COMPAT global variable:

- When **`SQL_COMPAT='NPS'`**, the 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 when operating in NPS compatibility mode.
- Otherwise, the routine is sent directly to the SQL PL compiler. A routine written in NZPLSQL (or any other language) will almost certainly result in a syntax error.

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 must return an integer, Boolean, or null value. If the routine returns a non-integer or non-Boolean value, it must use a `return;` (or `return null;`) statement.
- The routine cannot contain argument lists or variable arguments (varargs).

Also, use of the following syntax keywords is restricted:

Syntax Keyword	Description
ARRAY	The routine cannot contain array variables.
AUTOCOMMIT	The routine cannot use the following SQL clauses: • BEGIN AUTOCOMMIT ON • AUTOCOMMIT ON blocks
FOR	The routine cannot use record types except in a FOR statement.

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).
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.
REFTABLE	The return type cannot be REFTABLE.
RUN AS	The routine cannot contain a RUN AS statement. Instead, you must use some other means to authorize the use of the static package that is associated with the stored procedure.
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.

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.

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'." 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;
```

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.

Index

A

actual parameter 38
alert log 38
archive log 38
archive log mode 38

B

bdump directory 38
BPCHAR data type 43
  details 43

C

character constants 11
compatibility
  features summary 1, 43, 44, 45, 47, 49, 50
concurrency
  improving 27
configuration parameters
  date_compat 1, 23
  number_compat 4, 23
  varchar2_compat 7, 23
CONNECT BY clause 15
CONNECT_BY_ROOT unary operator 20
constants
  handling 11
cur_commit database configuration parameter
  overview 27
cursor sharing 38
cursors
  insensitive 26

D

data
  access levels 12
data block 38
data buffer cache 38
data dictionaries
  Db2-Oracle terminology mapping 38
  Oracle
    compatible views 29
data dictionary cache 38
data file 38
data types
  BPCHAR 43
  DATE 1
  FLOAT4 43
  FLOAT8 43
  INT2 43
  INT4 43
  INT8 43
  NUMBER 4
  NVARCHAR2 7
  VARCHAR2 7
data-access levels
  routines 12
  stored procedures 12
  user-defined functions 12
database links
  syntax 31
  terminology mapping 38
DATASLICEID
  ROWNUM 43
DATASLICEID pseudocolumn 43
DATE data type
  based on TIMESTAMP(0) 1
date_compat database configuration parameter
  DATE based on TIMESTAMP(0) 1
  overview 23
DB2_COMPATIBILITY_VECTOR registry variable
  details 32
deadlocks
  avoiding 27
DUAL table 25
dynamic performance views 38

F

FLOAT4 data type 43
  details 43
FLOAT8 data type 43
  details 43
formal parameter 38
functions
  scalar
    CONCAT 7
    INSERT 7
    LENGTH 7
    REPLACE 7
    SUBSTR 7
    SYS_CONNECT_BY_PATH 22
    TRANSLATE 7
    TRIM 7

G

global index 38
graphic data
  constants
    handling 11

H

hierarchical queries 15

I

inactive log 38
init.ora 38
INOUT parameters 27
insensitive cursors 26
INT2 data type 43
  details 43
INT4 data type 43
  details 43
INT8 data type 43
  details 43

L

large pool 38
LEVEL pseudocolumn 15
library cache 38
literals
  handling 11
local index 38
locks
  timeouts
    avoiding 27

M

materialized view 38

N

noarchive log mode 38
NULL-producer 13
NUMBER data type
  details 4
number_compat database configuration parameter
  effect 4
  overview 23
NVARCHAR2 data type 7

O

OLAP
  specification 24
operators
  CONNECT_BY_ROOT 20
  outer join 13
  PRIOR 21
  unary 15
Oracle
  application enablement 36
  data dictionary--compatible views 29
  database link syntax 31
  Db2 terminology mapping 38
Oracle Call Interface (OCI) 38
ORACLE_SID environment variable 38

P

parameters
  INOUT 27
PL/SQL
  Oracle application enablement 36
PRIOR unary operator 21

© Copyright IBM Corp. 1994, 2017

program global area (PGA) 38
pseudocolumns
LEVEL 15
ROWNUM 24

Q

queries
hierarchical 15

R

redo log 38
registry variables
DB2_COMPATIBILITY_VECTOR 32
rounding 4
routines
data-access levels 12
ROW_NUMBER() OVER() function 24
ROWNUM pseudocolumn 24

S

segment 38
Server Parameter File (SPFILE) 38
session 38
START WITH clause 15
startup nomount 38
stored procedures
data-access levels 12
strings
NVARCHAR2 data type 7
VARCHAR2 data type 7
synonyms
DB2_COMPATIBILITY_VECTOR registry variable 32
Db2-Oracle terminology mapping 38
SYS_CONNECT_BY_PATH scalar function 22
system global area (SGA) 38
SYSTEM table space 38

T

tables
DUAL 25
terminology mapping
Db2-Oracle 38
TIMESTAMP(0) data type
DATE data type based on 1

U

UDFs
data-access levels 12
unary operators
CONNECT_BY_ROOT 15, 20
PRIOR 21
user global area (UGA) 38

V

VARCHAR2 data type
details 7
varchar2_compat database configuration parameter
overview 23
VARCHAR2 data type 7
views
Oracle data dictionary compatibility 29

Printed in USA