Database SQL Language Reference

Data Type Comparison Rules

This section describes how Oracle Database compares values of each data type.

Numeric Values

A larger value is considered greater than a smaller one. All negative numbers are less than zero and all positive numbers. Thus, -1 is less than 100; -100 is less than -1.

The floating-point value `NaN` (not a number) is greater than any other numeric value and is equal to itself.

> **See Also:**
>
> "Numeric Precedence" (sql_elements001.htm#i156865) and "Floating-Point Numbers" (sql_elements001.htm#i140176) for more information on comparison semantics

Date Values

A later date is considered greater than an earlier one. For example, the date equivalent of '29-MAR-2005' is less than that of '05-JAN-2006' and '05-JAN-2006 1:35pm' is greater than '05-JAN-2005 10:09am'.

Character Values

Character values are compared on the basis of two measures:

- Binary or linguistic sorting
- Blank-padded or nonpadded comparison semantics

The following subsections describe the two measures.

Binary and Linguistic Comparisons

In binary comparison, which is the default, Oracle compares character strings according to the concatenated value of the numeric codes of the characters in the database character set. One character is greater than another if it has a greater numeric value than the other in the character set. Oracle considers blanks to be less than any character, which is true in most character sets.

These are some common character sets:

- 7-bit ASCII (American Standard Code for Information Interchange)
- EBCDIC Code (Extended Binary Coded Decimal Interchange Code)
- ISO 8859/1 (International Organization for Standardization)
- JEUC Japan Extended UNIX

Linguistic comparison is useful if the binary sequence of numeric codes does not match the linguistic sequence of the characters you are comparing. Linguistic comparison is used if the `NLS_SORT` parameter has a setting other than `BINARY` and the `NLS_COMP` parameter is set to `LINGUISTIC`. In linguistic sorting, all SQL sorting and comparison are based on the linguistic rule specified by `NLS_SORT`.

> ## See Also:
>
> *Oracle Database Globalization Support Guide* (../NLSPG/ch5lingsort.htm#NLSPG005) for more information about linguistic sorting

Blank-Padded and Nonpadded Comparison Semantics

With blank-padded semantics, if the two values have different lengths, then Oracle first adds blanks to the end of the shorter one so their lengths are equal. Oracle then compares the values character by character up to the first character that differs. The value with the greater character in the first differing position is considered greater. If two values have no differing characters, then they are considered equal. This rule means that two values are equal if they differ only in the number of trailing blanks. Oracle uses blank-padded comparison semantics only when both values in the comparison are either expressions of data type `CHAR`, `NCHAR`, text literals, or values returned by the `USER` function.

With nonpadded semantics, Oracle compares two values character by character up to the first character that differs. The value with the greater character in that position is considered greater. If two values of different length are identical up to the end of the shorter one, then the longer value is considered greater. If two values of equal length have no differing characters, then the values are considered equal. Oracle uses nonpadded comparison semantics whenever one or both values in the comparison have the data type `VARCHAR2` or `NVARCHAR2`.

The results of comparing two character values using different comparison semantics may vary. The table that follows shows the results of comparing five pairs of character values using each comparison semantic. Usually, the results of blank-padded and nonpadded comparisons are the same. The last comparison in the table illustrates the differences between the blank-padded and nonpadded comparison semantics.

Blank-Padded	Nonpadded
`'ac' > 'ab'`	`'ac' > 'ab'`
`'ab' > 'a '`	`'ab' > 'a  '`
`'ab' > 'a'`	`'ab' > 'a'`
`'ab' = 'ab'`	`'ab' = 'ab'`
`'a ' = 'a'`	**`'a ' > 'a'`**

Portions of the ASCII and EBCDIC character sets appear in Table 2-8 (#g196234) and Table 2-9 (#g196333) . Uppercase and lowercase letters are not equivalent. The numeric values for the characters of a character set may not match the linguistic sequence for a particular language.

Table 2-8 ASCII Character Set

Symbol	Decimal value	Symbol	Decimal value
blank	32	;	59
!	33	<	60
"	34	=	61
#	35	>	62
$	36	?	63
%	37	@	64
&	38	A-Z	65-90
'	39	[	91
(	40	\	92
)	41	]	93
*	42	^	94
+	43	_	95
,	44	'	96
-	45	a-z	97-122
.	46	{	123

Symbol	Decimal value	Symbol	Decimal value
/	47	|	124
0-9	48-57	}	125
:	58	~	126

Table 2-9 EBCDIC Character Set

Symbol	Decimal value	Symbol	Decimal value
blank	64	%	108
¢	74	_	109
.	75	>	110
<	76	?	111
(	77	:	122
+	78	#	123
|	79	@	124
&	80	'	125
!	90	=	126
$	91	"	127
*	92	a-i	129-137
)	93	j-r	145-153

Symbol	Decimal value	Symbol	Decimal value
;	94	s-z	162-169
ÿ	95	A-I	193-201
-	96	J-R	209-217
/	97	S-Z	226-233

Object Values

Object values are compared using one of two comparison functions: `MAP` and `ORDER`. Both functions compare object type instances, but they are quite different from one another. These functions must be specified as part of any object type that will be compared with other object types.

> **See Also:**
>
> CREATE TYPE (statements_8001.htm#BABHJHEB) for a description of `MAP` and `ORDER` methods and the values they return

Varrays and Nested Tables

Comparison of nested tables is described in "Comparison Conditions" (conditions002.htm#i1033286) .

Data Type Precedence

Oracle uses data type precedence to determine implicit data type conversion, which is discussed in the section that follows. Oracle data types take the following precedence:

- Datetime and interval data types
- `BINARY_DOUBLE`
- `BINARY_FLOAT`
- `NUMBER`
- Character data types
- All other built-in data types

Data Conversion

Generally an expression cannot contain values of different data types. For example, an expression cannot multiply 5 by 10 and then add 'JAMES'. However, Oracle supports both implicit and explicit conversion of values from one data type to another.

Implicit and Explicit Data Conversion

Oracle recommends that you specify explicit conversions, rather than rely on implicit or automatic conversions, for these reasons:

- SQL statements are easier to understand when you use explicit data type conversion functions.
- Implicit data type conversion can have a negative impact on performance, especially if the data type of a column value is converted to that of a constant rather than the other way around.
- Implicit conversion depends on the context in which it occurs and may not work the same way in every case. For example, implicit conversion from a datetime value to a `VARCHAR2` value may return an unexpected year depending on the value of the `NLS_DATE_FORMAT` parameter.
- Algorithms for implicit conversion are subject to change across software releases and among Oracle products. Behavior of explicit conversions is more predictable.
- If implicit data type conversion occurs in an index expression, then Oracle Database might not use the index because it is defined for the pre-conversion data type. This can have a negative impact on performance.

Implicit Data Conversion

Oracle Database automatically converts a value from one data type to another when such a conversion makes sense.

Table 2-10 (#g195937) is a matrix of Oracle implicit conversions. The table shows all possible conversions, without regard to the direction of the conversion or the context in which it is made. The rules governing these details follow the table.

Table 2-10 Implicit Type Conversion Matrix

	CHAR	VARCHAR2	NCHAR	NVARCHAR2	DATE	DATETIME/INTERVAL	NUMBER	BINARY_FLOAT	BINARY_DOUBLE	LONG	RAW	ROWID	CLOB	BLOB	NCLOB
CHAR	--	X	X	X	X	X	X	X	X	X	X	X	X	X	X
VARCHAR2	X	--	X	X	X	X	X	X	X	X	X	X	X	--	X

	CHAR	VARCHAR2	NCHAR	NVARCHAR2	DATE	DATETIME/INTERVAL	NUMBER	BINARY_FLOAT	BINARY_DOUBLE	LONG	RAW	ROWID	CLOB	BLOB	NCLOB
NCHAR	X	X	--	X	X	X	X	X	X	X	X	X	X	--	X
NVARCHAR2	X	X	X	--	X	X	X	X	X	X	X	X	X	--	X
DATE	X	X	X	X	--	--	--	--	--	--	--	--	--	--	--
DATETIME/INTERVAL	X	X	X	X	--	--	--	--	--	X	--	--	--	--	--
NUMBER	X	X	X	X	--	--	--	X	X	--	--	--	--	--	--
BINARY_FLOAT	X	X	X	X	--	--	X	--	X	--	--	--	--	--	--

	CHAR	VARCHAR2	NCHAR	NVARCHAR2	DATE	DATETIME/INTERVAL	NUMBER	BINARY_FLOAT	BINARY_DOUBLE	LONG	RAW	ROWID	CLOB	BLOB	NCLOB
BINARY_DOUBLE	X	X	X	X	--	--	X	X	--	--	--	--	--	--	--
LONG	X	X	X	X	--	X[Foot 1] (#sthref299)	--	--	--	--	X	--	X	--	X
RAW	X	X	X	X	--	--	--	--	--	X	--	--	--	X	--
ROWID	X	X	X	X	--	--	--	--	--	--	--	--	--	--	--
CLOB	X	X	X	X	--	--	--	--	--	X	--	--	--	--	X
BLOB	--	--	--	--	--	--	--	--	--	--	X	--	--	--	--
NCLOB	X	X	X	X	--	--	--	--	--	X	--	--	X	--	--

Footnote 1 You cannot convert `LONG` to `INTERVAL` directly, but you can convert `LONG` to `VARCHAR2` using `TO_CHAR(`*`interval`*`)`, and then convert the resulting `VARCHAR2` value to `INTERVAL`.

The following rules govern implicit data type conversions:

- During `INSERT` and `UPDATE` operations, Oracle converts the value to the data type of the affected column.

- During SELECT FROM operations, Oracle converts the data from the column to the type of the target variable.
- When manipulating numeric values, Oracle usually adjusts precision and scale to allow for maximum capacity. In such cases, the numeric data type resulting from such operations can differ from the numeric data type found in the underlying tables.
- When comparing a character value with a numeric value, Oracle converts the character data to a numeric value.
- Conversions between character values or NUMBER values and floating-point number values can be inexact, because the character types and NUMBER use decimal precision to represent the numeric value, and the floating-point numbers use binary precision.
- When converting a CLOB value into a character data type such as VARCHAR2, or converting BLOB to RAW data, if the data to be converted is larger than the target data type, then the database returns an error.
- During conversion from a timestamp value to a DATE value, the fractional seconds portion of the timestamp value is truncated. This behavior differs from earlier releases of Oracle Database, when the fractional seconds portion of the timestamp value was rounded.
- Conversions from BINARY_FLOAT to BINARY_DOUBLE are exact.
- Conversions from BINARY_DOUBLE to BINARY_FLOAT are inexact if the BINARY_DOUBLE value uses more bits of precision that supported by the BINARY_FLOAT.
- When comparing a character value with a DATE value, Oracle converts the character data to DATE.
- When you use a SQL function or operator with an argument of a data type other than the one it accepts, Oracle converts the argument to the accepted data type.
- When making assignments, Oracle converts the value on the right side of the equal sign (=) to the data type of the target of the assignment on the left side.
- During concatenation operations, Oracle converts from noncharacter data types to CHAR or NCHAR.
- During arithmetic operations on and comparisons between character and noncharacter data types, Oracle converts from any character data type to a numeric, date, or rowid, as appropriate. In arithmetic operations between CHAR/VARCHAR2 and NCHAR/NVARCHAR2, Oracle converts to a NUMBER.
- Most SQL character functions are enabled to accept CLOBs as parameters, and Oracle performs implicit conversions between CLOB and character types. Therefore, functions that are not yet enabled for CLOBs can accept CLOBs through implicit conversion. In such cases, Oracle converts the CLOBs to CHAR or VARCHAR2 before the function is invoked. If the CLOB is larger than 4000 bytes, then Oracle converts only the first 4000 bytes to CHAR.
- When converting RAW or LONG RAW data to or from character data, the binary data is represented in hexadecimal form, with one hexadecimal character representing every four bits of RAW data. Refer to "RAW and LONG RAW Data Types" (sql_elements001.htm#i46018) for more information.
- Comparisons between CHAR and VARCHAR2 and between NCHAR and NVARCHAR2 types may entail different character sets. The default direction of conversion in such cases is from the database character set to the national character set. Table 2-11 (#g196434) shows the direction of implicit conversions between different

/

character types.

Table 2-11 Conversion Direction of Different Character Types

	to CHAR	to VARCHAR2	to NCHAR	to NVARCHAR2
from CHAR	--	VARCHAR2	NCHAR	NVARCHAR2
from VARCHAR2	VARCHAR2	--	NVARCHAR2	NVARCHAR2
from NCHAR	NCHAR	NCHAR	--	NVARCHAR2
from NVARCHAR2	NVARCHAR2	NVARCHAR2	NVARCHAR2	--

User-defined types such as collections cannot be implicitly converted, but must be explicitly converted using CAST ... MULTISET.

Implicit Data Conversion Examples

Text Literal Example
The text literal '10' has data type CHAR. Oracle implicitly converts it to the NUMBER data type if it appears in a numeric expression as in the following statement:

```
SELECT salary + '10' FROM employees;
```

Character and Number Values Example
When a condition compares a character value and a NUMBER value, Oracle implicitly converts the character value to a NUMBER value, rather than converting the NUMBER value to a character value. In the following statement, Oracle implicitly converts '200' to 200:

```
SELECT last_name FROM employees WHERE employee_id = '200';
```

Date Example
In the following statement, Oracle implicitly converts '24-JUN-06' to a DATE value using the default date format 'DD-MON-YY':

```
SELECT last_name FROM employees WHERE hire_date = '24-JUN-06';
```

Explicit Data Conversion

You can explicitly specify data type conversions using SQL conversion functions. Table 2-12 (#g195600) shows SQL functions that explicitly convert a value from one data type to another.

You cannot specify `LONG` and `LONG RAW` values in cases in which Oracle can perform implicit data type conversion. For example, `LONG` and `LONG RAW` values cannot appear in expressions with functions or operators. Refer to "LONG Data Type" (sql_elements001.htm#i45885) for information on the limitations on `LONG` and `LONG RAW` data types.

Table 2-12 Explicit Type Conversions

	to CHAR, VARCHAR2, NCHAR,NVARCHAR2	to NUMBER	to Datetime/Interval	to RAW	to ROWID	to LONG, LONG RAW	to CLOB, NCLOB,BLOB	to BINARY_FLOAT	to BINARY_DOUBLE

from CHAR, VARCHAR2, NCHAR, NVARCHAR2	`TO_CHAR (char.)`
	`TO_NCHAR (char.)`
		`TO_NUMBER`	`TO_DATE`
			`TO_TIMESTAMP`
			`TO_TIMESTAMP_TZ`
			`TO_YMINTERVAL`
			`TO_DSINTERVAL`
				`HEXTORAW`	`CHARTO-=ROWID`	`--`	`TO_CLOB`
							`TO_NCLOB`
								`TO_BINARY_FLOAT`	`TO_BINARY_DOUBLE`

from NUMBER	`TO_CHAR (number)`
	`TO_NCHAR (number)`
		`--`	`TO_DATE`
			`NUMTOYM-INTERVAL`
			`NUMTODS-INTERVAL`
				`--`	`--`	`--`	`--`	`TO_BINARY_FLOAT`	`TO_BINARY_DOUBLE`

	to CHAR, VARCHAR2, NCHAR,NVARCHAR2	to NUMBER	to Datetime/Interval	to RAW	to ROWID	to LONG, LONG RAW	to CLOB, NCLOB,BLOB	to BINARY_FLOAT	to BINARY_DOUBLE
from Datetime/ Interval	TO_CHAR (date) TO_NCHAR (datetime)	--	--	--	--	--	--	--	--
from RAW	RAWTOHEX RAWTONHEX	--	--	--	--	--	TO_BLOB	--	--
from ROWID	ROWIDTOCHAR	--	--	--	--	--	--	--	--
from LONG / LONG RAW	--	--	--	--	--	--	TO_LOB	--	--
from CLOB, NCLOB, BLOB	TO_CHAR TO_NCHAR	--	--	--	--	--	TO_CLOB TO_NCLOB	--	--
from CLOB, NCLOB, BLOB	TO_CHAR TO_NCHAR	--	--	--	--	--	TO_CLOB TO_NCLOB	--	--

	to CHAR, VARCHAR2, NCHAR,NVARCHAR2	to NUMBER	to Datetime/Interval	to RAW	to ROWID	to LONG, LONG RAW	to CLOB, NCLOB,BLOB	to BINARY_FLOAT	to BINARY_DOUBLE

from BINARY_FLOAT	`TO_CHAR (char.)`
	
	`TO_NCHAR (char.)`
		`TO_NUMBER`	--	--	--	--	--	`TO_BINARY_FLOAT`	`TO_BINARY_DOUBLE`

from BINARY_DOUBLE	`TO_CHAR (char.)`
	
	`TO_NCHAR (char.)`
		`TO_NUMBER`	--	--	--	--	--	`TO_BINARY_FLOAT`	`TO_BINARY_DOUBLE`

See Also:

"Conversion Functions" (functions002.htm#i88892) for details on all of the explicit conversion functions

Security Considerations for Data Conversion

When a datetime value is converted to text, either by implicit conversion or by explicit conversion that does not specify a format model, the format model is defined by one of the globalization session parameters. Depending on the source data type, the parameter name is `NLS_DATE_FORMAT`, `NLS_TIMESTAMP_FORMAT`, or `NLS_TIMESTAMP_TZ_FORMAT`. The values of these parameters can be specified in the client environment or in an `ALTER SESSION` statement.

The dependency of format models on session parameters can have a negative impact on database security when conversion without an explicit format model is applied to a datetime value that is being concatenated to text of a dynamic SQL statement. Dynamic SQL statements are those statements whose text is concatenated from fragments before being passed to a database for execution. Dynamic SQL is frequently associated with the built-in PL/SQL package `DBMS_SQL` or with the PL/SQL statement `EXECUTE IMMEDIATE`, but these are not the only places where dynamically constructed SQL text may be passed as argument. For example:

```
EXECUTE IMMEDIATE 'SELECT last_name FROM employees WHERE hire_date > ''' || start_date
|| '''';
```

where *start_date* has the data type DATE.

In the above example, the value of *start_date* is converted to text using a format model specified in the session parameter NLS_DATE_FORMAT. The result is concatenated into SQL text. A datetime format model can consist simply of literal text enclosed in double quotation marks. Therefore, any user who can explicitly set globalization parameters for a session can decide what text is produced by the above conversion. If the SQL statement is executed by a PL/SQL procedure, the procedure becomes vulnerable to SQL injection through the session parameter. If the procedure runs with definer's rights, with higher privileges than the session itself, the user can gain unauthorized access to sensitive data.

See Also:

Oracle Database PL/SQL Language Reference (../LNPLS/dynamic.htm#LNPLS01109) for further examples and for recommendations on avoiding this security risk

Note:

This security risk also applies to middle-tier applications that construct SQL text from datetime values converted to text by the database or by OCI datetime functions. Those applications are vulnerable if session globalization parameters are obtained from a user preference.

Implicit and explicit conversion for numeric values may also suffer from the analogous problem, as the conversion result may depend on the session parameter NLS_NUMERIC_CHARACTERS. This parameter defines the decimal and group separator characters. If the decimal separator is defined to be the quotation mark or the double quotation mark, some potential for SQL injection emerges.

See Also:

- *Oracle Database Globalization Support Guide* (../NLSPG/ch3globenv.htm#NLSPG003) for detailed descriptions of the session globalization parameters
- "Format Models" (sql_elements004.htm#i34510) for information on the format models