SQL DATE Type
Type Conversion

SQL DATE Type

Oracle's SQL has a single data type to represent both dates and time: DATE.

The DATE data type is unique in how it is used.

- A value of DATE type stored in an Oracle database is an number.
- A value of DATE type displayed to the user is a string.

Basically, Oracle associates a *display format* with values of the DATE type.

- Values of DATE type get converted into a string according to the *display format* to be shown to users.
- When a DATE value is read as input (in an `INSERT INTO` statement, for example), Oracle assumes that it reads a string. An attempt is made to convert the string into number according to the *display format*. If the value matches the *display format*, it is successfully converted and stored. If it does not, an error is reported.

Oracle's default *display format* for a DATE value is

'DD-MON-YYYY' or 'DD-MON-YY',

where DD is days, `MON` is the three letter abbreviation of the month (e.g., 'JAN', 'FEB', etc.), and YYYY and YY are, respectively, the full year (e.g., 1984) and the two last digits of a year (e.g., 07).

Reading/Inserting DATE data in other formats

Often, one needs to input DATE values in formats different from the default. For example, the default *display format* does not include hours, minutes and seconds. Similarly, it may be important to convert data stored in an Oracle database to a DATE format that is different from the default.

These operations are achieved using two SQL's built-in functions: `TO_DATE()` and `TO_CHAR()`.

TO_DATE(*date,format*): takes as input a string representing the value of type DATE (*date*) and a format specification (format string), and produces the DATE value from them for storage in the database.

TO_CHAR(*date,format*): takes as input a DATE value (*date*, typically, a table column name) and a format specification (format string), and produces a string representing the DATE value in the desired format.

To insert a non-default DATE value into a table: in the `INSERT INTO` statement use `TO_DATE()` function on the value being inserted and supply the format, the date is in.

To see the DATE value displayed in a non-default format in a report: use `TO_CHAR()` function in the `SELECT` clause of the `SELECT` statement, with the DATE column and a format as parameters.

See examples below.

Format Specifications

.

A *date format specification* is a sting which intermixes includes special identifiers for various components of the DATE type vlaues with regular strings. The table of formatting components is provided for your convenience below.

specification	explanation	examples
`MM`	Numeric month	01, ..., 12
`MON`	Three-letter abbreviation for month name	`'JAN'`, ...,`'DEC'`
`MONTH`	Full month name	`'JANUARY`,...,`'DECEMBER'`
`DD`	Day of the month	1,..., 31
`DY`	Three-letter abbreviation for day of the week	`'MON'`, ...,`'SUN'`
`YYYY`	4-digit year	..., 1998, 1999, 2000, 2001, ...
`YY`	Last 2 digits of the year	01,..., 99
`RR`	Like `YY`, but the two digits are "rounded" to a year in the range 1950 to 2049	01,..., 99
`HH`	Twelve-hour hour of day	1,...,12
`HH24`	Twenty-four-hour hour of day	0,...,23
`MI`	Minute	0,...,59
`SS`	Second	0,...,59
`AM`, `PM`	Meridian indicator	`'AM'`, `'PM'`

Examples

Here are some examples of how to insert and output DATE values.

```
SQL> CREATE TABLE Example (
  2  Id INT PRIMARY KEY,
  3  Month DATE,
  4  BDay DATE,
  5  Time DATE);

Table created.

SQL> INSERT INTO Example VALUES(1, '1-Jun-2007','3-Oct-2007', TO_DATE('1:33','HH:MI'));
```

```
1 row created.

SQL> SELECT * FROM Example;

ID MONTH     BDAY      TIME
---------- --------- ---------
1 01-JUN-07 03-OCT-07 01-OCT-07
```

In this sequence, we create a simple table, with three DATE columns and insert one row in it. The first two DATE values are inserted in the default display format, while the third column gets a value expressed as hours and minutes. When the information in the table is displayed back using default display format, the hours-minutes nature of the third DATE value (`Example.Time`) is obscured. It is not, however, lost, as evidenced by the following continuation of the session.

```
SQL> SELECT Id, TO_CHAR(Month, 'MONTH') AS Month, BDAY, TO_CHAR(Time, 'HH:MI') AS Time
  2  FROM Example;

 ID MONTH     BDAY      TIME
 ---------- --------- ---------
1 JUNE      03-OCT-07 01:33


SQL>  SELECT Id, TO_CHAR(Month, '== MON ==')  AS Month,
  2  TO_CHAR(BDAY,'MON/DD/YYYY') AS Birthday,
  3  TO_CHAR(Time, '*** HH24:MI:SS ***') AS Time
  4  FROM Example;

ID   MONTH       BIRTHDAY    TIME
---  --------    ---------   ----------------
1    == JUN ==   OCT/03/2007 *** 01:33:00 ***


SQL>  SELECT TO_CHAR(BDay, 'DY') AS DayOfWeek
  2  FROM Example;

DAY
---
WED
```

The first query, shows the contents of the `Example` table witht the `Month` and `Time` values formatted to show the month and the time in hours-minutes, as is presumed by the meaning of the table. In the second query, we use more complex formatting instructions, surrounding the month and time information with special characters. We also override the display format for the `Birthday` column.Finally, in the third query, we display infromation about the birthday as a day of the week.

Data Conversion and String Manipulation in SQL

In addition to `TO_DATE()` and `TO_CHAR()` function, SQL provides a number of other built-in function that make working with column values easier. In particular,

- `TO_CHAR()` function can be used to convert **any** value into a string. The unformatted version of the function is `TO_CHAR(`*value*`)`. E.g., `TO_CHAR(120)` returns '120'.

- `TO_NUMBER()` function can be used to convert strings to numbers. The format is `TO_NUMBER(`*`value`*`)`. E.g., `TO_NUMBER('123')` returns 123, and `TO_NUMBER('1.45')` returns 1.45.

In addition to data type conversion functions, a number of Oracle SQL built-in functions are devoted to string manipulation. A list of most popular functions and operations is below.

Function/Operation	Explanation	Example
||	String concatenation operation	`'a'`|| `'b'` yields `'ab'`
`lpad(string, length, chars)`	pad input string to the left with chars	`lpad('a',3,'b')` returns `'bba'`
`lpad(string, length)`	pad input to the left with spaces	`lpad('a',3)` returns `'  a'`
`rpad(string, length, chars)`	pad input string to the right	`rpad('a',3,'b')` returns `'abb'`
`rpad(string, length)`	pad input to the right with spaces	`rpad('a',3)` returns `'a  '`
`ltrim(string,chars)`	removes from the left of string any character from the list	`ltrim('abca','ab')` returns `'ca'`
`ltrim(string)`	removes leading spaces	`ltrim(' ab')` returns `'ab'`
`rtrim(string,chars)`	removes from the right of string any character from the list	`rtrim('abca','ac')` returns `'ab'`
`rtrim(string)`	removes trailing spaces	`ltrim('ab ')` returns `'ab'`
`lower(string)`	converts to lowercase	`lower('BaT')` returns `'bat'`
`upper(string)`	converts to UPPERCASE	`upper('BaT')` returns `'BAT'`
`initcap(string)`	capitalizes first letter	`lower('bat')` returns `'Bat'`
`substr(string,start,n)`	returns substring of length n	`substr('abcde',2,3)` returns `'bcd'`
`length(string)`	returns length of string	`length('abcde')` returns 5

Three more functions require more explanations.

- `translate(string, from, to)` replaces all occurrences of the characters in `from` in the input string `string` with the matching characters in `to` (if no matching character - omit). Example: `translate('database', 'dat', 'cod')` returns `'codobose'`.

- `decode(string, if1String, then1String,...,ifNString, thenNString, elseString)` checks to see if the value of the input string `string` matches with any of the `if1String`,...,`ifNString`, and if it does, replaces the input `string` with the matching `theniString`. For example, here is a way to ensure that my last name will be spelled correctly if one of the three common misspellings is used.

```
decode(LastName, 'Dekhtiar', 'Dekhtyar', 'Dekytar', 'Dekhtyar', 'Dekhtyer', 'Dekhtyar', LastName)
```

- `instr(string, chars, start, n)` searches for the substring `chars` in the input string `string` starting with the (optional) position `start` and returns the the position of the `n`th occurrence (or 0 if not found). Examples: `instr('database', 'a')` returns 2; `instr('database', 'a', 3)` returns 4; `instr('database,'a',1,3)` returns 6.