Oracle SQL

Data Type

**CHAR Datatype**

The CHAR datatype stores fixed-length character strings. When you create a table with a CHAR column, you must specify a string length (in bytes or characters) between 1 and 2000 bytes for the CHAR column width. The default is 1 byte. Oracle then guarantees that:

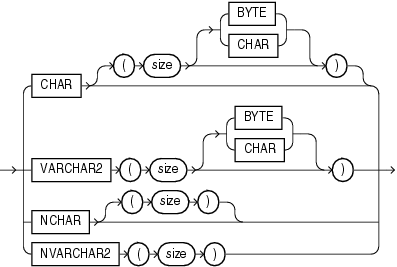
* When you insert or update a row in the table, the value for the CHAR column has the fixed length.
* If you give a shorter value, then the value is blank-padded to the fixed length.
* If a value is too large, Oracle Database returns an error.

Oracle Database compares CHAR values using blank-padded comparison semantics.

### VARCHAR2 and VARCHAR Datatypes

The VARCHAR2 datatype stores variable-length character strings. When you create a table with a VARCHAR2 column, you specify a maximum string length (in bytes or characters) between 1 and 4000 bytes for the VARCHAR2 column.

For example, assume you declare a column VARCHAR2 with a maximum size of 50 characters. In a single-byte character set, if only 10 characters are given for the VARCHAR2 column value in a particular row, the column in the row's row piece stores only the 10 characters (10 bytes), not 50.



### NUMBER Datatype

The NUMBER datatype stores fixed and floating-point numbers.

For numeric columns, you can specify the column as:

column\_name NUMBER

Optionally, you can also specify a precision (total number of digits) and scale (number of digits to the right of the decimal point):

column\_name NUMBER (precision, scale)

If a precision is not specified, the column stores values as given. If no scale is specified, the scale is zero.

| **Input Data** | **Specified As** | **Stored As** |
| --- | --- | --- |
| 7,456,123.89 | NUMBER | 7456123.89 |
| 7,456,123.89 | NUMBER(\*,1) | [7456123.9](http://support.oracle.com/epmos/faces/DocumentDisplay?id=7456123.9) |
| 7,456,123.89 | NUMBER(9) | 7456124 |
| 7,456,123.89 | NUMBER(9,2) | 7456123.89 |
| 7,456,123.89 | NUMBER(9,1) | [7456123.9](http://support.oracle.com/epmos/faces/DocumentDisplay?id=7456123.9) |
| 7,456,123.89 | NUMBER(6) | (not accepted, exceeds precision) |
| 7,456,123.89 | NUMBER(7,-2) | 7456100 |

If you specify a negative scale, Oracle Database rounds the actual data to the specified number of places to the left of the decimal point. For example, specifying (7,-2) means Oracle Database rounds to the nearest hundredths,

## DATE Datatype

The DATE datatype stores point-in-time values (dates and times) in a table.

the standard Oracle date format is DD-MON-YY, as follows:

'13-NOV-92'

To enter dates that are not in standard Oracle date format, use the TO\_DATE function with a format mask:

TO\_DATE ('November 13, 1992', 'MONTH DD, YYYY')

==================================================================================

Oracle Database stores time in 24-hour format—HH:MI:SS. By default, the time in a date field is 00:00:00 A.M. (midnight) if no time portion is entered. In a time-only entry, the date portion defaults to the first day of the current month.

Oracle Database provides daylight savings support for DATETIME datatypes in the server. You can insert and query DATETIME values based on local time in a specific region. The DATETIME datatypes TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE are time-zone aware.

| **Datatype** | **Time Zone** | **Fractional Seconds** |
| --- | --- | --- |
| DATE | No | No |
| TIMESTAMP | No | Yes |
| TIMESTAMP WITH TIME ZONE | Explicit | Yes |
| TIMESTAMP WITH LOCAL TIME ZONE | Relative | Yes |

#### TIMESTAMP Datatype

The TIMESTAMP datatype is an extension of the DATE datatype. It stores year, month, day, hour, minute, and second values. It also stores fractional seconds, which are not stored by the DATE datatype.

Specify the TIMESTAMP datatype as follows:

TIMESTAMP [(fractional\_seconds\_precision)]

fractional\_seconds\_precision is optional and specifies the number of digits in the fractional part of the SECOND datetime field. It can be a number in the range 0 to 9. The default is 6.

For example, '26-JUN-02 09:39:16.78' shows 16.78 seconds. The fractional seconds precision is 2 because there are 2 digits in '78'.

You can specify the TIMESTAMP literal in a format like the following:

TIMESTAMP 'YYYY-MM-DD HH24:MI:SS.FF'

Using the example format, specify TIMESTAMP as a literal as follows:

TIMESTAMP '1997-01-31 09:26:50.12'

The value of NLS\_TIMESTAMP\_FORMAT initialization parameter determines the timestamp format when a character string is converted to the TIMESTAMP datatype. NLS\_DATE\_LANGUAGE determines the language used for character data such as MON.

#### TIMESTAMP WITH TIME ZONE Datatype

TIMESTAMP WITH TIME ZONE is a variant of TIMESTAMP that includes a time zone offset or time zone region name in its value. The time zone offset is the difference (in hours and minutes) between local time and UTC (Coordinated Universal Time, formerly Greenwich Mean Time). Specify the TIMESTAMP WITH TIME ZONE datatype as follows:

TIMESTAMP [(fractional\_seconds\_precision)] WITH TIME ZONE

fractional\_seconds\_precision is optional and specifies the number of digits in the fractional part of the SECOND datetime field.

You can specify TIMESTAMP WITH TIME ZONE as a literal as follows:

TIMESTAMP '1997-01-31 09:26:56.66 +02:00'

Two TIMESTAMP WITH TIME ZONE values are considered identical if they represent the same instant in UTC, regardless of the TIME ZONE offsets stored in the data. For example, the following expressions have the same value:

TIMESTAMP '1999-01-15 8:00:00 -8:00'

TIMESTAMP '1999-01-15 11:00:00 -5:00'

You can replace the UTC offset with the TZR (time zone region) format element. The following expression specifies US/Pacific for the time zone region:

TIMESTAMP '1999-01-15 8:00:00 US/Pacific'

To eliminate the ambiguity of boundary cases when the time switches from Standard Time to Daylight Saving Time, use both the TZR format element and the corresponding TZD format element. The TZD format element is an abbreviation of the time zone region with Daylight Saving Time information included. Examples are PST for US/Pacific standard time and PDT for US/Pacific daylight time. The following specification ensures that a Daylight Saving Time value is returned:

TIMESTAMP '1999-10-29 01:30:00 US/Pacific PDT'

If you do not add the TZD format element, and the datetime value is ambiguous, then Oracle returns an error if you have the ERROR\_ON\_OVERLAP\_TIME session parameter set to TRUE. If ERROR\_ON\_OVERLAP\_TIME is set to FALSE (the default value), then Oracle interprets the ambiguous datetime as Standard Time.

The default date format for the TIMESTAMP WITH TIME ZONE datatype is determined by the value of the NLS\_TIMESTAMP\_TZ\_FORMAT initialization parameter.

#### TIMESTAMP WITH LOCAL TIME ZONE Datatype

TIMESTAMP WITH LOCAL TIME ZONE is another variant of TIMESTAMP. It differs from TIMESTAMP WITH TIME ZONE as follows: data stored in the database is normalized to the database time zone, and the time zone offset is not stored as part of the column data. When users retrieve the data, Oracle returns it in the users' local session time zone. The time zone offset is the difference (in hours and minutes) between local time and UTC (Coordinated Universal Time, formerly Greenwich Mean Time).

Specify the TIMESTAMP WITH LOCAL TIME ZONE datatype as follows:

TIMESTAMP [(fractional\_seconds\_precision)] WITH LOCAL TIME ZONE

fractional\_seconds\_precision is optional and specifies the number of digits in the fractional part of the SECOND datetime field.

There is no literal for TIMESTAMP WITH LOCAL TIME ZONE, but TIMESTAMP literals and TIMESTAMP WITH TIME ZONE literals can be inserted into a TIMESTAMP WITH LOCAL TIME ZONE column.

The default date format for TIMESTAMP WITH LOCAL TIME ZONE is determined by the value of the NLS\_TIMESTAMP\_FORMAT initialization parameter.

#### Inserting Values into Datetime Datatypes

You can insert values into a datetime column in the following ways:

* Insert a character string whose format is based on the appropriate NLS format value
* Insert a literal
* Insert a literal for which implicit conversion is performed
* Use the TO\_TIMESTAMP, TO\_TIMESTAMP\_TZ, or TO\_DATE SQL function

The following examples show how to insert data into datetime datatypes.

Example 4-1 Inserting Data into a DATE Column

Set the date format.

SQL> ALTER SESSION SET NLS\_DATE\_FORMAT='DD-MON-YYYY HH24:MI:SS';

Create a table table\_dt with columns c\_id and c\_dt. The c\_id column is of NUMBER datatype and helps to identify the method by which the data is entered. The c\_dt column is of DATE datatype.

SQL> CREATE TABLE table\_dt (c\_id NUMBER, c\_dt DATE);

Insert a date as a character string.

SQL> INSERT INTO table\_dt VALUES(1, '01-JAN-2003');

Insert the same date as a DATE literal.

SQL> INSERT INTO table\_dt VALUES(2, DATE '2003-01-01');

Insert the date as a TIMESTAMP literal. Oracle drops the time zone information.

SQL> INSERT INTO table\_dt VALUES(3, TIMESTAMP '2003-01-01 00:00:00 US/Pacific');

Insert the date with the TO\_DATE function.

SQL> INSERT INTO table\_dt VALUES(4, TO\_DATE('01-JAN-2003', 'DD-MON-YYYY'));

Display the data.

SQL> SELECT \* FROM table\_dt;

C\_ID C\_DT

---------- --------------------

1 01-JAN-2003 00:00:00

2 01-JAN-2003 00:00:00

3 01-JAN-2003 00:00:00

4 01-JAN-2003 00:00:00

Example 4-2 Inserting Data into a TIMESTAMP Column

Set the timestamp format.

SQL> ALTER SESSION SET NLS\_TIMESTAMP\_FORMAT='DD-MON-YY HH:MI:SSXFF';

Create a table table\_ts with columns c\_id and c\_ts. The c\_id column is of NUMBER datatype and helps to identify the method by which the data is entered. The c\_ts column is of TIMESTAMP datatype.

SQL> CREATE TABLE table\_ts(c\_id NUMBER, c\_ts TIMESTAMP);

Insert a date and time as a character string.

SQL> INSERT INTO table\_ts VALUES(1, '01-JAN-2003 2:00:00');

Insert the same date and time as a TIMESTAMP literal.

SQL> INSERT INTO table\_ts VALUES(2, TIMESTAMP '2003-01-01 2:00:00');

Insert the same date and time as a TIMESTAMP WITH TIME ZONE literal. Oracle converts it to a TIMESTAMP value, which means that the time zone information is dropped.

SQL> INSERT INTO table\_ts VALUES(3, TIMESTAMP '2003-01-01 2:00:00 -08:00');

Display the data.

SQL> SELECT \* FROM table\_ts;

C\_ID C\_TS

---------- -----------------------------

1 01-JAN-03 02:00:00.000000 AM

2 01-JAN-03 02:00:00.000000 AM

3 01-JAN-03 02:00:00.000000 AM

Note that the three methods result in the same value being stored.

Example 4-3 Inserting Data into the TIMESTAMP WITH TIME ZONE Datatype

Set the timestamp format.

SQL> ALTER SESSION SET NLS\_TIMESTAMP\_TZ\_FORMAT='DD-MON-RR HH:MI:SSXFF AM TZR';

Set the time zone to '-07:00'.

SQL> ALTER SESSION SET TIME\_ZONE='-7:00';

Create a table table\_tstz with columns c\_id and c\_tstz. The c\_id column is of NUMBER datatype and helps to identify the method by which the data is entered. The c\_tstz column is of TIMESTAMP WITH TIME ZONE datatype.

SQL> CREATE TABLE table\_tstz (c\_id NUMBER, c\_tstz TIMESTAMP WITH TIME ZONE);

Insert a date and time as a character string.

SQL> INSERT INTO table\_tstz VALUES(1, '01-JAN-2003 2:00:00 AM -07:00');

Insert the same date and time as a TIMESTAMP literal. Oracle converts it to a TIMESTAMP WITH TIME ZONE literal, which means that the session time zone is appended to the TIMESTAMP value.

SQL> INSERT INTO table\_tstz VALUES(2, TIMESTAMP '2003-01-01 2:00:00');

Insert the same date and time as a TIMESTAMP WITH TIME ZONE literal.

SQL> INSERT INTO table\_tstz VALUES(3, TIMESTAMP '2003-01-01 2:00:00 -8:00');

Display the data.

SQL> SELECT \* FROM table\_tstz;

C\_ID C\_TSTZ

---------- ------------------------------------

1 01-JAN-03 02:00.00:000000 AM -07:00

2 01-JAN-03 02:00:00.000000 AM -07:00

3 01-JAN-03 02:00:00.000000 AM -08:00

Note that the time zone is different for method 3, because the time zone information was specified as part of the TIMESTAMP WITH TIME ZONE literal.

Example 4-4 Inserting Data into the TIMESTAMP WITH LOCAL TIME ZONE Datatype

Consider data that is being entered in Denver, Colorado, U.S.A., whose time zone is UTC-7.

SQL> ALTER SESSION SET TIME\_ZONE='-07:00';

Create a table table\_tsltz with columns c\_id and c\_tsltz. The c\_id column is of NUMBER datatype and helps to identify the method by which the data is entered. The c\_tsltz column is of TIMESTAMP WITH LOCAL TIME ZONE datatype.

SQL> CREATE TABLE table\_tsltz (c\_id NUMBER, c\_tsltz TIMESTAMP WITH LOCAL TIME ZONE);

Insert a date and time as a character string.

SQL> INSERT INTO table\_tsltz VALUES(1, '01-JAN-2003 2:00:00');

Insert the same data as a TIMESTAMP WITH LOCAL TIME ZONE literal.

SQL> INSERT INTO table\_tsltz VALUES(2, TIMESTAMP '2003-01-01 2:00:00');

Insert the same data as a TIMESTAMP WITH TIME ZONE literal. Oracle converts the data to a TIMESTAMP WITH LOCAL TIME ZONE value. This means the time zone that is entered (-08:00) is converted to the session time zone value (-07:00).

SQL> INSERT INTO table\_tsltz VALUES(3, TIMESTAMP '2003-01-01 2:00:00 -08:00');

Display the data.

SQL> SELECT \* FROM table\_tsltz;

C\_ID C\_TSLTZ

---------- ------------------------------------

1 01-JAN-03 02.00.00.000000 AM

2 01-JAN-03 02.00.00.000000 AM

3 01-JAN-03 03.00.00.000000 AM

Note that the information that was entered as UTC-8 has been changed to the local time zone, changing the hour from 2 to 3.

See Also:

["Datetime SQL Functions"](http://docs.oracle.com/cd/B19306_01/server.102/b14225/ch4datetime.htm#i1006333) for more information about the TO\_TIMESTAMP or TO\_TIMESTAMP\_TZ SQL functions

#### Choosing a TIMESTAMP Datatype

Use the TIMESTAMP datatype when you need a datetime value without locale information. For example, you can store information about the times when workers punch a timecard in and out of their assembly line workstations. The TIMESTAMP datatype uses 7 or 11 bytes of storage.

Use the TIMESTAMP WITH TIME ZONE datatype when the application is used across time zones. Consider a banking company with offices around the world. It records a deposit to an account at 11 a.m. in London and a withdrawal of the same amount from the account at 9 a.m. in New York. The money is in the account for three hours. Unless time zone information is stored with the account transactions, it appears that the account is overdrawn from 9 a.m. to 11 a.m.

The TIMESTAMP WITH TIME ZONE datatype requires 13 bytes of storage, or two more bytes of storage than the TIMESTAMP and TIMESTAMP WITH LOCAL TIME ZONE datatypes because it stores time zone information. The time zone is stored as an offset from UTC or as a time zone region name. The data is available for display or calculations without additional processing. A TIMESTAMP WITH TIME ZONE column cannot be used as a primary key. If an index is created on a TIMESTAMP WITH TIME ZONE column, it becomes a function-based index.

The TIMESTAMP WITH LOCAL TIME ZONE datatype stores the timestamp without time zone information. It normalizes the data to the database time zone every time the data is sent to and from a client. It requires 11 bytes of storage.

The TIMESTAMP WITH LOCAL TIME ZONE datatype is appropriate when the original time zone is of no interest, but the relative times of events are important. Consider the transactions described in the previous banking example. Suppose the data is recorded using the TIMESTAMP WITH LOCAL TIME ZONE datatype. If the database time zone of the bank is set to Asia/Hong\_Kong, then an employee in Hong Kong who displays the data would see that the deposit was made at 7 p.m. and the withdrawal was made at 10 p.m. If the same data is displayed in London, it would show that the deposit was made at 11 a.m. and the withdrawal was made at 2 p.m. The three-hour difference is preserved, but the time zone/region of the original transaction is not. Because of this, the actual time of the transaction can be interpreted differently depending on the time zone/region from which the information is retrieved. For example, in London, the transactions appear to be conducted within business hours, in Hong Kong, they do not.

Note that, because the original time zone region of the time data is not preserved in the TIMESTAMP WITH LOCAL TIME ZONE data type, time data referring to times from regions such as Brazil and Israel, regions that update their Daylight Savings Transition dates frequently and at irregular periods, may be inaccurate. If time information from these regions is key to your application, you may wish to consider using one of the other datetime types.