



Database Globalization Support Guide

4 Datetime Data Types and Time Zone Support

This chapter includes the following topics:

- [Overview of Datetime and Interval Data Types and Time Zone Support](#)
- [Datetime and Interval Data Types](#)
- [Datetime and Interval Arithmetic and Comparisons](#)
- [Datetime SQL Functions](#)
- [Datetime and Time Zone Parameters and Environment Variables](#)
- [Choosing a Time Zone File](#)
- [Upgrading the Time Zone File and Timestamp with Time Zone Data](#)
- [Clients and Servers Operating with Different Versions of Time Zone Files](#)
- [Setting the Database Time Zone](#)
- [Setting the Session Time Zone](#)
- [Converting Time Zones With the AT TIME ZONE Clause](#)
- [Support for Daylight Saving Time](#)

4.1 Overview of Datetime and Interval Data Types and Time Zone Support

Businesses conduct transactions across different time zones. Oracle Database datetime and interval data types and time zone support make it possible to store consistent information about the time of events and transactions.



Note: This chapter describes Oracle Database datetime and interval data types. It does not attempt to describe ANSI data types unless noted.

Was this page helpful?

4.2 Datetime and Interval Data Types

The **datetime data types** are DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE. Values of datetime data types are sometimes called **datetimes**.

The **interval data types** are INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND. Values of interval data types are sometimes called **intervals**.

Both datetimes and intervals are made up of fields. The values of these fields determine the value of the data type. The fields that apply to all Oracle Database datetime and interval data types are:

- YEAR
- MONTH
- DAY
- HOUR
- MINUTE
- SECOND

TIMESTAMP WITH TIME ZONE also includes these fields:

- TIMEZONE_HOUR
- TIMEZONE_MINUTE
- TIMEZONE_REGION
- TIMEZONE_ABBR

TIMESTAMP WITH LOCAL TIME ZONE does not store time zone information internally, but you can see local time zone information in SQL output if the TZH : TZM or TZR TZD format elements are specified.

The following sections describe the datetime data types and interval data types in more detail:

- [Datetime Data Types](#)
- [Interval Data Types](#)



See Also:

- [Oracle Database SQL Language Reference](#) for the valid values of the datetime and interval fields

Was this page helpful?

- [Oracle Database SQL Language Reference](#) for information about format elements

4.2.1 Datetime Data Types

This section includes the following topics:

- [DATE Data Type](#)
- [TIMESTAMP Data Type](#)
- [TIMESTAMP WITH TIME ZONE Data Type](#)
- [TIMESTAMP WITH LOCAL TIME ZONE Data Type](#)
- [Inserting Values into Datetime Data Types](#)
- [Choosing a TIMESTAMP Data Type](#)

4.2.1.1 DATE Data Type

The DATE data type stores date and time information. Although date and time information can be represented in both character and number data types, the DATE data type has special associated properties. For each DATE value, Oracle Database stores the following information: century, year, month, date, hour, minute, and second.

You can specify a date value by:

- Specifying the date value as a literal
- Converting a character or numeric value to a date value with the TO_DATE function

A date can be specified as an ANSI date literal or as an Oracle Database date value.

An ANSI date literal contains no time portion and must be specified in exactly the following format:

```
DATE 'YYYY-MM-DD'
```

 Copy

The following is an example of an ANSI date literal:

```
DATE '1998-12-25'
```

 Copy

Was this page helpful?

Alternatively, you can specify an Oracle Database date value as shown in the following example:

```
TO_DATE( '1998-DEC-25 17:30' , 'YYYY-MON-DD HH24:MI' , 'NLS_DATE_LANGUAGE=AMERICAN' )
```

 Copy

The default date format for an Oracle Database date value is derived from the `NLS_DATE_FORMAT` and `NLS_DATE_LANGUAGE` initialization parameters. The date format in the example includes a two-digit number for the day of the month, an abbreviation of the month name, the four digits of the year, and a 24-hour time designation. The specification for `NLS_DATE_LANGUAGE` is included because 'DEC' is not a valid value for MON in all locales.

Oracle Database automatically converts character values that are in the default date format into date values when they are used in date expressions.

If you specify a date value without a time component, then the default time is midnight. If you specify a date value without a date, then the default date is the first day of the current month.

Oracle Database DATE columns always contain fields for both date and time. If your queries use a date format without a time portion, then you must ensure that the time fields in the DATE column are set to midnight. You can use the `TRUNC (date)` SQL function to ensure that the time fields are set to midnight, or you can make the query a test of greater than or less than (`<`, `<=`, `>=`, or `>`) instead of equality or inequality (`=` or `!=`). Otherwise, Oracle Database may not return the query results you expect.



See Also:

- [Oracle Database SQL Language Reference](#) for more information about the DATE data type
- `"NLS_DATE_FORMAT"`
- `"NLS_DATE_LANGUAGE"`
- [Oracle Database SQL Language Reference](#) for more information about literals, format elements such as MM, and the TO_DATE function

4.2.1.2 TIMESTAMP Data Type

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

Specify the TIMESTAMP data type as follows:

```
TIMESTAMP [(fractional_seconds_precision)]
```

Was this page helpful?

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'
```

 Copy

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

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

 Copy

The value of NLS_TIMESTAMP_FORMAT initialization parameter determines the timestamp format when a character string is converted to the TIMESTAMP data type. NLS_DATE_LANGUAGE determines the language used for character data such as MON.



See Also:

- [Oracle Database SQL Language Reference](#) for more information about the TIMESTAMP data type
- "NLS_TIMESTAMP_FORMAT"
- "NLS_DATE_LANGUAGE"

4.2.1.3 TIMESTAMP WITH TIME ZONE Data Type

TIMESTAMP WITH TIME ZONE is a variant of TIMESTAMP that includes a time zone region name or time zone offset 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 data type as follows:

```
TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE
```

 Copy

fractional_seconds_precision is optional and specifies the number of digits in the fractional part of the SECOND datetime field.

Was this page helpful?

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

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

 Copy

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'
```

 Copy

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

```
TIMESTAMP '1999-01-15 8:00:00 America/Los_Angeles'
```

 Copy

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 U. S. Pacific Standard Time and `PDT` for U. S. Pacific Daylight Time. The following specification ensures that a Daylight Saving Time value is returned:

```
TIMESTAMP '1999-10-29 01:30:00 America/Los_Angeles PDT'
```

 Copy

If you do not add the TZD format element, and the datetime value is ambiguous, then Oracle Database 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 Database interprets the ambiguous datetime as Standard Time.

The default date format for the `TIMESTAMP WITH TIME ZONE` data type is determined by the value of the `NLS_TIMESTAMP_TZ_FORMAT` initialization parameter.



See Also:

- [Oracle Database SQL Language Reference](#) for more information about the `TIMESTAMP WITH TIME ZONE` data type
- ["TIMESTAMP Data Type"](#) for more information about fractional seconds precision
- ["Support for Daylight Saving Time"](#)

Was this page helpful?

- ["NLS_TIMESTAMP_TZ_FORMAT"](#)
- [Oracle Database SQL Language Reference](#) for more information about format elements
- [Oracle Database SQL Language Reference](#) for more information about setting the `ERROR_ON_OVERLAP_TIME` session parameter

4.2.1.4 TIMESTAMP WITH LOCAL TIME ZONE Data Type

`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 Database 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` data type as follows:

```
TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE
```

 Copy

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.



See Also:

- [Oracle Database SQL Language Reference](#) for more information about the `TIMESTAMP WITH LOCAL TIME ZONE` data type
- ["TIMESTAMP Data Type"](#) for more information about fractional seconds precision
- ["NLS_TIMESTAMP_FORMAT"](#)

4.2.1.5 Inserting Values into Datetime Data Types

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

Was this page helpful?

- 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 data types.



See Also:

"[Datetime SQL Functions](#)" for more information about the TO_TIMESTAMP or TO_TIMESTAMP_TZ SQL functions

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';
```

 Copy

Create a table table_dt with columns c_id and c_dt. The c_id column is of NUMBER data type and helps to identify the method by which the data is entered. The c_dt column is of DATE data type.

```
SQL> CREATE TABLE table_dt (c_id NUMBER, c_dt DATE);
```

 Copy

Insert a date as a character string.

```
SQL> INSERT INTO table_dt VALUES(1, '01-JAN-2003');
```

 Copy

Insert the same date as a DATE literal.

```
SQL> INSERT INTO table_dt VALUES(2, DATE '2003-01-01');
```

 Copy

Was this page helpful?

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

```
SQL> INSERT INTO table_dt VALUES(3, TIMESTAMP '2003-01-01 00:00:00 America/Los_Angeles');
```

 Copy


Insert the date with the `TO_DATE` function.

```
SQL> INSERT INTO table_dt VALUES(4, TO_DATE('01-JAN-2003', 'DD-MON-YYYY'));
```

 Copy

Display the data.

```
SQL> SELECT * FROM table_dt;
```


 Copy

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';
```

 Copy

Create a table `table_ts` with columns `c_id` and `c_ts`. The `c_id` column is of `NUMBER` data type and helps to identify the method by which the data is entered. The `c_ts` column is of `TIMESTAMP` data type.

```
SQL> CREATE TABLE table_ts(c_id NUMBER, c_ts TIMESTAMP);
```

 Copy

Insert a date and time as a character string.

Was this page helpful?

```
SQL> INSERT INTO table_ts VALUES(1, '01-JAN-2003 2:00:00');
```

[Copy](#)

Insert the same date and time as a `TIMESTAMP` literal.

```
SQL> INSERT INTO table_ts VALUES(2, TIMESTAMP '2003-01-01 2:00:00');
```

[Copy](#)

Insert the same date and time as a `TIMESTAMP WITH TIME ZONE` literal. Oracle Database 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');
```

[Copy](#)

Display the data.

```
SQL> SELECT * FROM table_ts;
```

[Copy](#)

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` Data Type

Set the timestamp format.

```
SQL> ALTER SESSION SET NLS_TIMESTAMP_TZ_FORMAT='DD-MON-RR HH:MI:SSXFF AM TZR';
```

[Copy](#)

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

Was this page helpful?

```
SQL> ALTER SESSION SET TIME_ZONE='-7:00';
```

 Copy

Create a table `table_tstz` with columns `c_id` and `c_tstz`. The `c_id` column is of `NUMBER` data type and helps to identify the method by which the data is entered. The `c_tstz` column is of `TIMESTAMP WITH TIME ZONE` data type.

```
SQL> CREATE TABLE table_tstz (c_id NUMBER, c_tstz TIMESTAMP WITH TIME ZONE);
```

 Copy

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');
```

 Copy

Insert the same date and time as a `TIMESTAMP` literal. Oracle Database 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');
```

 Copy

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');
```

 Copy

Display the data.

Was this page helpful?

```
SQL> SELECT * FROM table_tstz;
```

[Copy](#)

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` Data Type

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';
```

[Copy](#)

Create a table `table_tsltz` with columns `c_id` and `c_tsltz`. The `c_id` column is of `NUMBER` data type and helps to identify the method by which the data is entered. The `c_tsltz` column is of `TIMESTAMP WITH LOCAL TIME ZONE` data type.

```
SQL> CREATE TABLE table_tsltz (c_id NUMBER, c_tsltz TIMESTAMP WITH LOCAL TIME ZONE);
```

[Copy](#)

Insert a date and time as a character string.

```
SQL> INSERT INTO table_tsltz VALUES(1, '01-JAN-2003 2:00:00');
```

[Copy](#)

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');
```

[Copy](#)

Insert the same data as a `TIMESTAMP WITH TIME ZONE` literal. Oracle Database converts the data to a `TIMESTAMP WITH LOCAL TIME Z`

Was this page helpful?

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');
```

Copy

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

Copy

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.

4.2.1.6 Choosing a TIMESTAMP Data Type

Use the `TIMESTAMP` data type when you need a datetime value to record the time of an event without the time zone. For example, you can store information about the times when workers punch a time card in and out of their assembly line workstations. Because this is always a local time it is then not needed to store the timezone part. The `TIMESTAMP` data type uses 7 or 11 bytes of storage.

Use the `TIMESTAMP WITH TIME ZONE` data type when the datetime value represents a future local time or the time zone information must be recorded with the value. Consider a scheduled appointment in a local time. The future local time may need to be adjusted if the time zone definition, such as daylight saving rule, changes. Otherwise, the value can become incorrect. This data type is most immune to such impact.

The `TIMESTAMP WITH TIME ZONE` data type requires 13 bytes of storage, or two more bytes of storage than the `TIMESTAMP` and `TIMESTAMP WITH LOCAL TIME ZONE` data types because it stores time zone information. The time zone is stored as a time zone region name or as an offset from UTC. 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` data type 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` data type is appropriate when the original time zone is of no interest, but the relative times of events are important and daylight saving adjustment does not have to be accurate. The time zone conversion that this data type performs to and from the database time

daylight saving adjustment. Because this data type does not preserve the time zone information, it does not distinguish values near the adjustment in fall, whether they are daylight saving time or standard time. This confusion between distinct instants can cause an application to behave unexpectedly, especially if the adjustment takes place during the normal working hours of a user.

Note that some regions, such as Brazil and Israel, that update their Daylight Saving Transition dates frequently and at irregular periods, are particularly susceptible to time zone adjustment issues. If time information from these regions is key to your application, you may want to consider using one of the other datetime types.

4.2.2 Interval Data Types

Interval data types store time durations. They are used primarily with analytic functions. For example, you can use them to calculate a moving average of stock prices. You must use interval data types to determine the values that correspond to a particular percentile. You can also use interval data types to update historical tables.

This section includes the following topics:

- [INTERVAL YEAR TO MONTH Data Type](#)
- [INTERVAL DAY TO SECOND Data Type](#)
- [Inserting Values into Interval Data Types](#)



See Also:

[Oracle Database Data Warehousing Guide](#) for more information about analytic functions, including moving averages and inverse percentiles

4.2.2.1 INTERVAL YEAR TO MONTH Data Type

INTERVAL YEAR TO MONTH stores a period of time using the YEAR and MONTH datetime fields. Specify INTERVAL YEAR TO MONTH as follows:

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

 Copy

year_precision is the number of digits in the YEAR datetime field. Accepted values are 0 to 9. The default value of *year_precision* is 2.

Interval values can be specified as literals. There are many ways to specify interval literals. The following is one example of specifying an interval of 123 years and 2 months. The year precision is 3.

```
INTERVAL '123-2' YEAR(3) TO MONTH
```

 Copy



See Also:

[Oracle Database SQL Language Reference](#) for more information about specifying interval literals with the INTERVAL YEAR TO MONTH data type

4.2.2.2 INTERVAL DAY TO SECOND Data Type

Was this page helpful?

INTERVAL DAY TO SECOND stores a period of time in terms of days, hours, minutes, and seconds. Specify this data type as follows:

```
INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds_precision)]
```

 Copy

day_precision is the number of digits in the DAY datetime field. Accepted values are 0 to 9. The default is 2.

fractional_seconds_precision is the number of digits in the fractional part of the SECOND datetime field. Accepted values are 0 to 9. The default is 6.

The following is one example of specifying an interval of 4 days, 5 hours, 12 minutes, 10 seconds, and 222 thousandths of a second. The fractional second precision is 3.

```
INTERVAL '4 5:12:10.222' DAY TO SECOND(3)
```

 Copy

Interval values can be specified as literals. There are many ways to specify interval literals.



See Also:

[Oracle Database SQL Language Reference](#) for more information about specifying interval literals with the INTERVAL DAY TO SECOND data type

4.2.2.3 Inserting Values into Interval Data Types

You can insert values into an interval column in the following ways:

- Insert an interval as a literal. For example:

```
INSERT INTO table1 VALUES (INTERVAL '4-2' YEAR TO MONTH);
```

 Copy

This statement inserts an interval of 4 years and 2 months.

Oracle Database recognizes literals for other ANSI interval types and converts the values to Oracle Database interval values.

- Use the NUMTODSINTERVAL, NUMTOYMINTERVAL, TO_DSINTERVAL, and TO_YMINTERVAL SQL functions.



Was this page helpful?

See Also:

["Datetime SQL Functions"](#)

4.3 Datetime and Interval Arithmetic and Comparisons

This section includes the following topics:

- [Datetime and Interval Arithmetic](#)
- [Datetime Comparisons](#)
- [Explicit Conversion of Datetime Data Types](#)

4.3.1 Datetime and Interval Arithmetic

You can perform arithmetic operations on date (DATE), timestamp (TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE) and interval (INTERVAL DAY TO SECOND and INTERVAL YEAR TO MONTH) data. You can maintain the most precision in arithmetic operations by using a timestamp data type with an interval data type.

You can use NUMBER constants in arithmetic operations on date and timestamp values. Oracle Database internally converts timestamp values to date values before doing arithmetic operations on them with NUMBER constants. This means that information about fractional seconds is lost during operations that include both date and timestamp values. Oracle Database interprets NUMBER constants in datetime and interval expressions as number of days.

Each DATE value contains a time component. The result of many date operations includes a fraction. This fraction means a portion of one day. For example, 1.5 days is 36 hours. These fractions are also returned by Oracle Database built-in SQL functions for common operations on DATE data. For example, the built-in MONTHS_BETWEEN SQL function returns the number of months between two dates. The fractional portion of the result represents that portion of a 31-day month.

Oracle Database performs all timestamp arithmetic in UTC time. For TIMESTAMP WITH LOCAL TIME ZONE data, Oracle Database converts the datetime value from the database time zone to UTC and converts back to the database time zone after performing the arithmetic. For TIMESTAMP WITH TIME ZONE data, the datetime value is always in UTC, so no conversion is necessary.



See Also:

- [Oracle Database SQL Language Reference](#) for detailed information about datetime and interval arithmetic operations
- ["Datetime SQL Functions"](#) for information about which functions cause implicit conversion to DATE

Was this page helpful?

4.3.2 Datetime Comparisons

When you compare date and timestamp values, Oracle Database converts the data to the more precise data type before doing the comparison. For example, if you compare data of `TIMESTAMP WITH TIME ZONE` data type with data of `TIMESTAMP` data type, Oracle Database converts the `TIMESTAMP` data to `TIMESTAMP WITH TIME ZONE`, using the session time zone.

The order of precedence for converting date and timestamp data is as follows:

- 1. `DATE`
- 2. `TIMESTAMP`
- 3. `TIMESTAMP WITH LOCAL TIME ZONE`
- 4. `TIMESTAMP WITH TIME ZONE`

For any pair of data types, Oracle Database converts the data type that has a smaller number in the preceding list to the data type with the larger number.

4.3.3 Explicit Conversion of Datetime Data Types

If you want to do explicit conversion of datetime data types, use the `CAST` SQL function. You can explicitly convert `DATE`, `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` to another data type in the list.



See Also:

[Oracle Database SQL Language Reference](#)

4.4 Datetime SQL Functions

Datetime functions operate on date (`DATE`), timestamp (`TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE`) and interval (`INTERVAL DAY TO SECOND`, `INTERVAL YEAR TO MONTH`) values.

Some of the datetime functions were designed for the Oracle Database `DATE` data type. If you provide a timestamp value as their argument, then Oracle Database internally converts the input type to a `DATE` value. Oracle Database does not perform internal conversion for the `ROUND` and `TRUNC` functions.

The following table shows the datetime functions that were designed for the Oracle Database `DATE` data type.

Table 4-1 Datetime Functions Designed for the `DATE` Data Type

Was this page helpful?

Function	Description
<code>ADD_MONTHS</code>	Returns the date <i>d</i> plus <i>n</i> months
<code>LAST_DAY</code>	Returns the last day of the month that contains <i>date</i>
<code>MONTHS_BETWEEN</code>	Returns the number of months between <i>date1</i> and <i>date2</i>
<code>NEW_TIME</code>	Returns the date and time in <i>zone2</i> time zone when the date and time in <i>zone1</i> time zone are <i>date</i> Note: This function takes as input only a limited number of time zones. You can have access to a much greater number of time zones by combining the <code>FROM_TZ</code> function and the datetime expression.
<code>NEXT_DAY</code>	Returns the date of the first weekday named by <i>char</i> that is later than <i>date</i>
<code>ROUND(date)</code>	Returns <i>date</i> rounded to the unit specified by the <i>fmt</i> format model
<code>TRUNC(date)</code>	Returns <i>date</i> with the time portion of the day truncated to the unit specified by the <i>fmt</i> format model

The following table describes additional datetime functions.

Table 4-2 Additional Datetime Functions

Datetime Function	Description
<code>CURRENT_DATE</code>	Returns the current date in the session time zone in a value in the Gregorian calendar, of the DATE data type
<code>CURRENT_TIMESTAMP</code>	Returns the current date and time in the session time zone as a <code>TIMESTAMP WITH TIME ZONE</code> value
<code>DBTIMEZONE</code>	Returns the value of the database time zone. The value is a time zone offset or a time zone region name
<code>EXTRACT(datetime)</code>	Extracts and returns the value of a specified datetime field from a datetime or interval value expression
<code>FROM_TZ</code>	Converts a <code>TIMESTAMP</code> value at a time zone to a <code>TIMESTAMP WITH TIME ZONE</code> value
<code>LOCALTIMESTAMP</code>	Returns the current date and time in the session time zone in a value of the <code>TIMESTAMP</code> data type
<code>NUMTODSINTERVAL</code>	Converts number <i>n</i> to an <code>INTERVAL DAY TO SECOND</code> literal
<code>NUMTOYMINTERVAL</code>	Converts number <i>n</i> to an <code>INTERVAL YEAR TO MONTH</code> literal

Was this page helpful?

Datetime Function	Description
SESSIONTIMEZONE	Returns the value of the current session's time zone
SYS_EXTRACT_UTC	Extracts the UTC from a datetime with time zone offset
SYSDATE	Returns the date and time of the operating system on which the database resides, taking into account the time zone of the database server's operating system that was in effect when the database was started
SYSTIMESTAMP	Returns the system date, including fractional seconds and time zone of the system on which the database resides
TO_CHAR (datetime)	Converts a datetime or interval value of DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, or TIMESTAMP WITH LOCAL TIME ZONE data type to a value of VARCHAR2 data type in the format specified by the <i>fmt</i> date format
TO_DSINTERVAL	Converts a character string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of INTERVAL DAY TO SECOND data type
TO_NCHAR (datetime)	Converts a datetime or interval value of DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE, INTERVAL MONTH TO YEAR, or INTERVAL DAY TO SECOND data type from the database character set to the national character set
TO_TIMESTAMP	Converts a character string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of TIMESTAMP data type
TO_TIMESTAMP_TZ	Converts a character string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of the TIMESTAMP WITH TIME ZONE data type
TO_YMINTERVAL	Converts a character string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of the INTERVAL YEAR TO MONTH data type
TZ_OFFSET	Returns the time zone offset that corresponds to the entered value, based on the date that the statement is executed

The following table describes the functions related to the Daylight Saving Time (DST) upgrade process.

Table 4-3 Time Zone Conversion Functions

Time Zone Function	Description
ORA_DST_AFFECTED	Enables you to verify whether the data in a column is affected by upgrading the DST rules from one version to another version
ORA_DST_CONVERT	Enables you to upgrade your TSTZ column data from one version to another
ORA_DST_ERROR	Enables you to verify that there are no errors when upgrading a datetime value



Was this page helpful?

See Also:

- [Oracle Database SQL Language Reference](#) for more information about the Oracle Database datetime functions
- ["Support for Daylight Saving Time"](#) for more information about the Daylight Saving Time functionality of Oracle Database
- ["Daylight Saving Time Session Parameter"](#) for information about the session parameter `ERROR_ON_OVERLAP_TIME` related to Daylight Saving Time
- ["Daylight Saving Time Upgrade Parameter"](#) for information about the initialization parameter `DST_UPGRADE_INSERT_CONV` that is used during the Daylight Saving Time upgrade process

4.5 Datetime and Time Zone Parameters and Environment Variables

This section includes the following topics:

- [Datetime Format Parameters](#)
- [Time Zone Environment Variables](#)
- [Daylight Saving Time Session Parameter](#)
- [Daylight Saving Time Upgrade Parameter](#)

4.5.1 Datetime Format Parameters

The following table contains the names and descriptions of the datetime format parameters.

Table 4-4 Datetime Format Parameters

Parameter	Description
NLS_DATE_FORMAT	Defines the default date format to use with the TO_CHAR and TO_DATE functions
NLS_TIMESTAMP_FORMAT	Defines the default timestamp format to use with the TO_CHAR and TO_TIMESTAMP functions
NLS_TIMESTAMP_TZ_FORMAT	Defines the default timestamp with time zone format to use with the TO_CHAR and TO_TIMESTAMP_TZ functions

Their default values are derived from NLS_TERRITORY.

You can specify their values by setting them in the initialization parameter file. If you change the values in the initialization parameter file, you must restart the instance for the change to take effect. You can also specify their values for a client as client environment variables. For Java clients, the value of NLS_TERRITORY is used as the locale of JRE. The values specified in the initialization parameter file are not used for JDBC sessions.

Was this page helpful?

To change their values during a session, use the ALTER SESSION statement.



See Also:

- ["Date and Time Parameters"](#) for more information, including examples
- ["NLS_DATE_FORMAT"](#)
- ["NLS_TIMESTAMP_FORMAT"](#)
- ["NLS_TIMESTAMP_TZ_FORMAT"](#)

4.5.2 Time Zone Environment Variables

The time zone environment variables are:

- ORA_TZFILE, which enables you to specify a time zone on the client and Oracle Database server. Note that when you specify ORA_TZFILE on Oracle Database server, the only time when this environment variable takes effect is during the creation of the database.
- ORA_SDTZ, which specifies the default session time zone.



See Also:

- ["Choosing a Time Zone File"](#)
- ["Setting the Session Time Zone"](#)

4.5.3 Daylight Saving Time Session Parameter

ERROR_ON_OVERLAP_TIME is a session parameter that determines how Oracle Database handles an ambiguous datetime boundary value. Ambiguous datetime values can occur when the time changes between Daylight Saving Time and standard time.

The possible values are TRUE and FALSE. When ERROR_ON_OVERLAP_TIME is TRUE, then an error is returned when Oracle Database encounters an ambiguous datetime value. When ERROR_ON_OVERLAP_TIME is FALSE, then ambiguous datetime values are assumed to be the standard time representation for the region. The default value is FALSE.



Was this page helpful?

See Also:

- ["Support for Daylight Saving Time"](#)
- [Oracle Database SQL Language Reference](#)

4.5.4 Daylight Saving Time Upgrade Parameter

DST_UPGRADE_INSERT_CONV is an initialization parameter that is only used during the upgrade window of the Daylight Saving Time (DST) upgrade process. It is only applicable to tables with `TIMESTAMP WITH TIME ZONE` columns because those are the only ones that are modified during the DST upgrade.

During the upgrade window of the DST patching process (which is described in the `DBMS_DST` package), tables with `TIMESTAMP WITH TIMEZONE` data undergo conversion to the new time zone version. Columns in tables that have not yet been converted will still have the `TIMESTAMP WITH TIMEZONE` reflecting the previous time zone version. In order to present the data in these columns as though they had been converted to the new time zone version when you issue `SELECT` statements, Oracle adds by default conversion operators over the columns to convert them to the new version. Adding the conversion operator may, however, slow down queries and disable usage of indexes on the `TIMESTAMP WITH TIMEZONE` columns. Hence, Oracle provides a parameter, `DST_UPGRADE_INSERT_CONV`, that specifies whether or not internal operators are allocated on top of `TIMESTAMP WITH TIMEZONE` columns of tables that have not been upgraded. By default, its value is `TRUE`. If users know that the conversion process will not affect the `TIMESTAMP WITH TIMEZONE` columns, then this parameter can be set to `FALSE`.

Oracle strongly recommends that you set this parameter to `TRUE` throughout the DST patching process. By default, this parameter is set to `TRUE`. However, if set to `TRUE`, query performance may be degraded on unconverted tables. Note that this only applies during the upgrade window.



See Also:

- [Oracle Database Reference](#)
- [Oracle Database PL/SQL Packages and Types Reference](#)

4.6 Choosing a Time Zone File

The Oracle Database time zone files contain the valid time zone names. The following information is also included for each time zone:

- Offset from Coordinated Universal Time (UTC)
- Transition times for Daylight Saving Time
- Abbreviations for standard time and Daylight Saving Time

Was this page helpful?

Oracle Database supplies multiple versions of time zone files, and there are two types of file associated with each version: a large file, which contains all the time zones defined in the database, and a small file, which contains only the most commonly used time zones. The large version files are named as `timezlg_version_number.dat` and the small version files are named as `timezone_version_number.dat`, where `version_number` is the version number of the time zone file. The time zone files are stored in the `$ORACLE_HOME/oracore/zoneinfo` directory and the default time zone file is a large time zone file having the highest version number. In Oracle Database 21c, the default time zone file is `$ORACLE_HOME/oracore/zoneinfo/timezlg_35.dat`.

Examples of time zone files are:

```
$ORACLE_HOME/oracore/zoneinfo/timezlg_34.dat -- large version 34
$ORACLE_HOME/oracore/zoneinfo/timezone_34.dat -- small version 34
$ORACLE_HOME/oracore/zoneinfo/timezlg_35.dat -- large version 35
$ORACLE_HOME/oracore/zoneinfo/timezone_35.dat -- small version 35
```

 Copy

During the database creation process, you choose the time zone version for the server. This version is fixed, but you can, however, go through the upgrade process to achieve a higher version. You can use different versions of time zone files on the client and server, but Oracle recommends that you do not. This is because there is a performance penalty when a client on one version communicates with a server on a different version. The performance penalty arises because the `TIMESTAMP WITH TIME ZONE (TSTZ)` data is transferred using a local timestamp instead of UTC.

On the server, Oracle Database always uses a large file. On the client, you can use either a large or a small file. If you use a large time zone file on the client, then you should continue to use it unless you are sure that no information will be missing if you switch to a smaller one. If you use a small file, you have to make sure that the client does not query data that is not present in the small time zone file. Otherwise, you get an error.

You can enable the use of a specific time zone file on the client or on the server. If you want to enable a time zone file on the server, there are two situations. One is when you want to upgrade the time zone to the target version. See "[Upgrading the Time Zone File and Timestamp with Time Zone Data](#)" for more information. Another is when you want to create a new database. In this case, you can set the `ORA_TZFILE` environment variable to point to the time zone file of your choice.

To enable a specific time zone file on a client, you can set `ORA_TZFILE` to whatever time zone file you want. If `ORA_TZFILE` is not set, Oracle Database automatically picks up and uses the file with the latest time zone version.



See Also:

[Oracle Call Interface Programmer's Guide](#) for more information on how to upgrade Daylight Saving Time on a client



Note: Oracle Database time zone data is derived from the public domain information available on [The IANA Functions](#) website. Oracle Database time zone data may not reflect the most recent data available on this website.

Was this page helpful?

You can obtain a list of time zone names and time zone abbreviations from the time zone file that is installed with your database by entering the following statement:

```
SELECT TZNAME, TZABBREV
FROM V$TIMEZONE_NAMES
ORDER BY TZNAME, TZABBREV;
```

Copy

For the default time zone file, this statement results in output similar to the following:

```
TZNAME          TZABBREV
-----
Africa/Abidjan  GMT
Africa/Abidjan  LMT
...
Africa/Algiers  CEST
Africa/Algiers  CET
Africa/Algiers  LMT
Africa/Algiers  PMT
Africa/Algiers  WET
Africa/Algiers  WEST
...
WET             LMT
WET             WEST
WET             WET

2137 rows selected.
```

Copy

In the above output, 2 time zone abbreviations are associated with the Africa/Abidjan time zone, and 6 abbreviations are associated with the Africa/Algiers time zone. The following table shows some of the time zone abbreviations and their meanings.

Time Zone Abbreviation	Meaning
LMT	Local Mean Time
PMT	Paris Mean Time
WET	Western European Time
WEST	Western European Summer Time

Was this page helpful?

Time Zone Abbreviation	Meaning
CET	Central Europe Time
CEST	Central Europe Summer Time
EET	Eastern Europe Time
EEST	Eastern Europe Summer Time

Note that an abbreviation can be associated with multiple time zones. For example, CET is associated with both Africa/Algiers and Africa/Casablanca, as well as time zones in Europe.

If you want a list of time zones without repeating the time zone name for each abbreviation, use the following query:

```
SELECT UNIQUE TZNAME
FROM V$TIMEZONE_NAMES;
```

Copy

For example, version 11 contains output similar to the following:

```
TZNAME
-----
Africa/Addis_Ababa
Africa/Bissau
Africa/Ceuta
...
Turkey
US/East-Indiana
US/Samoa
```

Copy

The default time zone file, that is, the large time zone file contains more than 350 unique time zone names. The small time zone file contains more than 180 unique time zone names.



See Also:

- ["Time Zone Region Names"](#) for a list of valid Oracle Database time zone names
- `$ORACLE_HOME/oracore/zoneinfo/timzdif.csv` provided with your Oracle Database software installation for a full list of

Was this page helpful?

version of the time zone file.

4.7 Upgrading the Time Zone File and Timestamp with Time Zone Data

The time zone files that are supplied with the Oracle Database are updated periodically to reflect changes in transition rules for various time zone regions. To find which time zone file your database currently uses, query the `V$TIMEZONE_FILE` view.



Note: Each Oracle Database release includes a time zone file that is current at the time of the release and a number of older version files. Between Oracle Database releases, new time zone file versions may be provided in patch sets or individual patches to reflect the changes in transition rules for various time zone regions. Older time zone file versions allow you to run upgraded databases without a need to immediately upgrade the time zone file to the most current version.

Daylight Saving Time (DST) Transition Rules Changes

Governments can and do change the rules for when Daylight Saving Time takes effect or how it is handled. When this occurs, Oracle provides a new set of transition rules for handling timestamp with time zone data.

Transition periods for the beginning or ending of Daylight Saving Time can potentially introduce problems (such as data loss) when handling timestamps with time zone data. Oracle has provided the PL/SQL package `DBMS_DST` and the `utltz_*` scripts to deal with this transition.

The changes to DST transition rules may affect existing data of `TIMESTAMP WITH TIME ZONE` data type, because of the way Oracle Database stores this data internally. When users enter timestamps with time zone, Oracle Database converts the data to UTC, based on the transition rules in the time zone file, and stores the data together with the ID of the original time zone on disk. When data is retrieved, the reverse conversion from UTC takes place. For example, in the past, under version 2 transition rules, the value `TIMESTAMP '2007-11-02 12:00:00 America/Los_Angeles'` was stored as UTC value `'2007-11-02 20:00:00'` *plus* the time zone ID for `'America/Los_Angeles'`. The time in Los Angeles was stored as UTC *minus* eight hours (PST). Under version 3 of the transition rules, the offset for the same day is *minus* seven hours (PDT). Beginning with year 2007, the DST has been in effect longer (until the first Sunday of November, which was November 4th in 2007). Now, when users retrieve the same timestamp and the new offset is added to the stored UTC time, they receive `TIMESTAMP '2007-11-02 13:00:00 America/Los_Angeles'`. There is a one hour difference compared to the data previous to version 3 taking effect.



Note: For any time zone region whose transition rules have been updated, the upgrade process discussed throughout this section affects only timestamps that point to the future relative to the effective date of the corresponding DST rule change. For example, no timestamp before year 2007 is affected by the version 3 change to the `'America/Los_Angeles'` time zone region.

Preparing to Upgrade the Time Zone File and Timestamp with Time Zone Data

Before you actually upgrade any data, that is `TIMESTAMP WITH TIME ZONE` (TSTZ) data in a database, you should verify what the im

Was this page helpful?

general, you can consider the upgrade process to have two separate sub-processes – prepare and upgrade. To prepare for the upgrade, you start a prepare window, which is the time when you check how much data has to be updated in the database. To upgrade, you start an upgrade window, which is the time when changes to the data actually occur.

While not required, Oracle strongly recommends that you perform the prepare step. In addition to finding out how much data will have to be modified during the upgrade, thus giving you an estimate of how much time the upgrade will take, you will also see any semantic errors that you may encounter.

Upgrading the Time Zone File and Timestamp with Time Zone Data in a Multitenant Environment

The following guidelines apply when upgrading the time zone file and timestamp with time zone data in a multitenant environment:

- Each container in a multitenant environment has its own time zone file. Therefore, to perform a time zone data upgrade across an entire CDB, you must upgrade the CDB root and each PDB separately. Note that Oracle allows different containers to have different time zone file versions, so you have the option of upgrading only a subset of containers in a CDB.
- When performing a time zone data upgrade in a CDB (using either the DBMS_DST package or the `utlztz_*` scripts), you must perform the Prepare Window steps and the Upgrade Window steps completely in one container before moving on to the next container.
- A new PDB is always assigned the time zone version of PDB\$SEED.
- PDB\$SEED is always assigned the time zone version at the time of CDB creation. The time zone version of PDB\$SEED cannot be changed.

Methods for Upgrading the Time Zone File and Timestamp with Time Zone Data

You can upgrade the time zone data in your database using either of the following methods:

- [Upgrading the Time Zone Data Using the DBMS_DST Package](#)

This method provides the most control over the individual steps of the time zone data upgrade. Starting with Oracle Database 21c, you have the option of performing this method while the database is online or offline. The online version of this method requires one restart of the database at your convenience and requires minimal locks on tables that need to be upgraded. It allows applications to query all time zone data and to insert and modify time zone data for all tables that can be migrated online. The offline version of this method, which was also available in previous releases, requires the database to be in UPGRADE mode during part of the procedure and is more restrictive about when applications can insert and modify time zone data.

- [Upgrading the Time Zone Data Using the `utlztz_*` Scripts](#)

This method is easier to perform than the method that uses the DBMS_DST package, because it provides a set of wrapper scripts that call the various DBMS_DST procedures. However, during the time zone upgrade process, the database is automatically restarted multiple times and applications cannot query or insert time zone data in the database.

4.7.1 Upgrading the Time Zone Data Using the DBMS_DST Package

This section describes how to perform a time zone data upgrade using the DBMS_DST package.

Was this page helpful?

Each container in a multitenant environment has its own time zone file. Therefore, to perform a time zone data upgrade across an entire CDB, you must upgrade the CDB root and each PDB separately. Note that Oracle allows different containers to have different time zone file versions, so you have the option of upgrading only a subset of containers in a CDB.

Perform the steps in the following sections completely in one container before moving on to the next container:

- [Prepare Window](#)
- [Upgrade Window](#)
- [Upgrade Example](#)
- [Upgrade Error Handling](#)

4.7.1.1 Prepare Window

During the prepare window, you can get the information about the data that will be affected during the time zone upgrade process using the following steps:

1. Install the desired version of time zone files to which you will be later migrating into `$ORACLE_HOME/oracore/zoneinfo`. If the desired version is *version_number*, then you must add the file `timezlg_version_number.dat`. You can add the file `timezone_version_number.dat` at your discretion later. These files can be found on My Oracle Support. The desired version should be the latest version available, unless the latest version contains relevant DST rule changes that were rolled back by the appropriate government after the version had been released.
2. You can optionally create the following tables:
 - an error table that contains the errors generated during the upgrade process by using the `DBMS_DST.CREATE_ERROR_TABLE` procedure. If you do not explicitly create this table, then the default table used is `sys.dst$error_table`.
 - a table that contains the affected timestamp with time zone information by using the `DBMS_DST.CREATE_AFFECTED_TABLE` procedure. If you do not explicitly create this table, then the default table used is `sys.dst$affected_tables`.
 - a trigger table that stores the disabled TSTZ table triggers information by using the `DBMS_DST.CREATE_TRIGGER_TABLE` procedure. If you do not explicitly create this table, then the default table used is `sys.dst$trigger_table`. Note that during the upgrade window, Oracle Database first disables the triggers on a TSTZ table and then performs the upgrade of its affected TSTZ data. Oracle Database saves the information about those triggers in the `sys.dst$trigger_table` table. After completing the upgrade of the affected TSTZ data in the table, the disabled triggers are enabled by reading their information from the `sys.dst$trigger_table` table and then their information is removed from the `sys.dst$trigger_table` table. If any fatal error occurs, such as an unexpected instance shutdown during the upgrade window, you should check the `sys.dst$trigger_table` table to see if any trigger has not been restored to its previous active state before the upgrade.
3. Purge unneeded data.

The `DBMS_SCHEDULER` log tables contain a large amount of time zone data. If this data is not needed, then delete it using the following steps. Stop the main jobs before running this command as it may not delete all of the data from the `DBMS_SCHEDULER`

Was this page helpful?

jobs in a chain of jobs are still running.

```
exec dbms_scheduler.purge_log;
```

 Copy

The other tables that may contain a large amount of time zone data are the `SYS.WRI$_OPTSTAT_HISTGRM_HISTORY` and `SYS.WRI$_OPTSTAT_HISTHEAD_HISTORY` tables. In case you do not need this data, then you may delete it using the following commands:

```
-- check the number of rows in the tables
select count(*) from SYS.WRI$_OPTSTAT_HISTGRM_HISTORY;
select count(*) from SYS.WRI$_OPTSTAT_HISTHEAD_HISTORY;

-- check the data retention period of the stats
-- the default value is 31
select systimestamp - dbms_stats.get_stats_history_availability from dual;

-- disable stats retention
exec dbms_stats.alter_stats_history_retention(0);

-- remove all the stats
exec DBMS_STATS.PURGE_STATS(systimestamp);

-- check the result of the purge operation
select count(*) from SYS.WRI$_OPTSTAT_HISTGRM_HISTORY;
select count(*) from SYS.WRI$_OPTSTAT_HISTHEAD_HISTORY;
```

 Copy

You may set the data retention period back to its original value using the following command once the time zone data upgrade is completed:

```
exec dbms_stats.alter_stats_history_retention(31);
```

 Copy

4. Execute the procedure `DBMS_DST.BEGIN_PREPARE(new_version)`, where *new_version* is the time zone file version you chose in Step 1.
5. Collect information about affected data by executing the procedure `DBMS_DST.FIND_AFFECTED_TABLES`, optionally passing the names of custom tables created in Step 2 as parameters. Verify the affected columns by querying `sys.dst$aaffected_tables` or the corresponding custom table. Also, it is particularly important to check `sys.dst$aaffected_tables.error_count` or the corresponding `error_count` column in the custom table. If the `error_count` column is greater than 0, you can check what kind of errors you might expect during the upgrade by checking `sys.dst$error_t`

Was this page helpful?

custom error table. See "[Upgrade Error Handling](#)".

6. End the prepare window by executing the procedure `DBMS_DST . END_PREPARE`.

Note:



- Only one DBA should run the prepare window at one time. Also, make sure to correct all errors before running the upgrade.
- You can find the matrix of available patches for updating your time zone files by going to Oracle Support and reading Document ID 412160.1.



See Also:

[Oracle Database PL/SQL Packages and Types Reference](#) for more information about `DBMS_DST` package

4.7.1.2 Upgrade Window



Note: In a multitenant environment, all the PDBs must be shut down before running the `DBMS_DST . UPGRADE_DATABASE` procedure on a CDB.

During the upgrade window, you can upgrade the time zone data using the following steps:

1. If you have not already done so, download the desired version of `timezlg_version_number . dat` and install it in `$ORACLE_HOME/oracore/zoneinfo`. In addition, you can optionally download `timezone_version_number . dat` from My Oracle Support and put it in the same location.
2. Starting with Oracle Database 21c, you have the option of performing this method while the database is online or offline. The online version of this method requires one restart of the database at your convenience and requires minimal locks on tables that need to be upgraded. It allows applications to query all time zone data and to insert and modify time zone data for all tables that can be migrated online. The offline version of this method, which was also available in previous releases, requires the database to be in `UPGRADE` mode during part of the procedure and is more restrictive about when applications can insert and modify time zone data.
 - a. If you choose to perform the online version of this procedure:
 - i. Set the `TIMEZONE_VERSION_UPGRADE_ONLINE` initialization parameter to `true`:

```
ALTER SYSTEM SET TIMEZONE_VERSION_UPGRADE_ONLINE = true;
```

Copy

- ii. Proceed to Step 3.

- b. If you choose to perform the offline version of this procedure:

Was this page helpful?

- i. Ensure that the `TIMEZONE_VERSION_UPGRADE_ONLINE` initialization parameter is set to its default value of `false`:

```
ALTER SYSTEM SET TIMEZONE_VERSION_UPGRADE_ONLINE = false;
```



- ii. Shut down the database. In Oracle RAC, you must shut down all instances.

- iii. Start up the database in the UPGRADE mode. Note that, in Oracle RAC, only one instance should be started. See [Oracle Database Upgrade Guide](#) for more information about the UPGRADE mode.

3. Run the procedure `DBMS_DST.BEGIN_UPGRADE(new_version)`. Optionally, you can set the `error_on_overlap_time` and `error_on_nonexisting_time` parameters to `TRUE` if you do not want to ignore semantic errors during the upgrade of dictionary tables that contain timestamp with time zone data. Examples of such errors could be related to `DBMS_SCHEDULER`, as discussed in "[Upgrade Error Handling](#)". If you specify `TRUE` for either or both of these parameters, the errors are populated into `sys.dst$error_table`. In this case, you might want to truncate the error table before you run the `BEGIN_UPGRADE` procedure. See *Oracle Database PL/SQL Packages and Types Reference* for more information.

The `BEGIN_UPGRADE` procedure modifies time zone information in the database, but does not modify any user table data. While the `BEGIN_UPGRADE` procedure is operating, you cannot add tables containing TSTZ columns to the database, nor can you add TSTZ columns to existing tables. TSTZ columns are columns defined on `TIMESTAMP WITH TIME ZONE` data types or object types containing attributes of `TIMESTAMP WITH TIME ZONE` data types.

After the `BEGIN_UPGRADE` procedure has successfully executed, user tables that contain TSTZ data will be marked with the `UPGRADE IN PROGRESS` property.

4. If the `BEGIN_UPGRADE` procedure fails, the error "ORA-56927: Starting an upgrade window failed" is displayed.

After the `BEGIN_UPGRADE` procedure finishes executing with errors, check `sys.dst$error_table` to determine if the dictionary conversion was successful. If successful, there will not be any rows in the table. If there are errors, correct those errors manually and rerun the `BEGIN_UPGRADE` procedure. See "[Upgrade Error Handling](#)".

5. Truncate the error and trigger tables (by default, `sys.dst$error_table` and `sys.dst$trigger_table`).

6. This step depends on whether you are performing the online or offline version of this procedure:

- If you are performing the online version of this procedure, then allow the database to continue running until you reach a convenient time to perform a reboot. During this time, the database is still operating with the old time zone version and the TSTZ data has not yet been updated. You are allowed to add tables that contain TSTZ columns to the database and you can add TSTZ columns to existing tables. When it is a convenient time to reboot the database, shut down the database and then restart it in normal mode. In Oracle RAC, you must shut down all instances first before restarting them.
- If you are performing the offline version of this procedure (that is, you put the database in UPGRADE mode in Step 2), then restart the database now in normal mode.

7. Confirm that the new time zone version is the primary version by running the following query:

Was this page helpful?


```
SELECT * FROM V$TIMEZONE_FILE;
```

 Copy

8. Allow the database to continue running until you reach a convenient time to upgrade the TSTZ data in all tables. A convenient time could be when the database has low usage.

While the database continues running, previously existing TSTZ columns still have the `TIMESTAMP WITH TIMEZONE` reflecting the previous time zone version. In order to present the data in these columns as though they have been converted to the new time zone version when you issue `SELECT` statements, Oracle adds conversion operators over the columns to convert them to the new version. Newly created TSTZ columns will have `TIMESTAMP WITH TIMEZONE` reflecting the new time zone version.

9. When it is a convenient time to upgrade the TSTZ data in all tables, run the procedure `DBMS_DST.Upgrade_Database`.

Each TSTZ table is upgraded in a separate transaction. For TSTZ base tables with materialized view logs, the base table and materialized view log table are upgraded in one transaction. During the upgrade, if `TIMEZONE_VERSION_UPGRADE_ONLINE` is set to `true`, the upgrade operation will be done online whenever possible. For tables containing TSTZ data that cannot be upgraded online, the database will acquire an exclusive DML lock. If `TIMEZONE_VERSION_UPGRADE_ONLINE` is set to `false`, all upgrade operations will acquire an exclusive DML lock on each table and do the upgrade. All dependent cursors on a table will be invalidated when the upgrade of the table is complete.

10. Verify that all tables have been upgraded by querying the `DBA_TSTZ_TABLES` view, as shown in "[Upgrade Example](#)". Then check `sys.dst$error_table` to see if there are any errors. If there are errors, correct the errors and rerun the `DBMS_DST.Upgrade_Table` procedure for the relevant tables. Or, if you do not think those errors are important, then rerun the `DBMS_DST.Upgrade_Table` procedure with the parameters set to ignore errors.
11. End the upgrade window by running the procedure `DBMS_DST.End_Upgrade`.

Was this page helpful?

Notes:

- Tables containing `TIMESTAMP WITH TIME ZONE` columns need to be in a state where they can be updated. Therefore, as an example, the columns cannot have validated and disabled check constraints because this prevents updating.
- Oracle recommends that you use the parallel option if a table size is greater than 2 Gigabytes. Oracle also recommends that you allow Oracle to handle any semantic errors that may arise.
- When you run the `CREATE` statements for error, trigger, or affected tables, you must pass the table name only, not the schema name. This is because the tables are created in the schema from which the `CREATE` statements are run.
- You cannot create materialized views or materialized view logs on tables marked with the `UPGRADE_IN_PROGRESS` property; if you attempt to do so, error `ORA-30403` or `ORA-32426` will be raised. You cannot alter existing materialized view logs on tables marked with the `UPGRADE_IN_PROGRESS` property; if you attempt to do so, error `ORA-32426` will be raised. Existing materialized views on tables that are marked with the `UPGRADE_IN_PROGRESS` property are not allowed to perform a refresh; if a refresh is attempted, error `ORA-30403` will be raised.



See Also:

[Oracle Database PL/SQL Packages and Types Reference](#) for more information about `DBMS_DST` package

4.7.1.3 Upgrade Example

This example illustrates updating DST behavior by using the online method of the procedure. It upgrades an Oracle database to time zone version 14. First, assume that your current database is using time zone version 3, and also assume you have an existing table `t`, which contains timestamp with time zone data.

Connect to the database as the user `scot t` and execute the following statements:

```

DROP TABLE t;
CREATE TABLE t (c NUMBER, mark VARCHAR(25), ts TIMESTAMP WITH TIME ZONE);

INSERT INTO t VALUES(1, 'not_affected',
                      to_timestamp_tz('22-sep-2006 13:00:00 america/los_angeles',
                      'dd-mon-yyyy hh24:mi:ss tzh tzh'));
INSERT INTO t VALUES(4, 'affected_err_exist',
                      to_timestamp_tz('11-mar-2007 00:30:00 america/st_johns',
                      'dd-mon-yyyy hh24:mi:ss tzh tzh'));
INSERT INTO t VALUES(6, 'affected_no_err',
                      to_timestamp_tz('11-mar-2007 01:30:00 america/st_johns',
                      'dd-mon-yyyy hh24:mi:ss tzh tzh'));
INSERT INTO t VALUES(14, 'affected_err_dup',
                      to_timestamp_tz('21-sep-2006 23:30:00 egypt',
                      'dd-mon-yyyy hh24:mi:ss tzh tzh'));

COMMIT;

```

 Copy

Then, optionally, you can start a prepare window to check the affected data and potential semantic errors where there is an overlap or non-existing time. To do this, you should start a window for preparation to migrate to time zone version 14. It is assumed that you have the necessary privileges. These privileges are controlled with the DBMS_DST package.



See Also:

[Oracle Database PL/SQL Packages and Types Reference](#) for more information about the DBMS_DST package

As an example, first, prepare the window.

```

connect / as sysdba
set serveroutput on
EXEC DBMS_DST.BEGIN_PREPARE(14);

```

 Copy

A prepare window has been successfully started.

PL/SQL procedure successfully completed.

Was this page helpful?

Note that the argument 14 causes the time zone version 14 to be used in this statement. After this window is successfully started, you can check the status of the DST in DATABASE_PROPERTIES as shown in the following example:

```
SELECT property_name, SUBSTR(property_value, 1, 30) value
FROM database_properties
WHERE property_name LIKE 'DST_%'
ORDER BY property_name;
```

 Copy

You will see the output similar to the following:

PROPERTY_NAME	VALUE
-----	-----
DST_PRIMARY_TT_VERSION	3
DST_SECONDARY_TT_VERSION	14
DST_UPGRADE_STATE	PREPARE

 Copy

Next, you can execute `DBMS_DST.FIND_AFFECTED_TABLES` to find all the tables in the database that are affected if you upgrade from version 3 to version 14. This table contains the table owner, table name, column name, row count, and error count. Here, you have the choice of using the defaults for error tables (`sys.dst$error_table`) and affected tables (`sys.dst$affected_tables`) or you can create your own. In this example, we create our own tables by using `DBMS_DST.CREATE_ERROR_TABLE` and `DBMS_DST.CREATE_AFFECTED_TABLE` and then pass to `FIND_AFFECTED_TABLES` as shown below.

Connect to the database as the user SYS and execute the following statements:

```
EXEC DBMS_DST.CREATE_AFFECTED_TABLE('my_affected_tables');
EXEC DBMS_DST.CREATE_ERROR_TABLE('my_error_table');
```

 Copy

Then, you can execute `FIND_AFFECTED_TABLES` to see which tables are impacted during the upgrade:

Was this page helpful?

```
connect / as sysdba
```

 Copy

```
BEGIN
  DBMS_DST.FIND_AFFECTED_TABLES(affected_tables => 'my_affected_tables',
                                log_errors      => TRUE,
                                log_errors_table => 'my_error_table');
END;
/
```

Then, check the affected tables:

```
SELECT * FROM my_affected_tables;
```

 Copy

TABLE_OWNER	TABLE_NAME	COLUMN_NAME	ROW_COUNT	ERROR_COUNT
SCOTT	T	TS	3	2

Then, check the error table:

```
SELECT * FROM my_error_table;
```

 Copy

TABLE_OWNER	TABLE_NAME	COLUMN_NAME	ROWID	ERROR_NUMBER
SCOTT	T	TS	AAAPW3AABAAANZoAAB	1878
SCOTT	T	TS	AAAPW3AABAAANZoAAE	1883

These errors can be corrected as described in "[Upgrade Error Handling](#)". Then, end the prepare window, as in the following statement:

```
EXEC DBMS_DST.END_PREPARE;
```

 Copy

A prepare window has been successfully ended.

PL/SQL procedure successfully completed.

Was this page helpful?

After this, you can check the DST status in DATABASE_PROPERTIES:

```
SELECT property_name, SUBSTR(property_value, 1, 30) value
FROM database_properties
WHERE property_name LIKE 'DST_%'
ORDER BY property_name;
```

 Copy

PROPERTY_NAME	VALUE
-----	-----
DST_PRIMARY_TT_VERSION	3
DST_SECONDARY_TT_VERSION	0
DST_UPGRADE_STATE	NONE

Next, you can use the upgrade window to upgrade the affected data. To do this, first set the TIMEZONE_VERSION_UPGRADE_ONLINE initialization parameter to true:

```
ALTER SYSTEM SET TIMEZONE_VERSION_UPGRADE_ONLINE = true;
```

 Copy

Execute DBMS_DST.BEGIN_UPGRADE. In Oracle RAC, all instances must be running. BEGIN_UPGRADE upgrades all dictionary tables in one transaction, so the invocation will either succeed or fail as one whole. During the procedure's execution, all tables with TSTZ data are marked as an upgrade in progress. You cannot add tables containing TSTZ columns to the database, nor can you add TSTZ columns to existing tables. See [Oracle Database Upgrade Guide](#) for more information.

So, BEGIN_UPGRADE upgrades system tables that contain TSTZ data and marks user tables (containing TSTZ data) with the UPGRADE_IN_PROGRESS property. This can be checked in DBA_TSTZ_TABLES, and is illustrated later in this example.

There are two parameters in BEGIN_UPGRADE that are for handling semantic errors: error_on_overlap_time (error number ORA-1883) and error_on_nonexisting_time (error number ORA-1878). If the parameters use the default setting of FALSE, Oracle converts the data using a default conversion and does not signal an error. See "[Upgrade Error Handling](#)" for more information regarding what they mean, and how to handle errors.

The following call can automatically correct semantic errors based on some default values when you upgrade the dictionary tables. If you do not ignore semantic errors, and you do have such errors in the dictionary tables, BEGIN_UPGRADE will fail. These semantic errors are populated into my_error_table.

Was this page helpful?

```
EXEC DBMS_DST.BEGIN_UPGRADE(14);  
An upgrade window has been successfully started.  
  
PL/SQL procedure successfully completed.
```

 Copy

After this, you can check the DST status in DATABASE_PROPERTIES, as in the following:

```
SELECT property_name, SUBSTR(property_value, 1, 30) value  
FROM database_properties  
WHERE property_name LIKE 'DST_%'  
ORDER BY property_name;
```

 Copy

PROPERTY_NAME	VALUE
DST_PRIMARY_TT_VERSION	14
DST_SECONDARY_TT_VERSION	3
DST_UPGRADE_STATE	NEEDRESTART

Then, note that all tables containing TSTZ data display YES for UPGRADE_IN_PROGRESS:

Was this page helpful?

```
SELECT owner, table_name, upgrade_in_progress FROM dba_tstz_tables;
```

 Copy

OWNER	TABLE_NAME	UPGRADE_IN_PROGRESS
----	-----	-----
SYS	WRI\$_OPTSTAT_AUX_HISTORY	YES
SYS	WRI\$_OPTSTAT_OPR	YES
SYS	OPTSTAT_HIST_CONTROL\$	YES
SYS	SCHEDULER\$_JOB	YES
SYS	KET\$_AUTOTASK_STATUS	YES
SYS	AQ\$_ALERT_QT_S	YES
SYS	AQ\$_KUPC\$DATAPUMP_QUETAB_S	YES
DBSNMP	MGMT_DB_FEATURE_LOG	YES
WMSYS	WM\$VERSIONED_TABLES	YES
SYS	WRI\$_OPTSTAT_IND_HISTORY	YES
SYS	OPTSTAT_USER_PREFS\$	YES
SYS	FGR\$_FILE_GROUP_FILES	YES
SYS	SCHEDULER\$_WINDOW	YES
SYS	WRR\$_REPLAY_DIVERGENCE	YES
SCOTT	T	YES
IX	AQ\$_ORDERS_QUEUE_TABLE_S	YES
...		

Note that the upgrade is in progress for table SCOTT . T. If you access SCOTT . T, the database will transparently apply conversion operators to ensure to properly reflect the new time zone rules on this table while the upgrade is in progress.

After BEGIN_UPGRADE has successfully executed, the database continues to operate with the old time zone file. When it is a convenient time to reboot the database, shut down the database and then restart it in normal mode.

Confirm that the new time zone version is the primary version by running the following query:

```
SELECT * FROM V$TIMEZONE_FILE;
```

 Copy

FILENAME	VERSION	CON_ID
-----	-----	-----
timezlg_14.dat	14	0

Now you can perform an upgrade of all tables containing TSTZ data by using DBMS_DST . UPGRADE_DATABASE. All tables must be upg

Was this page helpful?

3

end the upgrade window using the END_UPGRADE procedure.

Consider the following choices for running the DBMS_DST.UPGRADE_DATABASE procedure:

- Oracle recommends that you use the following code to perform the upgrade. It instructs the database to resolve any errors encountered and relieves you of having to resolve the errors manually.

```
VAR numfail number;
BEGIN
    DBMS_DST.UPGRADE_DATABASE(:numfail);
    DBMS_OUTPUT.PUT_LINE('Number of tables failed to upgrade:' || :numfail);
END;
/
```

 Copy

- However, if you prefer to view and resolve errors manually, you can use the following code:

```
VAR numfail number;
BEGIN
    DBMS_DST.UPGRADE_DATABASE(:numfail,
        parallel           => TRUE,
        log_errors         => TRUE,
        log_errors_table   => 'my_error_table',
        log_triggers_table => 'SYS.DST$TRIGGER_TABLE',
        error_on_overlap_time => TRUE,
        error_on_nonexisting_time => TRUE);

    DBMS_OUTPUT.PUT_LINE('Number of tables failed to upgrade:' || :numfail);
END;
/
```

 Copy

If there are any errors, you should correct them and use UPGRADE_TABLE on the individual tables. In that case, you may need to handle tables related to materialized views, such as materialized view base tables, materialized view log tables, and materialized view container tables. There are a couple of considerations to keep in mind when upgrading these tables. First, the base table and its materialized view log table have to be upgraded atomically. Next, the materialized view container table has to be upgraded after all its base tables and the materialized view log tables have been upgraded. In general, Oracle recommends that you handle semantic errors by letting Oracle Database take the default action.

For the sake of this example, let us assume there were some errors in SCOTT.T after you ran UPGRADE_DATABASE. In that case, you can check these errors by using the following query:

Was this page helpful?

```
SELECT * FROM my_error_table;
```

 Copy

TABLE_OWNER	TABLE_NAME	COLUMN_NAME	ROWID	ERROR_NUMBER
-----	-----	-----	-----	-----
SCOTT	T	TS	AAA02XAABAAANrgAAD	1878
SCOTT	T	TS	AAA02XAABAAANrgAAE	1878

In the output, you can see the errors having number 1878. This error means that an error has occurred for a non-existing time.

To continue with this example, assume that SCOTT.T has a materialized view log `log scott.mlog$_t`, and that there is a single materialized view on SCOTT.T. Then, assume that this 1878 error has been corrected.

Finally, you can upgrade the table, materialized view log and materialized view as follows:

Was this page helpful?



```
VAR numfail number;
BEGIN
  DBMS_DST.UPGRADE_TABLE(:numfail,
    table_list          => 'SCOTT.t, SCOTT.mlog$_T',
    parallel             => TRUE,
    continue_after_errors => FALSE,
    log_errors           => TRUE,
    log_errors_table     => 'my_error_table',
    error_on_overlap_time => FALSE,
    error_on_nonexisting_time => TRUE,
    log_triggers_table   => 'SYS.DST$TRIGGER_TABLE',
    atomic_upgrade       => TRUE);

  DBMS_OUTPUT.PUT_LINE('Number of tables failed to upgrade:' || :numfail);
END;
/

VAR numfail number;
BEGIN
  DBMS_DST.UPGRADE_TABLE(:numfail,
    table_list          => 'SCOTT.MYMV_T',
    parallel             => TRUE,
    continue_after_errors => FALSE,
    log_errors           => TRUE,
    log_errors_table     => 'my_error_table',
    error_on_overlap_time => FALSE,
    error_on_nonexisting_time => TRUE,
    log_triggers_table   => 'SYS.DST$TRIGGER_TABLE',
    atomic_upgrade       => TRUE);

  DBMS_OUTPUT.PUT_LINE('Number of tables failed to upgrade:' || :numfail);
END;
/
```

The `atomic_upgrade` parameter enables you to combine the upgrade of the table with its materialized view log.

After all the tables are upgraded, you can invoke `END_UPGRADE` to end an upgrade window as shown below:

Was this page helpful?

```

VAR numfail number;
BEGIN
    DBMS_DST.END_UPGRADE(:numfail);
    DBMS_OUTPUT.PUT_LINE('Number of tables failed to upgrade:' || :numfail);
END;
/

```



The upgrade window ends if all the affected tables are upgraded successfully, else the output shows how many tables did not upgrade successfully.

4.7.1.4 Upgrade Error Handling

There are three major semantic errors that can occur during an upgrade. The first is when an existing time becomes a non-existing time, the second is when a time becomes duplicated, and the third is when a duplicate time becomes a non-duplicate time.

As an example of the first case, consider the change from Pacific Standard Time (PST) to Pacific Daylight Saving Time (PDT) in 2007. This change takes place on Mar-11-2007 at 2AM according to version 3 (and any later version up to at least 32) when the clock moves forward one hour to 3AM and produces a gap between 2AM and 3AM. In version 2, this time change occurs on Apr-01-2007. If you upgrade the time zone file from version 2 to version 3, any time in the interval between 2AM and 3AM on Mar-11-2007 is not valid, and raises an error (ORA-1878) if `ERROR_ON_NONEXISTING_TIME` is set to `TRUE`. Therefore, there is a non-existing time problem. When `ERROR_ON_NONEXISTING_TIME` is set to `FALSE`, which is the default value for this parameter, the error is not reported and Oracle preserves UTC time in this case. For example, "Mar-11-2007 02:30 PST" in version 2 becomes "Mar-11-2007 03:30 PDT" in version 3 as they both are translated to the same UTC timestamp.

An example of the second case occurs when changing from PDT to PST. For example, in version 3 for 2007, the change occurs on Nov-04-2007, when the time falls back from 2AM to 1AM. This means that times in the interval <1AM, 2AM> on Nov-04-2007 can appear twice, once with PST and once with PDT. In version 2, this transition occurs on Oct-28-2007 at 2AM. Thus, any timestamp within <1AM, 2AM> on Nov-04-2007, which is uniquely identified in version 2, results in an error (ORA-1883) in version 3, if `ERROR_ON_OVERLAP_TIME` is set to `TRUE`. If you leave this parameter on its default setting of `FALSE`, then the UTC timestamp value is preserved and no error is raised. In this situation, if you change the version from 2 to 3, timestamp "Nov-04-2007 01:30 PST" in version 2 becomes "Nov-04-2007 01:30 PST" in version 3.

The third case happens when a duplicate time becomes a non-duplicate time. Consider the transition from PDT to PST in 2007, for example, where <1AM, 2AM> on Oct-28-2007 in version 2 is the overlapped interval. Then both "Oct-28-2007 01:30 PDT" and "Oct-28-2007 01:30 PST" are valid timestamps in version 2. If `ERROR_ON_OVERLAP_TIME` is set to `TRUE`, an ORA-1883 error is raised during an upgrade from version 2 to version 3. If `ERROR_ON_OVERLAP_TIME` is set to `FALSE` (the default value of this parameter), however, the LOCAL time is preserved and no error is reported. In this case, both "Oct-28-2007 01:30 PDT" and "Oct-28-2007 01:30 PST" in version 2 convert to the same "Oct-28-2007 01:30 PDT" in version 3. Note that setting `ERROR_ON_OVERLAP_TIME` to `FALSE` can potentially cause some time sequences to be reversed. For example, a job (Job A) scheduled at "Oct-28-2007 01:45 PDT" in version 2 is supposed to be executed before a job (Job B) scheduled at "Oct-28-2007 01:30 PST". After the upgrade to version 3, Job A is scheduled at "Oct-28-2007 01:45 PDT" and Job B remains at "Oct-28-2007 01:30 PDT", resulting in Job B being executed before Job A. Even though unchained scheduled jobs are not guaranteed to be executed in a certain order, this issue should be kept in mind when setting up scheduled jobs.



Was this page helpful?

See Also:

[Oracle Database PL/SQL Packages and Types Reference](#) for more information regarding how to use these parameters

4.7.2 Upgrading the Time Zone Data Using the utltz_* Scripts

This section describes how to perform a time zone data upgrade using the `utltz_*` scripts.

Each container in a multitenant environment has its own time zone file. Therefore, to perform a time zone data upgrade across an entire CDB, you must upgrade the CDB root and each PDB separately. Note that Oracle allows different containers to have different time zone file versions, so you have the option of upgrading only a subset of containers in a CDB.

Perform the steps in the following sections completely in one container before moving on to the next container:

- [Prepare Window](#)
- [Upgrade Window](#)

4.7.2.1 Prepare Window

During the prepare window, you can get the information about the data that will be affected during the time zone upgrade process using the following steps:

1. Install the desired version of time zone files to which you will be later migrating into `$ORACLE_HOME/oracore/zoneinfo`. If the desired version is `version_number`, then you must add the file `timezlg_version_number.dat`. You can add the file `timezone_version_number.dat` at your discretion later. These files can be found on My Oracle Support. The desired version should be the latest version available, unless the latest version contains relevant DST rule changes that were rolled back by the appropriate government after the version had been released.
2. Run any of the following scripts present in the `$ORACLE_HOME/rdbms/admin` directory to check how much data will need to be updated in the database during the time zone data upgrade:

- `utltz_countstats.sql`

This script shows the optimizer statistics of `num_rows` of all the tables having `TIMESTAMP WITH TIME ZONE (TSTZ)` data.

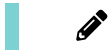


Note: Run the `utltz_countstats.sql` script only when the database optimizer statistics are up to date. Otherwise, run the `utltz_countstar.sql` script. If you run the `utltz_countstats.sql` script, then you need not run the `utltz_countstar.sql` script.

- `utltz_countstar.sql`

This script shows the result of the `count (*)` operation for all the tables having `TIMESTAMP WITH TIME ZONE (TSTZ)` data.

Was this page helpful?



3. Purge unneeded data.

Perform this step as described in the Prepare Window for upgrading the time zone data using the DBMS_DST package. Refer to Step 3 in "[Prepare Window](#)".

4.7.2.2 Upgrade Window

During the upgrade window, you can run the following scripts present in the \$ORACLE_HOME/rdbms/admin directory to upgrade the time zone data in the database:

1. If you have not already done so, download the desired version of `timezlg_version_number.dat` and install it in \$ORACLE_HOME/oracore/zoneinfo. In addition, you can optionally download `timezone_version_number.dat` from My Oracle Support and put it in the same location.
2. Ensure that the `TIMEZONE_VERSION_UPGRADE_ONLINE` initialization parameter is set to its default value of `false`:

```
ALTER SYSTEM SET TIMEZONE_VERSION_UPGRADE_ONLINE = false;
```

Copy

3. Run the `utltz_upg_check.sql` script from the \$ORACLE_HOME directory:

```
spool utltz_upg_check.log
@utltz_upg_check.sql
spool off
```

Copy

The following information is displayed on the screen after successful execution of the script:

```
INFO: A newer RDBMS DST version than the one currently used is found.
INFO: Note that NO DST update was yet done.
INFO: Now run utltz_upg_apply.sql to do the actual RDBMS DST update.
INFO: Note that the utltz_upg_apply.sql script will
INFO: restart the database 2 times WITHOUT any confirmation or prompt.
```

Copy

The script also writes the following information in the `alert.log` file:

`utltz_upg_check` successfully found newer RDBMS DSTv *new_time_zone_version* and took *number_of_minutes* minutes to run.

If the `utltz_upg_check.sql` script displays the following error, check the previous message displayed on the screen and proceed

Was this page helpful?

ORA-20xxx: Stopping script - see previous message...

4. Run the `utltz_upg_apply.sql` script from the `$ORACLE_HOME` directory after the `utltz_upg_check.sql` script is executed successfully:

```
spool utltz_upg_apply.log
@utltz_upg_apply.sql
spool off
```

 Copy

Note: The following are the prerequisites for running the `utltz_upg_apply.sql` script:

- In an Oracle RAC environment, the Oracle RAC database must be started as a single database instance.
- In a multitenant environment, all the PDBs must be shut down before running the `utltz_upg_apply.sql` script on the CDB.



Also, note the following:

- The `utltz_upg_apply.sql` script automatically restarts the database multiple times during its execution.
- The `utltz_upg_apply.sql` script generally takes less time to execute than the `utltz_upg_check.sql` script.

The following information is displayed on the screen after successful execution of the `utltz_upg_apply.sql` script:

```
INFO: The RDBMS DST update is successfully finished.
INFO: Make sure to exit this sqlplus session.
INFO: Do not use it for timezone related selects.
```

 Copy

The `TZ_VERSION` column in the `Registry$database` table now gets updated with the new time zone version.


If the script displays the following error message, then check the previous message displayed on the screen and proceed accordingly.

ORA-20xxx: Stopping script - see previous message...

Note: If you want to see what is happening when the scripts `utltz_upg_check.sql` and `utltz_upg_apply.sql` are being executed, run the following commands:

Was this page helpful?

```
set PAGES 1000
```

 Copy

```
-- query the V$SESSION_LONGOPS view
select TARGET, TO_CHAR(START_TIME, 'HH24:MI:SS - DD-MM-YY'),
       TIME_REMAINING, SOFAR, TOTALWORK, SID, SERIAL#, OPNAME
from V$SESSION_LONGOPS
where sid in
       (select SID from V$SESSION where CLIENT_INFO = 'upg_tzv')
       and
       SOFAR < TOTALWORK
order by START_TIME;

-- query the V$SESSION and V$SQLAREA views
select S.SID, S.SERIAL#, S.SQL_ID, S.PREV_SQL_ID,
       S.EVENT#, S.EVENT, S.P1TEXT, S.P1, S.P2TEXT,
       S.P2, S.P3TEXT, S.P3, S.TIME_REMAINING_MICRO,
       S.SEQ#, S.BLOCKING_SESSION, BS.PROGRAM "Blocking Program",
       Q1.SQL_TEXT "Current SQL", Q2.SQL_TEXT "Previous SQL"
from V$SESSION S, V$SQLAREA Q1, V$SQLAREA Q2, V$SESSION BS
where S.SQL_ID = Q1.SQL_ID(+) and
       S.PREV_SQL_ID = Q2.SQL_ID(+) and
       S.BLOCKING_SESSION = BS.SID(+) and
       S.CLIENT_INFO = 'upg_tzv';
```

4.8 Clients and Servers Operating with Different Versions of Time Zone Files

In Oracle Database 11g, Release 11.2 and later, you can use different versions of time zone file on the client and the server. This mode of operation was not supported in the earlier Oracle Database releases. Both client and server must be Oracle Database 11g, Release 11.2 or later to operate in such a mixed mode.



See Also:

[Oracle Call Interface Programmer's Guide](#) for the ramifications of working in the mixed mode

OCI, JDBC, Pro*C, and SQL*Plus clients can now continue to communicate with the database server without having to update client-side time zone files. For other products, if not explicitly stated in the product-specific documentation, it should be assumed that such clients cannot operate with a database server with a different time zone file than the client. Otherwise, computations on the `TIMESTAMP WITH TIMEZONE` values that are region ID based may give inconsistent results (such as incorrect DST rules in effect for the time zone regions affected between the different time zone file versions at the client and server).

Was this page helpful?

Note if an application connects to different databases directly or via database links, it is recommended that all databases be on the same time zone file version. Otherwise, computations on the `TIMESTAMP WITH TIMEZONE` values on these different databases may give inconsistent results. This is due to different DST rules in effect for the time zone regions affected between the different time zone file versions across the different database servers.

4.9 Setting the Database Time Zone

Set the database time zone when the database is created by using the `SET TIME_ZONE` clause of the `CREATE DATABASE` statement. If you do not set the database time zone, then it defaults to the time zone of the server's operating system.

The time zone may be set to a named region or an absolute offset from UTC. To set the time zone to a named region, use a statement similar to the following example:

```
CREATE DATABASE db01
...
SET TIME_ZONE='Europe/London';
```

 Copy

To set the time zone to an offset from UTC, use a statement similar to the following example:

```
CREATE DATABASE db01
...
SET TIME_ZONE='-05:00';
```

 Copy

The range of valid offsets is -12:00 to +14:00.



Note: The database time zone is relevant only for `TIMESTAMP WITH LOCAL TIME ZONE` columns. Oracle recommends that you set the database time zone to UTC (0:00) to avoid data conversion and improve performance when data is transferred among databases. This is especially important for distributed databases, replication, and exporting and importing.

You can change the database time zone by using the `SET TIME_ZONE` clause of the `ALTER DATABASE` statement. For example:

```
ALTER DATABASE SET TIME_ZONE='Europe/London';
ALTER DATABASE SET TIME_ZONE='-05:00';
```

 Copy

The `ALTER DATABASE SET TIME_ZONE` statement returns an error if the database contains a table with a `TIMESTAMP WITH LOCAL TIME ZONE` column and the column contains data.

The change does not take effect until the database has been shut down and restarted.

Was this page helpful?


You can find out the database time zone by entering the following query:

```
SELECT dbtimezone FROM DUAL;
```

 Copy

4.10 Setting the Session Time Zone

You can set the default session time zone with the `ORA_SDTZ` environment variable. When users retrieve `TIMESTAMP WITH LOCAL TIME ZONE` data, Oracle Database returns it in the users' session time zone. The session time zone also takes effect when a `TIMESTAMP` value is converted to the `TIMESTAMP WITH TIME ZONE` or `TIMESTAMP WITH LOCAL TIME ZONE` data type.

 **Note:** Setting the session time zone does not affect the value returned by the `SYSDATE` and `SYSTIMESTAMP` SQL function. `SYSDATE` returns the date and time of the operating system on which the database resides, taking into account the time zone of the database server's operating system that was in effect when the database was started.

The `ORA_SDTZ` environment variable can be set to the following values:

- Operating system local time zone (`'OS_TZ'`)
- Database time zone (`'DB_TZ'`)
- Absolute offset from UTC (for example, `'-05:00'`)
- Time zone region name (for example, `'Europe/London'`)

To set `ORA_SDTZ`, use statements similar to one of the following in a UNIX environment (C shell):

```
% setenv ORA_SDTZ 'OS_TZ'  
% setenv ORA_SDTZ 'DB_TZ'  
% setenv ORA_SDTZ 'Europe/London'  
% setenv ORA_SDTZ '-05:00'
```

 Copy

When setting the `ORA_SDTZ` variable in a Microsoft Windows environment -- in the Registry, among system environment variables, or in a command prompt window -- do not enclose the time zone value in quotes.

The default value of the `ORA_SDTZ` variable, which is used when the variable is not set or it is set to an invalid value, is `'OS_TZ'`.

You can change the time zone for a specific SQL session with the `SET TIME_ZONE` clause of the `ALTER SESSION` statement.

Was this page helpful?

TIME_ZONE can be set to the following values:

- Default local time zone when the session was started (local)
- Database time zone (dbtimezone)
- Absolute offset from UTC (for example, '+10:00')
- Time zone region name (for example, 'Asia/Hong_Kong')

Use ALTER SESSION statements similar to the following:

```
ALTER SESSION SET TIME_ZONE=local;  
ALTER SESSION SET TIME_ZONE=dbtimezone;  
ALTER SESSION SET TIME_ZONE='Asia/Hong_Kong';  
ALTER SESSION SET TIME_ZONE='+10:00';
```

 Copy

You can find out the current session time zone by entering the following query:

```
SELECT sessiontimezone FROM DUAL;
```

 Copy

4.11 Converting Time Zones With the AT TIME ZONE Clause

A datetime SQL expression can be one of the following:

- A datetime column
- A compound expression that yields a datetime value

A datetime expression can include an AT LOCAL clause or an AT TIME ZONE clause. If you include an AT LOCAL clause, then the result is returned in the current session time zone. If you include the AT TIME ZONE clause, then use one of the following settings with the clause:

- Time zone offset: The string '(+|-)HH:MM' specifies a time zone as an offset from UTC. For example, '-07:00' specifies the time zone that is 7 hours behind UTC. For example, if the UTC time is 11:00 a.m., then the time in the '-07:00' time zone is 4:00 a.m.
- DBTIMEZONE: Oracle Database uses the database time zone established (explicitly or by default) during database creation.
- SESSIONTIMEZONE: Oracle Database uses the session time zone established by default or in the most recent ALTER SESSION statement.
- Time zone region name: Oracle Database returns the value in the time zone indicated by the time zone region name. For example,

Was this page helpful?

Asia/Hong_Kong.

- An expression: If an expression returns a character string with a valid time zone format, then Oracle Database returns the input in that time zone. Otherwise, Oracle Database returns an error.

The following example converts the datetime value in the America/New_York time zone to the datetime value in the America/Los_Angeles time zone.



See Also:

[Previous Page](#) [Next Page](#) © Oracle [About Oracle](#) [Contact Us](#) [Products A-Z](#) [Terms of Use & Privacy](#) [Cookie Preferences](#) [Ad Choices](#)

Example 4-5 Converting a Datetime Value to Another Time Zone

```
SELECT FROM_TZ(CAST(TO_DATE('1999-12-01 11:00:00',
    'YYYY-MM-DD HH:MI:SS') AS TIMESTAMP), 'America/New_York')
    AT TIME ZONE 'America/Los_Angeles' "West Coast Time"
FROM DUAL;
```

Copy

West Coast Time

01-DEC-99 08.00.00.000000 AM AMERICA/LOS_ANGELES

4.12 Support for Daylight Saving Time

Oracle Database automatically determines whether Daylight Saving Time is in effect for a specified time zone and returns the corresponding local time. Normally, date/time values are sufficient to allow Oracle Database to determine whether Daylight Saving Time is in effect for a specified time zone. The periods when Daylight Saving Time begins or ends are boundary cases. For example, in the Eastern region of the United States, the time changes from 01:59:59 a.m. to 3:00:00 a.m. when Daylight Saving Time goes into effect. The interval between 02:00:00 and 02:59:59 a.m. does not exist. Values in that interval are invalid. When Daylight Saving Time ends, the time changes from 02:00:00 a.m. to 01:00:01 a.m. The interval between 01:00:01 and 02:00:00 a.m. is repeated. Values from that interval are ambiguous because they occur twice.

To resolve these boundary cases, Oracle Database uses the TZR and TZD format elements. TZR represents the time zone region in datetime input strings. Examples are 'Australia/North', 'UTC', and 'Singapore'. TZD represents an abbreviated form of the time zone region with Daylight Saving Time information. Examples are 'PST' for U. S. Pacific Standard Time and 'PDT' for U. S. Pacific Daylight Time. To see a list of valid values for the TZR and TZD format elements, query the TZNAME and TZABBREV columns of the V\$TIMEZONE_NAMES dynamic performance view.



Was this page helpful?

See Also:

- [Oracle Database SQL Language Reference](#) for more information regarding the session parameter `ERROR_ON_OVERLAP_TIME`
- ["Time Zone Region Names"](#) for a list of valid time zones

4.12.1 Examples: The Effect of Daylight Saving Time on Datetime Calculations

The `TIMESTAMP` data type does not accept time zone values and does not calculate Daylight Saving Time.


The `TIMESTAMP WITH TIME ZONE` and `TIMESTAMP WITH LOCAL TIME ZONE` data types have the following behavior:

- If a time zone region is associated with the datetime value, then the database server knows the Daylight Saving Time rules for the region and uses the rules in calculations.
- Daylight Saving Time is not calculated for regions that do not use Daylight Saving Time.

The rest of this section contains examples that use datetime data types. The examples use the `global_orders` table. It contains the `orderdate1` column of `TIMESTAMP` data type and the `orderdate2` column of `TIMESTAMP WITH TIME ZONE` data type. The `global_orders` table is created as follows:

```
CREATE TABLE global_orders ( orderdate1 TIMESTAMP(0),
                             orderdate2 TIMESTAMP(0) WITH TIME ZONE);
INSERT INTO global_orders VALUES ( '28-OCT-00 11:24:54 PM',
                                   '28-OCT-00 11:24:54 PM America/New_York');
```

 Copy

 **Note:** If you have created a `global_orders` table for the previous examples, then drop the `global_orders` table before you try [Example 4-7](#) through [Example 4-8](#).

Example 4-6 Comparing Daylight Saving Time Calculations Using `TIMESTAMP WITH TIME ZONE` and `TIMESTAMP`

```
SELECT orderdate1 + INTERVAL '8' HOUR, orderdate2 + INTERVAL '8' HOUR
FROM global_orders;
```

 Copy

The following output results:

Was this page helpful?

ORDERDATE1+INTERVAL '8' HOUR	ORDERDATE2+INTERVAL '8' HOUR
-----	-----
29-OCT-00 07.24.54.000000 AM	29-OCT-00 06.24.54.000000 AM AMERICA/NEW_YORK

 Copy

This example shows the effect of adding 8 hours to the columns. The time period includes a Daylight Saving Time boundary (a change from Daylight Saving Time to standard time). The orderdate1 column is of `TIMESTAMP` data type, which does not use Daylight Saving Time information and thus does not adjust for the change that took place in the 8-hour interval. The `TIMESTAMP WITH TIME ZONE` data type does adjust for the change, so the orderdate2 column shows the time as one hour earlier than the time shown in the orderdate1 column.

Example 4-7 Comparing Daylight Saving Time Calculations Using `TIMESTAMP WITH LOCAL TIME ZONE` and `TIMESTAMP`

The `TIMESTAMP WITH LOCAL TIME ZONE` data type uses the value of `TIME_ZONE` that is set for the session environment. The following statements set the value of the `TIME_ZONE` session parameter and create a `global_orders` table. The `global_orders` table has one column of `TIMESTAMP` data type and one column of `TIMESTAMP WITH LOCAL TIME ZONE` data type.

```
ALTER SESSION SET TIME_ZONE='America/New_York';
CREATE TABLE global_orders ( orderdate1 TIMESTAMP(0),
                             orderdate2 TIMESTAMP(0) WITH LOCAL TIME ZONE );
INSERT INTO global_orders VALUES ( '28-OCT-00 11:24:54 PM',
                                   '28-OCT-00 11:24:54 PM' );
```

 Copy

Add 8 hours to both columns.

```
SELECT orderdate1 + INTERVAL '8' HOUR, orderdate2 + INTERVAL '8' HOUR
FROM global_orders;
```

 Copy

Because a time zone region is associated with the datetime value for orderdate2, the Oracle Database server uses the Daylight Saving Time rules for the region. Thus the output is the same as in [Example 4-6](#). There is a one-hour difference between the two calculations because Daylight Saving Time is not calculated for the `TIMESTAMP` data type, and the calculation crosses a Daylight Saving Time boundary.

Example 4-8 Daylight Saving Time Is Not Calculated for Regions That Do Not Use Daylight Saving Time

Set the time zone region to UTC. UTC does not use Daylight Saving Time.

Was this page helpful?

```
ALTER SESSION SET TIME_ZONE='UTC';
```

 Copy

Truncate the global_orders table.

```
TRUNCATE TABLE global_orders;
```

 Copy

Insert values into the global_orders table.

```
INSERT INTO global_orders VALUES ( '28-OCT-00 11:24:54 PM',  
                                     TIMESTAMP '2000-10-28 23:24:54 ' );
```

 Copy

Add 8 hours to the columns.

```
SELECT orderdate1 + INTERVAL '8' HOUR, orderdate2 + INTERVAL '8' HOUR  
FROM global_orders;
```

 Copy

The following output results.

ORDERDATE1+INTERVAL '8' HOUR	ORDERDATE2+INTERVAL '8' HOUR
-----	-----
29-OCT-00 07.24.54.000000000 AM	29-OCT-00 07.24.54.000000000 AM UTC

 Copy

The times are the same because Daylight Saving Time is not calculated for the UTC time zone region.

Was this page helpful?