Manual: SQL Reference

Datetime and Interval Datatypes

The datetime datatypes are DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE. Values of datetime datatypes are sometimes called datetimes. The interval datatypes are INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND. Values of interval datatypes are sometimes called intervals.

Both datetimes and intervals are made up of fields. The values of these fields determine the value of the datatype. Table 2–4 lists the datetime fields and their possible values for datetimes and intervals.

To avoid unexpected results in your DML operations on datetime data, you can verify the database and session time zones by querying the built-in SQL functions DBTIMEZONE and SESSIONTIMEZONE. If the time zones have not been set manually, Oracle Database uses the operating system time zone by default. If the operating system time zone is not a valid Oracle time zone, then Oracle uses UTC as the default value.

Table 2–4 Datetime Fields and Values

Datetime Field	Valid Values for Datetime	Valid Values for INTERVAL
YEAR	-4712 to 9999 (excluding year 0)	Any positive or negative integer
MONTH	01 to 12	0 to 11
DAY	01 to 31 (limited by the values of MONTH and YEAR, according to the rules of the current NLS calendar parameter)	Any positive or negative integer
HOUR	00 to 23	0 to 23
MINUTE	00 to 59	0 to 59
SECOND	00 to 59.9(n), where 9(n) is the precision of time fractional seconds. The 9(n) portion is not applicable for DATE.	0 to 59.9(n), where 9(n) is the precision of interval fractional seconds
TIMEZONE_HOUR	-12 to 14 (This range accommodates daylight saving time changes.) Not applicable for DATE or TIMESTAMP.	Not applicable

Table 2–4 (Cont.) Datetime Fields and Values

Datetime Field	Valid Values for Datetime	Valid Values for INTERVAL
TIMEZONE_MINUTE (See note at end of table)	00 to 59. Not applicable for DATE or TIMESTAMP.	Not applicable
TIMEZONE_REGION	Query the TZNAME column of the V$TIMEZONE_NAMES data dictionary view. Not applicable for DATE or TIMESTAMP. For a complete listing of all timezone regions, refer to *Oracle Database Globalization Support Guide*.	Not applicable
TIMEZONE_ABBR	Query the TZABBREV column of the V$TIMEZONE_NAMES data dictionary view. Not applicable for DATE or TIMESTAMP.	Not applicable

Note: TIMEZONE_HOUR and TIMEZONE_MINUTE are specified together and interpreted as an entity in the format +|- hh:mm, with values ranging from -12:59 to +14:00. Please refer to Oracle Data Provider for .NET Developer's Guide for information on specifying time zone values for that API.

DATE Datatype

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

You can specify a DATE value as a literal, or you can convert a character or numeric value to a date value with the TO_DATE function.

Using Julian Days A Julian day number is the number of days since January 1, 4712 BC. Julian days allow continuous dating from a common reference. You can use the date format model "J" with date functions TO_DATE and TO_CHAR to convert between Oracle DATE values and their Julian equivalents.

The default date values are determined as follows:

■ The year is the current year, as returned by SYSDATE.

■ The month is the current month, as returned by SYSDATE.

■ The day is 01 (the first day of the month).

■ The hour, minute, and second are all 0.

These default values are used in a query that requests date values where the date itself is not specified, as in the following example, which is issued in the month of May:

SELECT TO_DATE(’2005’, ’YYYY’) FROM DUAL;

TO_DATE('

01-MAY-05

Example This statement returns the Julian equivalent of January 1, 1997:

SELECT TO_CHAR(TO_DATE('01-01-1997', 'MM-DD-YYYY'),'J') FROM DUAL;

TO_CHAR

2450450

TIMESTAMP Datatype

The TIMESTAMP datatype is an extension of the DATE datatype. It stores the year, month, and day of the DATE datatype, plus hour, minute, and second values. This datatype is useful for storing precise time values. Specify the TIMESTAMP datatype as follows:

TIMESTAMP [(fractional_seconds_precision)]

where fractional_seconds_precision optionally specifies the number of digits Oracle stores in the fractional part of the SECOND datetime field. When you create a column of this datatype, the value can be a number in the range 0 to 9. The default is 6.

TIMESTAMP WITH TIME ZONE Datatype

TIMESTAMP WITH TIME ZONE is a variant of TIMESTAMP that includes a 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). This datatype is useful for collecting and evaluating date information across geographic regions.

Specify the TIMESTAMP WITH TIME ZONE datatype as follows:

TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE

where fractional_seconds_precision optionally specifies the number of digits Oracle stores in the fractional part of the SECOND datetime field. When you create a column of this datatype, the value can be a number in the range 0 to 9. The default is 6.

TIMESTAMP WITH LOCAL TIME ZONE Datatype

TIMESTAMP WITH LOCAL TIME ZONE is another variant of TIMESTAMP that includes a time zone offset in its value. It differs from TIMESTAMP WITH TIME ZONE in that 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 a user retrieves the data, Oracle returns it in the user's 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). This datatype is useful for displaying date information in the time zone of the client system in a two-tier application.

Specify the TIMESTAMP WITH LOCAL TIME ZONE datatype as follows:

TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE

where fractional_seconds_precision optionally specifies the number of digits Oracle stores in the fractional part of the SECOND datetime field. When you create a column of this datatype, the value can be a number in the range 0 to 9. The default is 6.

INTERVAL YEAR TO MONTH Datatype

INTERVAL YEAR TO MONTH stores a period of time using the YEAR and MONTH datetime fields. This datatype is useful for representing the difference between two datetime values when only the year and month values are significant.

Specify INTERVAL YEAR TO MONTH as follows:

INTERVAL YEAR [(year_precision)] TO MONTH

where year_precision is the number of digits in the YEAR datetime field. The default value of year_precision is 2.

INTERVAL DAY TO SECOND Datatype

INTERVAL DAY TO SECOND stores a period of time in terms of days, hours, minutes, and seconds. This datatype is useful for representing the precise difference between two datetime values.

Specify this datatype as follows:

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

where

■ 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.

Datetime/Interval Arithmetic

You can perform a number of 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. Oracle calculates the results based on the following rules:

■ You can use NUMBER constants in arithmetic operations on date and timestamp values, but not interval values. Oracle internally converts timestamp values to date values and interprets NUMBER constants in arithmetic datetime and interval expressions as numbers of days. For example, SYSDATE + 1 is tomorrow. SYSDATE - 7 is one week ago. SYSDATE + (10/1440) is ten minutes from now. Subtracting the hire_date column of the sample table employees from SYSDATE returns the number of days since each employee was hired. You cannot multiply or divide date or timestamp values.

■ Oracle implicitly converts BINARY_FLOAT and BINARY_DOUBLE operands to NUMBER.

■ Each DATE value contains a time component, and the result of many date operations include 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 built-in functions for common operations on DATE data. For example, the MONTHS_ BETWEEN function returns the number of months between two dates. The fractional portion of the result represents that portion of a 31-day month.

■ If one operand is a DATE value or a numeric value (neither of which contains time zone or fractional seconds components), then:

– Oracle implicitly converts the other operand to DATE data. (The exception is multiplication of a numeric value times an interval, which returns an interval.)

– If the other operand has a time zone value, then Oracle uses the session time zone in the returned value.

-If the other operand has a fractional seconds value, then the fractional seconds value is lost.

■ When you pass a timestamp, interval, or numeric value to a built-in function that was designed only for the DATE datatype, Oracle implicitly converts the non-DATE value to a DATE value. Please refer to "Datetime Functions" on page 5-4 for information on which functions cause implicit conversion to DATE.

■ When interval calculations return a datetime value, the result must be an actual datetime value or the database returns an error. For example, the next two statements return errors:
SELECT TO_DATE('31-AUG-2004','DD-MON-YYYY') + TO_YMINTERVAL('0-1') FROM DUAL;

SELECT TO_DATE('29-FEB-2004','DD-MON-YYYY') + TO_YMINTERVAL('1-0') FROM DUAL;

The first fails because adding one month to a 31-day month would result in September 31, which is not a valid date. The second fails because adding one year to a date that exists only every four years is not valid. However, the next statement succeeds, because adding four years to a February 29 date is valid:

SELECT TO_DATE('29-FEB-2004', 'DD-MON-YYYY') + TO_YMINTERVAL('4-0') FROM DUAL;
TO_DATE(

' --------- 29-FEB-08

■ Oracle performs all timestamp arithmetic in UTC time. For TIMESTAMP WITH LOCAL TIME ZONE, Oracle 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, the datetime value is always in UTC, so no conversion is necessary.

Table 2–5 Matrix of Datetime Arithmetic

Operand & Operator	DATE	TIMESTAMP	INTERVAL	Numeric
DATE	—	—	—	—
+	–	–	DATE	DATE
-	DATE	DATE	DATE	DATE
*	–	–	–	–
/	–	–	–	–
TIMESTAMP	—	—	—	—
+	–	–	TIMESTAMP	–
-	INTERVAL	INTERVAL	TIMESTAMP	TIMESTAMP
*	–	–	–	–
/	–	–	–	–
INTERVAL	—	—	—	—
+	DATE	TIMESTAMP	INTERVAL	–
-	–	–	INTERVAL	–
*	–	–	–	INTERVAL

Table 2–5 (Cont.) Matrix of Datetime Arithmetic

Operand & Operator	DATE	TIMESTAMP	INTERVAL	Numeric
/	–	–	–	INTERVAL
Numeric	—	—	—	—
+	DATE	DATE	–	NA
-	–	–	–	NA
*	–	–	INTERVAL	NA
/	–	–	–	NA

Support for Daylight Saving

Times Oracle Database automatically determines, for any given time zone region, whether daylight saving is in effect and returns local time values accordingly. The datetime value is sufficient for Oracle to determine whether daylight saving time is in effect for a given region in all cases except boundary cases. A boundary case occurs during the period when daylight saving goes into or comes out of effect. For example, in the US-Pacific region, when daylight saving goes into effect, the time changes from 2:00 a.m. to 3:00 a.m. The one hour interval between 2 and 3 a.m. does not exist. When daylight saving goes out of effect, the time changes from 2:00 a.m. back to 1:00 a.m., and the one-hour interval between 1 and 2 a.m. is repeated.

To resolve these boundary cases, Oracle uses the TZR and TZD format elements, as described in Table 2–15. 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 information. Examples are 'PST' for US/Pacific standard time and 'PDT' for US/Pacific daylight time. To see a listing of valid values for the TZR and TZD format elements, query the TZNAME and TZABBREV columns of the V$TIMEZONE_NAMES dynamic performance view.

Timezone region names are needed by the daylight saving feature. The region names are stored in two time zone files. The default time zone file is the complete (larger) file containing all time zones. The other time zone file is a small file containing only the most common time zones to maximize performance. If your time zone is in the small file, and you want to maximize performance, then you must provide a path to the small file by way of the ORA_TZFILE environment variable. Please refer to Oracle Database Administrator's Guide for more information about setting the ORA_TZFILE environment variable. For a complete listing of the timezone region names in both files, please refer to Oracle Database Globalization Support Guide.

Datetime and Interval Examples ** (Ver en Manual, pag 2.22 y 2.23)

Datetime Literals

Oracle Database supports four datetime datatypes: DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE.

Date Literals You can specify a DATE value as a string literal, or you can convert a character or numeric value to a date value with the TO_DATE function. DATE literals are the only case in which Oracle Database accepts a TO_DATE expression in place of a string literal.

To specify a DATE value as a literal, you must use the Gregorian calendar. You can specify an ANSI literal, as shown in this example:

DATE '1998-12-25'

The ANSI date literal contains no time portion, and must be specified in exactly this format ('YYYY-MM-DD'). Alternatively you can specify an Oracle date value, as in the following example:

TO_DATE('98-DEC-25 17:30','YY-MON-DD HH24:MI')

The default date format for an Oracle DATE value is specified by the initialization parameter NLS_DATE_FORMAT. This example date format includes a two-digit number for the day of the month, an abbreviation of the month name, the last two digits of the year, and a 24-hour time designation.

Oracle 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 (00:00:00 or 12:00:00 for 24-hour and 12-hour clock time, respectively). If you specify a date value without a date, then the default date is the first day of the current month.

Oracle DATE columns always contain both the date and time fields. Therefore, if you query a DATE column, then you must either specify the time field in your query or ensure that the time fields in the DATE column are set to midnight. Otherwise, Oracle may not return the query results you expect. You can use the TRUNC (date) function to set the time field to midnight, or you can include a greater-than or less-than condition in the query instead of an equality or inequality condition.

Here are some examples that assume a table my_table with a number column row_ num and a DATE column datecol:

INSERT INTO my_table VALUES (1, SYSDATE);

INSERT INTO my_table VALUES (2, TRUNC(SYSDATE));

SELECT * FROM my_table;

ROW_NUM DATECOL

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

1 03-OCT-02

2 03-OCT-02 S

ELECT * FROM my_table WHERE datecol = TO_DATE('03-OCT-02','DD-MON-YY');

ROW_NUM DATECOL

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

2 03-OCT-02

SELECT * FROM my_table WHERE datecol > TO_DATE('02-OCT-02', 'DD-MON-YY');

ROW_NUM DATECOL

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

1 03-OCT-02

2 03-OCT-02

If you know that the time fields of your DATE column are set to midnight, then you can query your DATE column as shown in the immediately preceding example, or by using the DATE literal:

SELECT * FROM my_table WHERE datecol = DATE '2002-10-03';

However, if the DATE column contains values other than midnight, then you must filter out the time fields in the query to get the correct result. For example:

SELECT * FROM my_table WHERE TRUNC(datecol) = DATE '2002-10-03';

Oracle applies the TRUNC function to each row in the query, so performance is better if you ensure the midnight value of the time fields in your data. To ensure that the time fields are set to midnight, use one of the following methods during inserts and updates:

■ Use the TO_DATE function to mask out the time fields: I

NSERT INTO my_table VALUES (3, TO_DATE('3-OCT-2002','DD-MON-YYYY'));

■ Use the DATE literal: INSERT INTO my_table VALUES (4, '03-OCT-02');

■ Use the TRUNC function: INSERT INTO my_table VALUES (5, TRUNC(SYSDATE)); The date function SYSDATE returns the current system date and time. The function CURRENT_DATE returns the current session date.

TIMESTAMP Literals The TIMESTAMP datatype stores year, month, day, hour, minute, and second, and fractional second values. When you specify TIMESTAMP as a literal, the fractional_seconds_precision value can be any number of digits up to 9, as follows:

TIMESTAMP ’1997-01-31 09:26:50.124’

TIMESTAMP WITH TIME ZONE Literals The TIMESTAMP WITH TIME ZONE datatype is a variant of TIMESTAMP that includes a time zone offset. When you specify TIMESTAMP WITH TIME ZONE as a literal, the fractional_seconds_precision value can be any number of digits up to 9. For example:

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

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

TIMESTAMP '1999-04-15 8:00:00 -8:00'

is the same as

TIMESTAMP '1999-04-15 11:00:00 -5:00'

That is, 8:00 a.m. Pacific Standard Time is the same as 11:00 a.m. Eastern Standard Time.

You can replace the UTC offset with the TZR (time zone region) format element. For example, the following example has the same value as the preceding example:

TIMESTAMP '1999-04-15 8:00:00 US/Pacific'

To eliminate the ambiguity of boundary cases when the daylight saving time switches, use both the TZR and a corresponding TZD format element. The following example ensures that the preceding example will return a daylight saving time value:

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

You can also express the time zone offset using a datetime expression:

SELECT TIMESTAMP ’1999-10-29 01:30:00’ AT TIME ZONE ’US/Pacific’ FROM DUAL;

If you do not add the TZD format element, and the datetime value is ambiguous, then Oracle returns an error if you have the ERROR_ON_OVERLAP_TIME session parameter set to TRUE. If that parameter is set to FALSE, then Oracle interprets the ambiguous datetime as standard time in the specified region.

TIMESTAMP WITH LOCAL TIME ZONE Literals The TIMESTAMP WITH LOCAL TIME ZONE datatype differs from TIMESTAMP WITH TIME ZONE in that data stored in the database is normalized to the database time zone. The time zone offset is not stored as part of the column data. There is no literal for TIMESTAMP WITH LOCAL TIME ZONE. Rather, you represent values of this datatype using any of the other valid datetime literals. The table that follows shows some of the formats you can use to insert a value into a TIMESTAMP WITH LOCAL TIME ZONE column, along with the corresponding value returned by a query.

Value Specified in INSERT Statement	Value Returned by Query
'19-FEB-2004'	19-FEB-2004.00.00.000000 AM
SYSTIMESTAMP	19-FEB-04 02.54.36.497659 PM
TO_TIMESTAMP('19-FEB-2004', 'DD-MON-YYYY');	19-FEB-04 12.00.00.000000 AM
SYSDATE	19-FEB-04 02.55.29.000000 PM
TO_DATE('19-FEB-2004', 'DD-MON-YYYY');	19-FEB-04 12.00.00.000000 AM
TIMESTAMP'2004-02-19 8:00:00 US/Pacific');	19-FEB-04 08.00.00.000000 AM

Notice that if the value specified does not include a time component (either explicitly or implicitly, then the value returned defaults to midnight

Interval Literals

An interval literal specifies a period of time. You can specify these differences in terms of years and months, or in terms of days, hours, minutes, and seconds. Oracle Database supports two types of interval literals, YEAR TO MONTH and DAY TO SECOND.

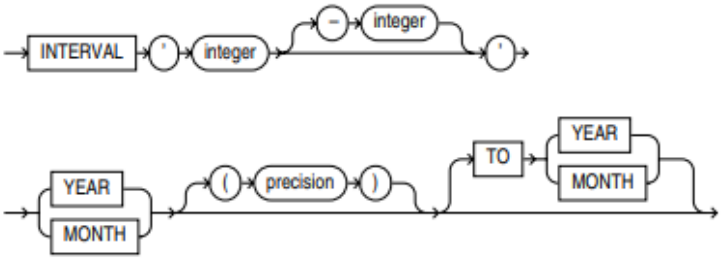
Each type contains a leading field and may contain a trailing field. The leading field defines the basic unit of date or time being measured. The trailing field defines the smallest increment of the basic unit being considered. For example, a YEAR TO MONTH interval considers an interval of years to the nearest month. A DAY TO MINUTE interval considers an interval of days to the nearest minute.

If you have date data in numeric form, then you can use the NUMTOYMINTERVAL or NUMTODSINTERVAL conversion function to convert the numeric data into interval values.

Interval literals are used primarily with analytic functions.

INTERVAL YEAR TO MONTH

Specify YEAR TO MONTH interval literals using the following syntax:



where

■ 'integer [-integer]' specifies integer values for the leading and optional trailing field of the literal. If the leading field is YEAR and the trailing field is MONTH, then the range of integer values for the month field is 0 to 11.

■ precision is the maximum number of digits in the leading field. The valid range of the leading field precision is 0 to 9 and its default value is 2.

Restriction on the Leading Field If you specify a trailing field, it must be less significant than the leading field. For example, INTERVAL '0-1' MONTH TO YEAR is not valid.

The following INTERVAL YEAR TO MONTH literal indicates an interval of 123 years, 2 months:

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

Examples of the other forms of the literal follow, including some abbreviated versions:

Form of Interval Literal	Interpretation
`INTERVAL '123-2' YEAR(3) TO MONTH`	An interval of 123 years, 2 months. You must specify the leading field precision if it is greater than the default of 2 digits.
`INTERVAL '123' YEAR(3)`	An interval of 123 years 0 months.
`INTERVAL '300' MONTH(3)`	An interval of 300 months.

Form of Interval Literal	Interpretation
`INTERVAL '4' YEAR`	Maps to `INTERVAL '4-0' YEAR TO MONTH` and indicates 4 years.
`INTERVAL '50' MONTH`	Maps to `INTERVAL '4-2' YEAR TO MONTH` and indicates 50 months or 4 years 2 months.
`INTERVAL '123' YEAR`	Returns an error, because the default precision is 2, and '123' has 3 digits.

You can add or subtract one INTERVAL YEAR TO MONTH literal to or from another to yield another INTERVAL YEAR TO MONTH literal. For example:

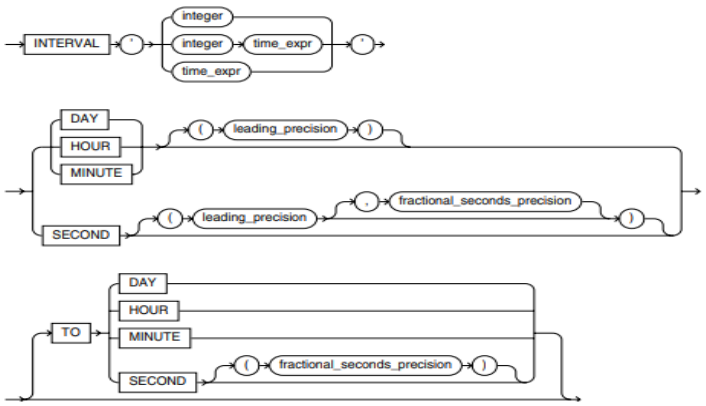
INTERVAL '5-3' YEAR TO MONTH + INTERVAL'20' MONTH =

INTERVAL '6-11' YEAR TO MONTH

INTERVAL DAY TO SECOND

Specify `DAY TO SECOND` interval literals using the following syntax:

***interval_day_to_second*::=**



where

- *integer* specifies the number of days. If this value contains more digits than the number specified by the leading precision, then Oracle returns an error.
- *time_expr* specifies a time in the format `HH[:MI[:SS[.n]]]` or `MI[:SS[.n]]` or `SS[.n]`, where *n* specifies the fractional part of a second. If *n* contains more digits than the number specified by *fractional_seconds_precision*, then *n* is rounded to the number of digits specified by the *fractional_seconds_precision* value. You can specify *time_expr* following an integer and a space only if the leading field is `DAY`.

■ leading_precision is the number of digits in the leading 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 1 to 9. The default is 6.

Restriction on the Leading Field: If you specify a trailing field, it must be less significant than the leading field. For example, INTERVAL MINUTE TO DAY is not valid. As a result of this restriction, if SECOND is the leading field, the interval literal cannot have any trailing field.

The valid range of values for the trailing field are as follows:

■ HOUR: 0 to 23

■ MINUTE: 0 to 59

■ SECOND: 0 to 59.999999999

Examples of the various forms of INTERVAL DAY TO SECOND literals follow, including some abbreviated versions:

Form of Interval Literal	Interpretation
`INTERVAL '4 5:12:10.222' DAY TO SECOND(3)`	4 days, 5 hours, 12 minutes, 10 seconds, and 222 thousandths of a second.
`INTERVAL '4 5:12' DAY TO MINUTE`	4 days, 5 hours and 12 minutes.
`INTERVAL '400 5' DAY(3) TO HOUR`	400 days 5 hours.
`INTERVAL '400' DAY(3)`	400 days.
`INTERVAL '11:12:10.2222222' HOUR TO SECOND(7)`	11 hours, 12 minutes, and 10.2222222 seconds.
`INTERVAL '11:20' HOUR TO MINUTE`	11 hours and 20 minutes.
`INTERVAL '10' HOUR`	10 hours.
`INTERVAL '10:22' MINUTE TO SECOND`	10 minutes 22 seconds.
`INTERVAL '10' MINUTE`	10 minutes.
`INTERVAL '4' DAY`	4 days.
`INTERVAL '25' HOUR`	25 hours.
`INTERVAL '40' MINUTE`	40 minutes.
`INTERVAL '120' HOUR(3)`	120 hours.
`INTERVAL '30.12345' SECOND(2,4)`	30.1235 seconds. The fractional second '12345' is rounded to '1235' because the precision is 4.

Datetime Format Models

You can use datetime format models in the following functions:

■ In the TO_* datetime functions to translate a character value that is in a format other than the default format into a datetime value. (The TO_* datetime functions are TO_CHAR, TO_DATE, TO_TIMESTAMP, TO_TIMESTAMP_TZ, TO_YMINTERVAL, and TO_DSINTERVAL.)

■ In the TO_CHAR function to translate a datetime value that is in a format other than the default format into a string (for example, to print the date from an application)

The total length of a datetime format model cannot exceed 22 characters.

The default datetime formats are specified either explicitly with the initialization parameter NLS_DATE_FORMAT or implicitly with the initialization parameter NLS_ TERRITORY. You can change the default datetime formats for your session with the ALTER SESSION statement.

Datetime Format Elements

A datetime format model is composed of one or more datetime format elements as listed in Table 2–19, " Attributes of the XMLFormat Object" on page 2-68.

■ For input format models, format items cannot appear twice, and format items that represent similar information cannot be combined. For example, you cannot use 'SYYYY' and 'BC' in the same format string.

■ Some of the datetime format elements cannot be used in the TO_* datetime functions, as noted in Table 2–19.

■ The following datetime format elements can be used in timestamp and interval format models, but not in the original DATE format model: FF, TZD, TZH, TZM, and TZR.

■ Many datetime format elements are blank padded to a specific length. Please refer to the format model modifier FM on page 2-64 for more information.

Uppercase Letters in Date Format Elements Capitalization in a spelled-out word, abbreviation, or Roman numeral follows capitalization in the corresponding format element. For example, the date format model 'DAY' produces capitalized words like 'MONDAY'; 'Day' produces 'Monday'; and 'day' produces 'monday'.

Punctuation and Character Literals in Datetime Format Models You can include these characters in a date format model:

■ Punctuation such as hyphens, slashes, commas, periods, and colons

■ Character literals, enclosed in double quotation marks These characters appear in the return value in the same location as they appear in the format model.

Tabla 2-15 Datetime Format Elements (Ver en Manual, pag 2.59)

Oracle returns an error if an alphanumeric character is found in the date string where a punctuation character is found in the format string. For example, the following format string returns an error:

TO_CHAR (TO_DATE('0297','MM/YY'), 'MM/YY')

Datetime Format Elements and Globalization Support

The functionality of some datetime format elements depends on the country and language in which you are using Oracle Database. For example, these datetime format elements return spelled values:

■ MONTH ■ MON ■ DAY ■ DY ■ BC or AD or B.C. or A.D. ■ AM or PM or A.M or P.M.

The language in which these values are returned is specified either explicitly with the initialization parameter NLS_DATE_LANGUAGE or implicitly with the initialization parameter NLS_LANGUAGE. The values returned by the YEAR and SYEAR datetime format elements are always in English.

The datetime format element D returns the number of the day of the week (1-7). The day of the week that is numbered 1 is specified implicitly by the initialization parameter NLS_TERRITORY.

ISO Standard Date Format Elements

Oracle calculates the values returned by the datetime format elements IYYY, IYY, IY, I, and IW according to the ISO standard. For information on the differences between these values and those returned by the datetime format elements YYYY, YYY, YY, Y, and WW, see the discussion of globalization support in Oracle Database Globalization Support Guide.

The RR Datetime Format Element

The RR datetime format element is similar to the YY datetime format element, but it provides additional flexibility for storing date values in other centuries. The RR datetime format element lets you store 20th century dates in the 21st century by specifying only the last two digits of the year.

If you use the TO_DATE function with the YY datetime format element, then the year returned always has the same first 2 digits as the current year. If you use the RR datetime format element instead, then the century of the return value varies according to the specified two-digit year and the last two digits of the current year.

That is:

■ If the specified two-digit year is 00 to 49, then

– If the last two digits of the current year are 00 to 49, then the returned year has the same first two digits as the current year.

– If the last two digits of the current year are 50 to 99, then the first 2 digits of the returned year are 1 greater than the first 2 digits of the current year.

■ If the specified two-digit year is 50 to 99, then

– If the last two digits of the current year are 00 to 49, then the first 2 digits of the returned year are 1 less than the first 2 digits of the current year.

– If the last two digits of the current year are 50 to 99, then the returned year has the same first two digits as the current year.

The following examples demonstrate the behavior of the RR datetime format element. RR Datetime Format Examples

RR Datetime Format Examples

Assume these queries are issued between 1950 and 1999:

```
SELECT TO_CHAR(TO_DATE('27-OCT-98', 'DD-MON-RR') ,'YYYY') "Year"
   FROM DUAL;

Year
----
1998

SELECT TO_CHAR(TO_DATE('27-OCT-17', 'DD-MON-RR') ,'YYYY') "Year"
   FROM DUAL;

Year
----
2017
```

Now assume these queries are issued between 2000 and 2049:

```
SELECT TO_CHAR(TO_DATE('27-OCT-98', 'DD-MON-RR') ,'YYYY') "Year"
   FROM DUAL;

Year
----
1998

SELECT TO_CHAR(TO_DATE('27-OCT-17', 'DD-MON-RR') ,'YYYY') "Year"
   FROM DUAL;

Year
----
```

2017

Note that the queries return the same values regardless of whether they are issued before or after the year 2000. The RR datetime format element lets you write SQL statements that will return the same values from years whose first two digits are different.

Datetime Format Element Suffixes

Table 2–16 Date Format Element Suffixes

Suffix	Meaning	Example Element	Example Value
TH	Ordinal Number	DDTH	4TH
SP	Spelled Number	DDSP	FOUR
SPTH or THSP	Spelled, ordinal number	DDSPTH	FOURTH

Notes on date format element suffixes: ■ When you add one of these suffixes to a datetime format element, the return value is always in English.

■ Datetime suffixes are valid only to format output. You cannot use them to insert a date into the database.

Datetime 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 DATE datatype (ADD_ MONTHS, CURRENT_DATE, LAST_DAY, NEW_TIME, and NEXT_DAY). If you provide a timestamp value as their argument, Oracle Database internally converts the input type to a DATE value and returns a DATE value. The exceptions are the MONTHS_BETWEEN function, which returns a number, and the ROUND and TRUNC functions, which do not accept timestamp or interval values at all.

The remaining datetime functions were designed to accept any of the three types of data (date, timestamp, and interval) and to return a value of one of these types.

The datetime functions are: ADD_MONTHS, CURRENT_DATE, CURRENT_TIMESTAMP DBTIMEZONE, EXTRACT (datetime) , FROM_TZ , LAST_DAY, LOCALTIMESTAMP MONTHS_BETWEEN, NEW_TIME, NEXT_DAY, NUMTODSINTERVAL, NUMTOYMINTERVAL ROUND (date), SESSIONTIMEZONE, SYS_EXTRACT_UTC, SYSDATE, SYSTIMESTAMP, TO_CHAR (datetime), TO_TIMESTAMP, TO_TIMESTAMP_TZ , TO_DSINTERVAL, TO_YMINTERVAL , TRUNC (date) , TZ_OFFSET

**Hay más de otros tipos (Conversion, Large Objects ….)

Selecting from the DUAL Table

DUAL is a table automatically created by Oracle Database along with the data dictionary. DUAL is in the schema of the user SYS but is accessible by the name DUAL to all users. It has one column, DUMMY, defined to be VARCHAR2(1), and contains one row with a value X. Selecting from the DUAL table is useful for computing a constant expression with the SELECT statement. Because DUAL has only one row, the constant is returned only once. Alternatively, you can select a constant, pseudocolumn, or expression from any table, but the value will be returned as many times as there are rows in the table

Manual: Application Developer´s Guide-Fundamentals

Database Administration Tasks Before Using Flashback Features

Before you can use flashback features in your application, you must perform the following administrative tasks to configure your database. Consult with your database administrator to perform these tasks:

■ Create an undo tablespace with enough space to keep the required data for flashback operations. The more often users update the data, the more space is required. Calculating the space requirements is usually performed by a database administrator. You can find the calculation formula in the Oracle Database Administrator's Guide.

■ Enable Automatic Undo Management, as explained in Oracle Database Administrator's Guide. In particular, you must set the following database initialization parameters:

– UNDO_MANAGEMENT

– UNDO_TABLESPACE

Note that for an undo tablespace with a fixed size, Oracle Database automatically performs the following actions:

– Tunes the system to give the best possible undo retention for the undo tablespace.

For an automatically extensible undo tablespace, Oracle Database retains undo data longer than the longest query duration as well as the low threshold of undo retention specified by the UNDO_RETENTION parameter.

■ Specify the RETENTION GUARANTEE clause for the undo tablespace to ensure that unexpired undo is not discarded. Setting UNDO_RETENTION is not, by itself, a strict guarantee. If the system is under space pressure, then Oracle can overwrite unexpired undo with freshly generated undo. Specifying RETENTION GUARANTEE prevents this behavior.

■ Grant flashback privileges to users, roles, or applications that need to use flashback features as follows:

– For the DBMS_FLASHBACK package, grant the EXECUTE privilege on DBMS_FLASHBACK to provide access to the features in this package.

– For Flashback Query and Flashback Version Query, grant FLASHBACK and SELECT privileges on specific objects to be accessed during queries or grant the FLASHBACK ANY TABLE privilege to allow queries on all tables.

– For Flashback Transaction Query, grant the SELECT ANY TRANSACTION privilege.

– For Execution of undo SQL code, grant SELECT, UPDATE, DELETE, and INSERT privileges for specific tables, as appropriate, to permit execution of undo SQL code retrieved by a Flashback Transaction Query.

■ To use the Flashback Transaction Query feature in Oracle Database 10g, the database must be running with version 10.0 compatibility, and must have supplemental logging turned on with the following SQL statement:

ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;

■ To enable flashback operations on specific LOB columns of a table, use the ALTER TABLE command with the RETENTION option. Because undo data for LOB columns can be voluminous, you must define which LOB columns to use with flashback operations.

Using Flashback Query (SELECT ... AS OF)

You perform a Flashback Query by using a SELECT statement with an AS OF clause. You use a Flashback Query to retrieve data as it existed at some time in the past. The query explicitly references a past time by means of a timestamp or SCN. It returns committed data that was current at that point in time.

Potential uses of Flashback Query include:

■ Recovering lost data or undoing incorrect, committed changes. For example, if you mistakenly delete or update rows, and then commit them, you can immediately undo the mistake.

■ Comparing current data with the corresponding data at some time in the past. For example, you might run a daily report that shows the change in data from yesterday. You can compare individual rows of table data or find intersections or unions of sets of rows.

■ Checking the state of transactional data at a particular time. For example, you could verify the account balance of a certain day.

■ Simplifying application design, by removing the need to store some kinds of temporal data. By using a Flashback Query, you can retrieve past data directly from the database.

■ Applying packaged applications such as report generation tools to past data.

■ Providing self-service error correction for an application, thereby enabling users to undo and correct their errors.

** ***Examining Past data: Example*** (Ver en Manual, pag 10-5)

Tips for Using Flashback Query

Keep the following in mind when using a Flashback Query (SELECT ... AS OF):

■ You can specify or omit the AS OF clause for each table and specify different times for different tables. Use an AS OF clause in a query to perform DDL operations (such as creating and truncating tables) or DML operations (such as inserting and deleting) in the same session as the query.

■ To use the results of a Flashback Query in a DDL or DML statement that affects the current state of the database, use an AS OF clause inside an INSERT or CREATE TABLE AS SELECT statement.

■ When choosing whether to use a timestamp or an SCN in Flashback Query, remember that Oracle Database uses SCNs internally and maps these to timestamps at a granularity of 3 seconds. If a possible 3-second error (maximum) is important to a Flashback Query in your application, then use an SCN instead of a timestamp.

■ You can create a view that refers to past data by using the AS OF clause in the SELECT statement that defines the view. If you specify a relative time by subtracting from the current time on the database host, then the past time is recalculated for each query. For example:

```
CREATE VIEW hour_ago AS

SELECT * FROM employees AS OF

 TIMESTAMP (SYSTIMESTAMP - INTERVAL '60' MINUTE);

-- SYSTIMESTAMP refers to the time zone of the database host environment
```

■ You can use the AS OF clause in self-joins, or in set operations such as INTERSECT and MINUS, to extract or compare data from two different times. You can store the results by preceding a Flashback Query with a CREATE TABLE AS SELECT or INSERT INTO TABLE SELECT statement. For example, the following query reinserts into table employees the rows that existed an hour ago:

```
INSERT INTO employees

(SELECT * FROM employees AS OF

TIMESTAMP (SYSTIMESTAMP - INTERVAL '60' MINUTE))

 -- SYSTIMESTAMP refers to the time zone of the database host environment

 MINUS SELECT * FROM employees);
```

Using the DBMS_FLASHBACK Package

In general, the DBMS_FLASHBACK package provides the same functionality as Flashback Query, but Flashback Query is sometimes more convenient.

The DBMS_FLASHBACK package acts as a time machine: you can turn back the clock, carry out normal queries as if you were at that time in the past, and then return to the present. Because you can use the DBMS_FLASHBACK package to perform queries on past data without special clauses such as AS OF or VERSIONS BETWEEN, you can reuse existing PL/SQL code to query the database at times in the past.

You must have the EXECUTE privilege on the DBMS_FLASHBACK package.

To use the DBMS_FLASHBACK package in your PL/SQL code:

1. Call DBMS_FLASHBACK.ENABLE_AT_TIME or DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER to turn back the clock to a specified time in the past. Afterwards all queries retrieve data that was current at the specified time.

2. Perform normal queries, that is, without any special flashback-feature syntax such as AS OF. The database is automatically queried at the specified past time. Perform only queries; do not try to perform DDL or DML operations.

3. Call DBMS_FLASHBACK.DISABLE to return to the present. You must call DISABLE before calling ENABLE again for a different time. You cannot nest ENABLE /DISABLE pairs.

You can use a cursor to store the results of queries. To do this, open the cursor before calling DBMS_FLASHBACK.DISABLE. After storing the results and then calling DISABLE, you can do the following:

■ Perform INSERT or UPDATE operations to modify the current database state by using the stored results from the past.

■ Compare current data with the past data. After calling DISABLE, open a second cursor. Fetch from the first cursor to retrieve past data; fetch from the second cursor to retrieve current data. You can store the past data in a temporary table, and then use set operators such as MINUS or UNION to contrast or combine the past and current data.

You can call DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER at any time to obtain the current System Change Number (SCN). Note that the current SCN is always returned; this takes no account of previous calls to DBMS_FLASHBACK.ENABLE*.

Using ORA_ROWSCN

ORA_ROWSCN is a pseudocolumn of any table that is not fixed or external. It represents the SCN of the most recent change to a given row, that is, the latest COMMIT operation for the row. For example:

SELECT ora_rowscn, last_name, salary

FROM employees WHERE employee_id = 7788;

ORA_ROWSCN NAME SALARY

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

202553 Fudd 3000

The latest COMMIT operation for the row took place at approximately SCN 202553. You can use function SCN_TO_TIMESTAMP to convert an SCN, like ORA_ROWSCN, to the corresponding TIMESTAMP value.

ORA_SCN is in fact a conservative upper bound of the latest commit time: the actual commit SCN can be somewhat earlier. ORA_SCN is more precise (closer to the actual commit SCN) for a row-dependent table (created using CREATE TABLE with the ROWDEPENDENCIES clause).

Noteworthy uses of ORA_ROWSCN in application development include concurrency control and client cache invalidation. To see how you might use it in concurrency control, consider the following scenario.

Your application examines a row of data, and records the corresponding ORA_ROWSCN as 202553. Later, the application needs to update the row, but only if its record of the data is still accurate. That is, this particular update operation depends, logically, on the row not having been changed. The operation is therefore made conditional on the ORA_ROWSCN being still 202553. Here is an equivalent interactive command:

UPDATE employees

SET salary = salary + 100

WHERE employee_id = 7788 AND ora_rowscn = 202553;

0 rows updated.

The conditional update fails in this case, because the ORA_ROWSCN is no longer 202553. This means that some user or another application changed the row and performed a COMMIT more recently than the recorded ORA_ROWSCN.

Your application queries again to obtain the new row data and ORA_ROWSCN. Suppose that the ORA_ROWSCN is now 415639. The application tries the conditional update again, using the new ORA_ROWSCN. This time, the update succeeds, and it is committed. Here is an interactive equivalent:

SQL> UPDATE employees SET salary = salary + 100

WHERE empno = 7788 AND ora_rowscn = 415639;

1 row updated.

SQL> COMMIT;

Commit complete.

SQL> SELECT ora_rowscn, name, salary FROM employees WHERE empno = 7788;

ORA_ROWSCN NAME SALARY

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

465461 Fudd 3100

The SCN corresponding to the new COMMIT is 465461.

Besides using ORA_ROWSCN in an UPDATE statement WHERE clause, you can use it in a DELETE statement WHERE clause or the AS OF clause of a Flashback Query.

Using Flashback Version Query

You use a Flashback Version Query to retrieve the different versions of specific rows that existed during a given time interval. A new row version is created whenever a COMMIT statement is executed.

You specify a Flashback Version Query using the VERSIONS BETWEEN clause of the SELECT statement. Here is the syntax:

VERSIONS {BETWEEN {SCN | TIMESTAMP} start AND end}

where start and end are expressions representing the start and end of the time interval to be queried, respectively. The interval is closed at both ends: the upper and lower limits specified (start and end) are both included in the time interval.

The Flashback Version Query returns a table with a row for each version of the row that existed at any time during the time interval you specify. Each row in the table includes pseudocolumns of metadata about the row version, described in Table 10–1. This information can reveal when and how a particular change (perhaps erroneous) occurred to your database.

Table 10–1 Flashback Version Query Row Data Pseudocolumns

Pseudocolumn Name	Description
VERSIONS_STARTSCN VERSIONS_STARTTIME	Starting System Change Number (SCN) or TIMESTAMP when the row version was created. This identifies the time when the data first took on the values reflected in the row version. You can use this to identify the past target time for a Flashback Table or Flashback Query operation. If this is NULL, then the row version was created before the lower time bound of the query BETWEEN clause.
VERSIONS_ENDSCN VERSIONS_ENDTIME	SCN or TIMESTAMP when the row version expired. This identifies the row expiration time. If this is NULL, then either the row version was still current at the time of the query or the row corresponds to a DELETE operation.
VERSIONS_XID	Identifier of the transaction that created the row version.
VERSIONS_OPERATION	Operation performed by the transaction: I for insertion, D for deletion, or U for update. The version is that of the row that was inserted, deleted, or updated; that is, the row *after* an INSERT operation, the row *before* a DELETE operation, or the row affected by an UPDATE operation. *Note:* For user updates of an index key, a Flashback Version Query may treat an UPDATE operation as two operations, DELETE plus INSERT, represented as two version rows with a D followed by an I VERSIONS_OPERATION.

A given row version is valid starting at its time VERSIONS_START* up to, but not including, its time VERSIONS_END*. That is, it is valid for any time *t* such that VERSIONS_START* <= *t* < VERSIONS_END*. For example, the following output indicates that the salary was 10243 from September 9, 2002, included, to November 25, 2003, not included.

```
VERSIONS_START_TIME      VERSIONS_END_TIME      SALARY
-------------------      -----------------      ------
09-SEP-2003              25-NOV-2003            10243
```

Here is a typical Flashback Version Query:

```
SELECT versions_startscn, versions_starttime,
       versions_endscn, versions_endtime,
       versions_xid, versions_operation,
       name, salary
  FROM employees
  VERSIONS BETWEEN TIMESTAMP
       TO_TIMESTAMP('2003-07-18 14:00:00', 'YYYY-MM-DD HH24:MI:SS')
  AND TO_TIMESTAMP('2003-07-18 17:00:00', 'YYYY-MM-DD HH24:MI:SS')
  WHERE name = 'JOE';
```

Pseudocolumn VERSIONS_XID provides a unique identifier for the transaction that put the data in that state. You can use this value in connection with a Flashback Transaction Query to locate metadata about this transaction in the FLASHBACK_TRANSACTION_QUERY view, including the SQL required to undo the row change and the user responsible for the change

Manual: Administrator's Guide

What Is Undo?

Every Oracle Database must have a method of maintaining information that is used to roll back, or undo, changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. These records are collectively referred to as undo.

Undo records are used to:

■ Roll back transactions when a ROLLBACK statement is issued

■ Recover the database

■ Provide read consistency

■ Analyze data as of an earlier point in time by using Oracle Flashback Query

■ Recover from logical corruptions using Oracle Flashback features

When a ROLLBACK statement is issued, undo records are used to undo changes that were made to the database by the uncommitted transaction. During database recovery, undo records are used to undo any uncommitted changes applied from the redo log to the datafiles. Undo records provide read consistency by maintaining the before image of the data for users who are accessing the data at the same time that another user is changing it.

Overview of Automatic Undo Management

Oracle provides a fully automated mechanism, referred to as automatic undo management, for managing undo information and space. In this management mode, you create an undo tablespace, and the server automatically manages undo segments and space among the various active sessions.

You set the UNDO_MANAGEMENT initialization parameter to AUTO to enable automatic undo management. A default undo tablespace is then created at database creation. An undo tablespace can also be created explicitly.

When the instance starts, the database automatically selects the first available undo tablespace. If no undo tablespace is available, then the instance starts without an undo tablespace and stores undo records in the SYSTEM tablespace. This is not recommended in normal circumstances, and an alert message is written to the alert log file to warn that the system is running without an undo tablespace.

If the database contains multiple undo tablespaces, you can optionally specify at startup that you want to use a specific undo tablespace. This is done by setting the UNDO_TABLESPACE initialization parameter, as shown in this example:

UNDO_TABLESPACE = undotbs_01

In this case, if you have not already created the undo tablespace (in this example, undotbs_01), the STARTUP command fails. The UNDO_TABLESPACE parameter can be used to assign a specific undo tablespace to an instance in an Oracle Real Application Clusters environment.

The following is a summary of the initialization parameters for automatic undo management:

Initialization Parameter	Description
UNDO_MANAGEMENT	If AUTO, use automatic undo management. The default is MANUAL.
UNDO_TABLESPACE	An optional dynamic parameter specifying the name of an undo tablespace. This parameter should be used only when the database has multiple undo tablespaces and you want to direct the database instance to use a particular undo tablespace.

When automatic undo management is enabled, if the initialization parameter file contains parameters relating to manual undo management, they are ignored.

Undo Retention

After a transaction is committed, undo data is no longer needed for rollback or transaction recovery purposes. However, for consistent read purposes, long-running queries may require this old undo information for producing older images of data blocks. Furthermore, the success of several Oracle Flashback features can also depend upon the availability of older undo information. For these reasons, it is desirable to retain the old undo information for as long as possible.

When automatic undo management is enabled, there is always a current undo retention period, which is the minimum amount of time that Oracle Database attempts to retain old undo information before overwriting it. Old (committed) undo information that is older than the current undo retention period is said to be expired. Old undo information with an age that is less than the current undo retention period is said to be unexpired.

Oracle Database automatically tunes the undo retention period based on undo tablespace size and system activity. You can specify a minimum undo retention period (in seconds) by setting the UNDO_RETENTION initialization parameter. The database makes its best effort to honor the specified minimum undo retention period, provided that the undo tablespace has space available for new transactions. When available space for new transactions becomes short, the database begins to overwrite expired undo. If the undo tablespace has no space for new transactions after all expired undo is overwritten, the database may begin overwriting unexpired undo information. If any of this overwritten undo information is required for consistent read in a current long-running query, the query could fail with the snapshot too old error message.

The following points explain the exact impact of the UNDO_RETENTION parameter on undo retention:

■ The UNDO_RETENTION parameter is ignored for a fixed size undo tablespace. The database may overwrite unexpired undo information when tablespace space becomes low.

■ For an undo tablespace with the AUTOEXTEND option enabled, the database attempts to honor the minimum retention period specified by UNDO_RETENTION. When space is low, instead of overwriting unexpired undo information, the tablespace auto-extends. If the MAXSIZE clause is specified for an auto-extending undo tablespace, when the maximum size is reached, the database may begin to overwrite unexpired undo information.

Retention Guarantee

To guarantee the success of long-running queries or Oracle Flashback operations, you can enable retention guarantee. If retention guarantee is enabled, the specified minimum undo retention is guaranteed; the database never overwrites unexpired undo data even if it means that transactions fail due to lack of space in the undo tablespace. If retention guarantee is not enabled, the database can overwrite unexpired undo when space is low, thus lowering the undo retention for the system. This option is disabled by default.

You enable retention guarantee by specifying the RETENTION GUARANTEE clause for the undo tablespace when you create it with either the CREATE DATABASE or CREATE UNDO TABLESPACE statement. Or, you can later specify this clause in an ALTER TABLESPACE statement. You disable retention guarantee with the RETENTION NOGUARANTEE clause.

You can use the DBA_TABLESPACES view to determine the retention guarantee setting for the undo tablespace. A column named RETENTION contains a value of GUARANTEE, NOGUARANTEE, or NOT APPLY (used for tablespaces other than the undo tablespace).

Automatic Tuning of Undo Retention

Oracle Database automatically tunes the undo retention period based on how the undo tablespace is configured.

■ If the undo tablespace is fixed size, the database tunes the retention period for the best possible undo retention for that tablespace size and the current system load. This tuned retention period can be significantly greater than the specified minimum retention period.

■ If the undo tablespace is configured with the AUTOEXTEND option, the database tunes the undo retention period to be somewhat longer than the longest-running query on the system at that time. Again, this tuned retention period can be greater than the specified minimum retention period.

You can determine the current retention period by querying the TUNED_UNDORETENTION column of the V$UNDOSTAT view. This view contains one row for each 10-minute statistics collection interval over the last 4 days. (Beyond 4 days, the data is available in the DBA_HIST_UNDOSTAT view.) TUNED_UNDORETENTION is given in seconds.

```
select to_char(begin_time, 'DD-MON-RR HH24:MI') begin_time,
to_char(end_time, 'DD-MON-RR HH24:MI') end_time, tuned_undoretention
from v$undostat order by end_time;

BEGIN_TIME      END_TIME        TUNED_UNDORETENTION
--------------- --------------- -------------------
04-FEB-05 00:01 04-FEB-05 00:11               12100
      ...
07-FEB-05 23:21 07-FEB-05 23:31               86700
07-FEB-05 23:31 07-FEB-05 23:41               86700
07-FEB-05 23:41 07-FEB-05 23:51               86700
07-FEB-05 23:51 07-FEB-05 23:52               86700

576 rows selected.
```

Undo Retention Tuning and Alert Thresholds For a fixed size undo tablespace, the database calculates the maximum undo retention period based on database statistics and on the size of the undo tablespace. For optimal undo management, rather than tuning based on 100% of the tablespace size, the database tunes the undo retention period based on 85% of the tablespace size, or on the warning alert threshold percentage for space used, whichever is lower. (The warning alert threshold defaults to 85%, but can be changed.) Therefore, if you set the warning alert threshold of the undo tablespace below 85%, this may reduce the tuned length of the undo retention period.

Setting the Undo Retention Period

You set the undo retention period by setting the UNDO_RETENTION initialization parameter. This parameter specifies the desired minimum undo retention period in seconds. As described in "Undo Retention" on page 10-3, the current undo retention period may be automatically tuned to be greater than UNDO_RETENTION, or, unless retention guarantee is enabled, less than UNDO_RETENTION if space is low.

To set the undo retention period:

■ Do one of the following:

– Set UNDO_RETENTION in the initialization parameter file. UNDO_RETENTION = 1800

– Change UNDO_RETENTION at any time using the ALTER SYSTEM statement: ALTER SYSTEM SET UNDO_RETENTION = 2400;

The effect of an UNDO_RETENTION parameter change is immediate, but it can only be honored if the current undo tablespace has enough space.

Manual: PL/SQL Packages and Types Reference

SLEEP Procedure

This procedure suspends the session for a given period of time.

Syntax DBMS_LOCK.SLEEP (seconds IN NUMBER);

SEED Procedures

This procedure resets the seed.

Syntax DBMS_RANDOM.SEED (seed IN BINARY_INTEGER);

DBMS_RANDOM.SEED (seed IN VARCHAR2);

Pragmas PRAGMA restrict_references (seed, WNDS);

VALUE Functions

The basic function gets a random number, greater than or equal to 0 and less than 1, with 38 digits to the right of the decimal (38-digit precision). Alternatively, you can get a random Oracle number x, where x is greater than or equal to low and less than high.

Syntax DBMS_RANDOM.VALUE RETURN NUMBER;

DBMS_RANDOM.VALUE(low IN NUMBER, high IN NUMBER) RETURN NUMBER;

Manual: Concepts

Temporary Tables

In addition to permanent tables, Oracle can create temporary tables to hold session-private data that exists only for the duration of a transaction or session.

The CREATE GLOBAL TEMPORARY TABLE statement creates a temporary table that can be transaction-specific or session-specific. For transaction-specific temporary tables, data exists for the duration of the transaction. For session-specific temporary tables, data exists for the duration of the session. Data in a temporary table is private to the session. Each session can only see and modify its own data. DML locks are not acquired on the data of the temporary tables. The LOCK statement has no effect on a temporary table, because each session has its own private data.

A TRUNCATE statement issued on a session-specific temporary table truncates data in its own session. It does not truncate the data of other sessions that are using the same table.

DML statements on temporary tables do not generate redo logs for the data changes. However, undo logs for the data and redo logs for the undo logs are generated. Data from the temporary table is automatically dropped in the case of session termination, either when the user logs off or when the session terminates abnormally such as during a session or instance failure.

You can create indexes for temporary tables using the CREATE INDEX statement. Indexes created on temporary tables are also temporary, and the data in the index has the same session or transaction scope as the data in the temporary table.

You can create views that access both temporary and permanent tables. You can also create triggers on temporary tables.

Oracle utilities can export and import the definition of a temporary table. However, no data rows are exported even if you use the ROWS clause. Similarly, you can replicate the definition of a temporary table, but you cannot replicate its data.

Segment Allocation

Temporary tables use temporary segments. Unlike permanent tables, temporary tables and their indexes do not automatically allocate a segment when they are created. Instead, segments are allocated when the first INSERT (or CREATE TABLE AS SELECT) is performed. This means that if a SELECT, UPDATE, or DELETE is performed before the first INSERT, then the table appears to be empty.

You can perform DDL statements (ALTER TABLE, DROP TABLE, CREATE INDEX, and so on) on a temporary table only when no session is currently bound to it. A session gets bound to a temporary table when an INSERT is performed on it. The session gets unbound by a TRUNCATE, at session termination, or by doing a COMMIT or ROLLBACK for a transaction-specific temporary table.

Temporary segments are deallocated at the end of the transaction for transaction-specific temporary tables and at the end of the session for session-specific temporary tables.

Parent and Child Transactions

Transaction-specific temporary tables are accessible by user transactions and their child transactions. However, a given transaction-specific temporary table cannot be used concurrently by two transactions in the same session, although it can be used by transactions in different sessions.

If a user transaction does an INSERT into the temporary table, then none of its child transactions can use the temporary table afterward.

If a child transaction does an INSERT into the temporary table, then at the end of the child transaction, the data associated with the temporary table goes away. After that, either the user transaction or any other child transaction can access the temporary table.