### Date / Time Arithmetic with Oracle

If you store date and time information in Oracle, you have two different options for the column's datatype - DATE and TIMESTAMP.

**DATE** is the datatype that we are all familiar with when we think about representing date and time values. It has the ability to store the month, day, year, century, hours, minutes, and seconds. It is typically good for representing data for when something has happened or should happen in the future. **The problem with the DATE datatype is its' granularity** when trying to determine a time interval between two events when the events happen within a second of each other. This issue is solved with the **TIMESTAMP** datatype.

In order to represent the date stored in a more readable format, the **TO_CHAR** function has traditionally been wrapped around the date:

```sql
SELECT TO_CHAR(hiredate,'DD.MM.YYYY:HH24:MI:SS') "hiredate"
FROM emp;
```

<table>
<thead>
<tr>
<th>hiredate</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.12.1980:00:00</td>
</tr>
<tr>
<td>20.02.1981:00:00</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

14 rows selected.
**Working with Dates**

- Oracle stores dates in an internal numeric format representing the century, year, month, day, hours, minutes, seconds.
- The default date format is DD-MON-YY.
- SYSDATE is a function returning date and time.
- DUAL is a dummy table used to view SYSDATE.

**Oracle Date Format**

The default display and input format for any date is DD-MON-YY. Valid Oracle dates are between January 1, 4712 B.C., and December 31, 9994 A.D.

**SYSDATE**

SYSDATE is a date function that returns the current date and time. You can use SYSDATE just as you would use any other column name. For example, you can display the current date by selecting SYSDATE from a table. It is customary to select SYSDATE from a dummy table called DUAL.

**DUAL**

The DUAL table is owned by the user SYS and can be accessed by users. It contains one column, DUMMY, and one row with the value X. The DUAL table is useful when you want to return a value once only — for instance, the value of a constant, pseudocolumn, or expression that is not derived from a table with user data.

**Example**

```
SELECT sysdate
FROM dual;
```

<table>
<thead>
<tr>
<th>SYSDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>18/03/2007</td>
</tr>
</tbody>
</table>
## Date Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_MONTHS</td>
<td>Adds the specified number of months to a date.</td>
</tr>
<tr>
<td>LAST_DAY</td>
<td>Returns the last day in the month of the specified date.</td>
</tr>
<tr>
<td>MONTHS_BETWEEN</td>
<td>Calculates the number of months between two dates.</td>
</tr>
<tr>
<td>NEW_TIME</td>
<td>Returns the date/time value, with the time shifted as requested by the specified time zones.</td>
</tr>
<tr>
<td>NEXT_DAY</td>
<td>Returns the date of the first weekday specified that is later than the date.</td>
</tr>
<tr>
<td>ROUND</td>
<td>Returns the date rounded by the specified format unit.</td>
</tr>
<tr>
<td>SYSDATE</td>
<td>Returns the current date and time in the Oracle Server.</td>
</tr>
<tr>
<td>TRUNC</td>
<td>Truncates the specified date of its time portion according to the format unit provided.</td>
</tr>
</tbody>
</table>
Arithmetic with Dates

• Add or subtract a number to or from a date for a resultant *date* value,
• Subtract two dates to find the *number of days* between those dates.
• Add *hours* to a date by dividing the number of hours by 24.

**Arithmetic with Dates**

Since the database stores dates as numbers, you can perform calculations using arithmetic operators such as addition and subtraction. You can add and subtract number constants as well as dates.

You can perform the following operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date + number</td>
<td>Date</td>
<td>Adds a number of days to a date</td>
</tr>
<tr>
<td>Date - number</td>
<td>Date</td>
<td>Subtracts a number of days from a date</td>
</tr>
<tr>
<td>Date – date</td>
<td>Number of days</td>
<td>Subtracts one date from another</td>
</tr>
<tr>
<td>Date + number/24</td>
<td>Date</td>
<td>Adds a number of hours to a date</td>
</tr>
</tbody>
</table>
Addition and Subtraction of Dates

You can add and subtract number constants as well as other dates from dates. Oracle interprets number constants in arithmetic date 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 `HIREDATE` column of the `EMP` table from `SYSDATE` returns the number of days since each employee was hired.

```sql
SELECT '03.12.2004:10:34:24' "Now",
       TO_CHAR(hiredate,'DD.MM.YYYY:HH24:MI:SS') "Hiredate",
       TO_DATE('03.12.2004:10:34:24','DD.MM.YYYY:HH24:MI:SS')
          - hiredate "Hired since [Days]"
FROM emp;
```

<table>
<thead>
<tr>
<th>Now</th>
<th>Hiredate</th>
<th>Hired since [Days]</th>
</tr>
</thead>
<tbody>
<tr>
<td>03.12.2004:10:34:24</td>
<td>17.12.1980:00:00:00</td>
<td>8752,44056</td>
</tr>
<tr>
<td>03.12.2004:10:34:24</td>
<td>20.02.1981:00:00:00</td>
<td>8687,44056</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

14 rows selected.

**Note:**

You cannot multiply or divide `DATE` values. Oracle provides functions for many common date operations.

```sql
SELECT '13.02.2007:10:34:24' "Şimdi",
       TO_DATE('13.02.2007:10:34:24','DD.MM.YYYY:HH24:MI:SS')
FROM dual;
```

| Şimdi                  | TO_DATE('13.02.2007:10:34:24','DD.MM.YYYY:HH24:MI:SS')
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>13.02.2007:10:34:24</td>
<td>23453,0117</td>
</tr>
</tbody>
</table>
Using Arithmetic Operators with Dates

SELECT ename, (SYSDATE - hiredate) / 7  WEEKS
FROM emp
WHERE  deptno = 10;

<table>
<thead>
<tr>
<th>ENAME</th>
<th>WEEKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLARK</td>
<td>1344,86479</td>
</tr>
<tr>
<td>KING</td>
<td>1321,86479</td>
</tr>
<tr>
<td>MILLER</td>
<td>1312,29336</td>
</tr>
</tbody>
</table>

Arithmetic with Dates (continued)

The example on the slide displays the name and the number of weeks employed for all employees in department 10. It subtracts the current date (SYSDATE) from the date on which the employee was hired and divides the result by 7 to calculate the number of weeks that a worker has been employed.

Note: SYSDATE is a SQL function that returns the current date and time. Your results may differ from the example.
Using Date Functions

MONTHS_BETWEEN ('01-SEP-95', '11-JAN-94') 19.6774194

ADD_MONTHS ('11-JAN-94', 6)  '11-JUL-94'

NEXT_DAY ('01-SEP-95', 'FRIDAY')  '08-SEP-95'

LAST_DAY('01-SEP-95')  '30-SEP-95'
Date Functions (continued)

For all employees employed for fewer than 200 months, display the employee number, hiredate, number of months employed, six-month review date, first Friday after hiredate, and last day of the month when hired.

```
SELECT empno, hiredate,
    MONTHS_BETWEEN(SYSDATE, hiredate) TENURE,
    ADD_MONTHS(hiredate, 6) REVIEW,
    NEXT_DAY(hiredate, 'CUMA') CUMA,
    LAST_DAY(hiredate) Giris
FROM emp
WHERE MONTHS_BETWEEN (SYSDATE, hiredate) > 310;
```
ADD_MONTHS

Move ahead date by three months:
\[\text{ADD\_MONTHS} \ ('12\text{-JAN\-1995}', \ 3) \Rightarrow \text{12\text{-APR\-1995}}\]

Specify negative number of months in first position:
\[\text{ADD\_MONTHS} \ (-12, \ '12\text{-MAR\-1990}') \Rightarrow \text{12\text{-MAR\-1989}}\]
ADD_MONTHS

ADD_MONTHS always shifts the date by whole months. You can provide a fractional value for the month_shift parameter, but ADD_MONTHS will always round down to the whole number nearest zero, as shown in these examples:

ADD_MONTHS ('28-FEB-1989', 1.5) same as
ADD_MONTHS ('28-FEB-1989', 1) ==> 31-MAR-1989

ADD_MONTHS ('28-FEB-1989', 1.9999) same as
ADD_MONTHS ('28-FEB-1989', 1) ==> 31-MAR-1989

ADD_MONTHS ('28-FEB-1989', -1.9999) same as
ADD_MONTHS ('28-FEB-1989', -1) ==> 31-JAN-1989

ADD_MONTHS ('28-FEB-1989', .5) same as
ADD_MONTHS ('28-FEB-1989', 0) ==> 28-FEB-1989
The LAST_DAY function

The LAST_DAY function returns the date of the last day of the month for a given date. The specification is:

```sql
FUNCTION LAST_DAY (date_in IN DATE) RETURN DATE
```

This function is useful because the number of days in a month varies throughout the year. With LAST_DAY, for example, you do not have to try to figure out if February of this or that year has 28 or 29 days. Just let LAST_DAY figure it out for you.

Here are some examples of LAST_DAY:

- Go to the last day in the month:
  ```sql
  LAST_DAY ('12-JAN-99') ==> 31-JAN-1999
  ```
- If already on the last day, just stay on that day:
  ```sql
  LAST_DAY ('31-JAN-99') ==> 31-JAN-1999
  ```
- Get the last day of the month three months after being hired:
  ```sql
  LAST_DAY (ADD_MONTHS (hiredate, 3))
  ```
- Tell me the number of days until the end of the month:
  ```sql
  LAST_DAY (SYSDATE) - SYSDATE
  ```
LAST_DAY (date)

LAST_DAY returns the date of the last day of the month that contains date. The return type is always DATE, regardless of the datatype of date.

Example

The following statement determines how many days are left in the current month:

```
SELECT SYSDATE,
       LAST_DAY(SYSDATE) "Last",
       LAST_DAY(SYSDATE) - SYSDATE "Days Left"
FROM DUAL;
```

<table>
<thead>
<tr>
<th>SYSDATE</th>
<th>Last</th>
<th>Days Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>19/03/2007</td>
<td>31/03/2007</td>
<td>12</td>
</tr>
</tbody>
</table>

Get the last date of a month:

```
SELECT LAST_DAY (TO_DATE ('02','MM'))
FROM dual;
```

<table>
<thead>
<tr>
<th>LAST_DAY(T</th>
</tr>
</thead>
<tbody>
<tr>
<td>28/02/2007</td>
</tr>
</tbody>
</table>
**NEXT_DAY** (*date, day*)

NEXT_DAY returns the date of the first weekday named by *day* that is later than *date*. The return type is always DATE, regardless of the datatype of *date*. The argument *day* must be a day of the week in the date language of your session, either the full name or the abbreviation. The minimum number of letters required is the number of letters in the abbreviated version. Any characters immediately following the valid abbreviation are ignored. The return value has the same hours, minutes, and seconds component as the argument *date*.

**Example**

Return the date of the next Monday after now:

```
SELECT TO_CHAR ( NEXT_DAY (sysdate, 'PAZARTESİ' ) ,
                 'DD.MM.YYYY' )

"Next Monday from now"

FROM DUAL;
```

<table>
<thead>
<tr>
<th>Next Monday from now</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.03.2007</td>
</tr>
</tbody>
</table>
MONTHS_BETWEEN function

The MONTHS_BETWEEN function calculates the number of months between two dates and returns that difference as a number. The specification is:

MONTHS_BETWEEN (date1, date2)

It returns a number calculated as the number of months between date1 and date2.

The following rules apply to MONTHS_BETWEEN:

- If date1 comes after date2, then MONTHS_BETWEEN returns a positive number.
- If date1 comes before date2, then MONTHS_BETWEEN returns a negative number.
- If date1 and date2 are in the same month, then MONTHS_BETWEEN returns a fraction (a value between -1 and +1).
- If date1 and date2 both fall on the last day of their respective months, then MONTHS_BETWEEN returns a whole number (no fractional component).
- If date1 and date2 are in different months and at least one of the dates is not a last day in the month, MONTHS_BETWEEN returns a fractional number. The fractional component is calculated on a 31-day month basis and also takes into account any differences in the time component of date1 and date2.
MONTHS_BETWEEN (continued)

Here are some examples of the uses of MONTHS_BETWEEN:

- Calculate two ends of month, the first earlier than the second:
  
  \[
  \text{MONTHS\_BETWEEN} \quad ('31\[JAN\[1994', \quad '28\[FEB\[1994')
  \]
  
  \[=> \quad -1\]

- Calculate two ends of month, the first later than the second:
  
  \[
  \text{MONTHS\_BETWEEN} \quad ('31\[MAR\[1995', \quad '28\[FEB\[1994')
  \]
  
  \[=> \quad 13\]

- Calculate when both dates fall in the same month:
  
  \[
  \text{MONTHS\_BETWEEN} \quad ('28\[FEB\[1994', \quad '15\[FEB\[1994')
  \]
  
  \[=> \quad 0\]

- Perform months_between calculations with a fractional component:
  
  - \[
  \text{MONTHS\_BETWEEN} \quad ('31\[JAN\[1994', \quad '1\[MAR\[1994')
  \]
  
  \[=> \quad -1.0322581\]

  - \[
  \text{MONTHS\_BETWEEN} \quad ('31\[JAN\[1994', \quad '2\[MAR\[1994')
  \]
  
  \[=> \quad -1.0645161\]

  - \[
  \text{MONTHS\_BETWEEN} \quad ('31\[JAN\[1994', \quad '10\[MAR\[1994')
  \]
  
  \[=> \quad -1.3225806\]

If you detect a pattern here you are right. As I said, MONTHS_BETWEEN calculates the fractional component of the number of months by assuming that each month has 31 days. Therefore, each additional day over a complete month counts for 1/31 of a month, and:

1 divided by 31 = .032258065—more or less!

According to this rule, the number of months between January 31, 1994 and February 28, 1994 is one -- a nice, clean integer. But to calculate the number of months between January 31, 1994 and March 1, 1994, I have to add an additional .032258065 to the difference (and make that additional number negative because in this case MONTHS_BETWEEN counts from the first date back to the second date.
The ROUND function

The ROUND function rounds a date value to the nearest date as specified by a format mask. It is just like the standard numeric ROUND function, which rounds a number to the nearest number of specified precision, except that it works with dates. The specification for ROUND is as follows:

```
ROUND (date [, format_mask VARCHAR2])
```

It returns a date.

The ROUND function always rounds the time component of a date to midnight (12:00 A.M.). The format mask is optional. If you do not include a format mask, ROUND rounds the date to the nearest day. In other words, it checks the time component of the date. If the time is past noon, then ROUND returns the next day with a time component of midnight.

Examples

Round up to the next century:

```
TO_CHAR (ROUND (TO_DATE ('01-MAR-1994'), 'CC'),
'DD-MON-YYYY')
01-JAN-2000
```

Round back to the beginning of the current century:

```
TO_CHAR (ROUND (TO_DATE ('01-MAR-1945'), 'CC'),
'DD-MON-YYYY')
01-JAN-1900
```

Round down and up to the first of the year:

```
ROUND (TO_DATE ('01-MAR-1994'), 'YYYY')
⇒ 01-JAN-1994
```

```
ROUND (TO_DATE ('01-SEP-1994'), 'YEAR')
⇒ 01-JAN-1995
```

Round up and down to the quarter (first date in the quarter):

```
ROUND (TO_DATE ('01-MAR-1994'), 'Q')
⇒ 01-APR-1994
```
ROUND (TO_DATE ('15-APR-1994'), 'Q')
\[=\] 01-APR-1994

Round down and up to the first of the month:

ROUND (TO_DATE ('12-MAR-1994'), 'MONTH')
\[=\] 01-MAR-1994

ROUND (TO_DATE ('17-MAR-1994'), 'MM')
\[=\] 01-APR-1994

Day of first of year is Saturday:

TO_CHAR (TO_DATE ('01-JAN-1994'), 'DAY')
\[=\] 'SATURDAY'

So round to date of nearest Saturday for `01-MAR-1994':

ROUND (TO_DATE ('01-MAR-1994'), 'WW')
\[=\] 26-FEB-1994

First day in the month is a Friday:

TO_CHAR (TO_DATE ('01-APR-1994'), 'DAY')
\[=\] FRIDAY

So round to date of nearest Friday from April 16, 1994:

TO_CHAR ('16-APR-1994'), 'DAY')
\[=\] SATURDAY
\[=\] 
ROUND (TO_DATE ('16-APR-1994'), 'W')
\[=\] 15-APR-1994
\[=\]
TO_CHAR (ROUND (TO_DATE ('16-APR-1994'), 'W'), 'DAY')
\[=\] FRIDAY
In the rest of the examples I use TO_DATE in order to pass a time component to the ROUND function, and TO_CHAR to display the new time.

Round back to nearest day (time always midnight):

```
TO_CHAR (ROUND (TO_DATE ('11-SEP-1994 10:00 AM',
                     'DD-MON-YY HH:MI AM'), 'DD'),
        'DD-MON-YY HH:MI AM')
11-SEP-1994 12:00 AM
```

Round forward to the nearest day:

```
TO_CHAR (ROUND (TO_DATE ('11-SEP-1994 4:00 PM',
                        'DD-MON-YY HH:MI AM'), 'DD'),
        'DD-MON-YY HH:MI AM')
12-SEP-1994 12:00 AM
```

Round back to the nearest hour:

```
TO_CHAR (ROUND (TO_DATE ('11-SEP-1994 4:17 PM',
                        'DD-MON-YY HH:MI AM'), 'HH'),
        'DD-MON-YY HH:MI AM')
==> 11-SEP-1994 04:00 PM
```
The TRUNC function

The TRUNC function truncates date values according to the specified format mask. The specification for TRUNC is:

\[
\text{TRUNC (date [, format_mask VARCHAR2])}
\]

It returns a date.

The TRUNC date function is similar to the numeric FLOOR function.

Here are some examples of TRUNC for dates (all assuming a default date format mask of DD-MON-YYYY):

Without a format mask, TRUNC sets the time to 12:00 A.M. of the same day:

\[
\text{TO_CHAR (TRUNC (TO_DATE ('11-SEP-1994 9:36 AM', 'DD-MON-YYYY HH:MI AM')))}
\]

\[
11-SEP-1994 12:00 AM
\]

Trunc to the beginning of the century in all cases:

\[
\text{TO_CHAR (TRUNC (TO_DATE ('01-MAR-1994'), 'CC'), 'DD-MON-YYYY')}
\]

\[
===> 01-JAN-1900
\]

\[
\text{TO_CHAR (TRUNC (TO_DATE ('01-MAR-1945'), 'CC'), 'DD-MON-YYYY')}
\]

\[
01-JAN-1900
\]

Trunc to the first of the current year:

\[
\text{TRUNC (TO_DATE ('01-MAR-1994'), 'YYYY')}\Rightarrow 01-JAN-1994
\]

\[
\text{TRUNC (TO_DATE ('01-SEP-1994'), 'YEAR')}\Rightarrow 01-JAN-1994
\]

Trunc to the first day of the quarter:

\[
\text{TRUNC (TO_DATE ('01-MAR-1994'), 'Q')}\Rightarrow 01-JAN-1994
\]
TRUNC (TO_DATE ('15-APR-1994'), 'Q')
=> 01-APR-1994

Trunc to the first of the month:
TRUNC (TO_DATE ('12-MAR-1994'), 'MONTH')
⇒ 01-MAR-1994

TRUNC (TO_DATE ('17-MAR-1994'), 'MM')
=> 01-APR-1994

In the rest of the examples I use TO_DATE to pass a time component to the TRUNC function, and TO_CHAR to display the new time:

Trunc back to the beginning of the current day (time is always midnight):
TO_CHAR (TRUNC (TO_DATE ('11-SEP-1994 10:00 AM', 'DD-MON-YYYY HH:MI AM'), 'DD'), 'DD-MON-YYYY HH:MI AM')

=> 11-SEP-1994 12:00 AM

TO_CHAR (TRUNC (TO_DATE ('11-SEP-1994 4:00 PM', 'DD-MON-YYYY HH:MI AM'), 'DD'), 'DD-MON-YYYY HH:MI AM')
11-SEP-1994 12:00 AM

Trunc to the beginning of the current hour:
TO_CHAR (TRUNC (TO_DATE ('11-SEP-1994 4:17 PM', 'DD-MON-YYYY HH:MI AM'), 'HH'), 'DD-MON-YYYY HH:MI AM')
11-SEP-1994 04:00 PM
New_Time Function

In Oracle/PLSQL, the new_time function returns a date in time zone1 to a date in time zone2.

The syntax for the new_time function is:

\[ \text{new_time( date, zone1, zone2 )} \]

zone1 and zone2 can be any of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST</td>
<td>Atlantic Standard Time</td>
</tr>
<tr>
<td>ADT</td>
<td>Atlantic Daylight Time</td>
</tr>
<tr>
<td>BST</td>
<td>Bering Standard Time</td>
</tr>
<tr>
<td>BDT</td>
<td>Bering Daylight Time</td>
</tr>
<tr>
<td>CST</td>
<td>Central Standard Time</td>
</tr>
<tr>
<td>CDT</td>
<td>Central Daylight Time</td>
</tr>
<tr>
<td>EST</td>
<td>Eastern Standard Time</td>
</tr>
<tr>
<td>EDT</td>
<td>Eastern Daylight Time</td>
</tr>
<tr>
<td>GMT</td>
<td>Greenwich Mean Time</td>
</tr>
<tr>
<td>HST</td>
<td>Alaska-Hawaii Standard Time</td>
</tr>
<tr>
<td>HDT</td>
<td>Alaska-Hawaii Daylight Time</td>
</tr>
<tr>
<td>MST</td>
<td>Mountain Standard Time</td>
</tr>
<tr>
<td>MDT</td>
<td>Mountain Daylight Time</td>
</tr>
<tr>
<td>NST</td>
<td>Newfoundland Standard Time</td>
</tr>
<tr>
<td>PST</td>
<td>Pacific Standard Time</td>
</tr>
<tr>
<td>PDT</td>
<td>Pacific Daylight Time</td>
</tr>
<tr>
<td>YST</td>
<td>Yukon Standard Time</td>
</tr>
<tr>
<td>YDT</td>
<td>Yukon Daylight Time</td>
</tr>
</tbody>
</table>
NEW_TIME FUNCTION

SELECT new_time( '17-03-2007' , 'GMT ' , 'EST ' )
FROM dual;

<table>
<thead>
<tr>
<th>NEW_TIME('</th>
</tr>
</thead>
<tbody>
<tr>
<td>16/03/2007</td>
</tr>
</tbody>
</table>

SELECT new_time (to_date ('2003/11/01 01:45','yyyy/mm/dd HH24:MI'), 'AST', 'MST')
FROM dual;

<table>
<thead>
<tr>
<th>NEW_TIME(T</th>
</tr>
</thead>
<tbody>
<tr>
<td>31/10/2003</td>
</tr>
</tbody>
</table>
