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:

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

hiredate
17.12.1980:00:00:00
20.02.1981:00:00:00
...

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 behween Januar 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;
```

SYSDATE
18/03/2007

Date Functions

The Built-In Date Functions	
Name	**Description**
ADD_MONTHS	Adds the specified number of months to a date.
LAST_DAY	Returns the last day in the month of the specified date.
MONTHS_ BETWEEN	Calculates the number of months between two dates.
NEW_TIME	Returns the date/time value, with the time shifted as requested by the specified time zones.
NEXT_DAY	Returns the date of the first weekday specified that is later than the date.
ROUND	Returns the date rounded by the specified format unit.
SYSDATE	Returns the current date and time in the Oracle Server.
TRUNC	Truncates the specified date of its time portion according to the format unit provided.

Arithmetic with Dates

- **Add or subtract a number to or from a date for a resultant *date* value,**
- **Subtract two dates to find the *numberof* 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:

Operation	Result	Description
Date + number	Date	Adds a number of days to a date
Date - number	Date	Subtracts a number of days from a date
Date – date	Number of days	Subracts one date from another
Date + number/24	Date	Adds a number of hours to a date

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.

```
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;
```

Now	Hiredate	Hired since [Days]
03.12.2004:10:34:24	17.12.1980:00:00:00	8752,44056
03.12.2004:10:34:24	20.02.1981:00:00:00	8687,44056
...		

14 rows selected.

Note:

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

```
SELECT '13.02.2007:10:34:24' "Şimdi",
       TO_DATE('13.02.2007:10:34:24','DD.MM.YYYY:HH24:MI:SS')
- TO_DATE( '28/11/1942:10:17:36' , 'DD/MM/YYYY:HH24:MI:SS' )
FROM dual;
```

Şimdi	TO_DATE('13.02.2007:10:34:24','DD.MM.YYYY:HH24:MI:SS')- TO_DATE('28/11/1942:10:17:36','DD/MM/YYYY:HH24:MI:SS')
13.02.2007:10:34:24	23453,0117

Using Arithmetic Operators with Dates

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

ENAME	WEEKS
CLARK	1344,86479
KING	1321,86479
MILLER	1312,29336

Arithmetic with Dates (continued)

The example on the slide displays the name and the number of wecks 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) Giriş
FROM emp
WHERE MONTHS_BETWEEN (SYSDATE, hiredate) > 310;
```

EMPNO	HIREDATE	TENURE	REVIEW	CUMA	GIRIŞ
7369	17/12/1980	315,065473	17/06/1981	19/12/1980	31/12/1980
7499	20/02/1981	312,968698	20/08/1981	27/02/1981	28/02/1981
7521	22/02/1981	312,904182	22/08/1981	27/02/1981	28/02/1981
7566	02/04/1981	311,549344	02/10/1981	03/04/1981	30/04/1981
7698	01/05/1981	310,581602	01/11/1981	08/05/1981	31/05/1981

ADD_MONTHS

Move ahead date by three months:

```
ADD_MONTHS ('12-JAN-1995', 3) ==> 12-APR-1995
```

Specify negative number of months in first position:

```
ADD_MONTHS (-12, '12-MAR-1990') ==> 12-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:

```
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:

  ```
  LAST_DAY ('12-JAN-99') ==> 31-JAN-1999
  ```

- If already on the last day, just stay on that day:

  ```
  LAST_DAY ('31-JAN-99') ==> 31-JAN-1999
  ```

- Get the last day of the month three months after being hired:

  ```
  LAST_DAY (ADD_MONTHS (hiredate, 3))
  ```

- Tell me the number of days until the end of the month:

  ```
  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;
```

SYSDATE	Last	Days Left
19/03/2007	31/03/2007	12

Get the last date of a month:

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

LAST_DAY(T
28/02/2007

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;
```

Next Monday from now
19.03.2007

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:

```
MONTHS_BETWEEN   ('31-JAN-1994',   '28-FEB-1994')
==> -1
```

- Calculate two ends of month, the first later than the second:

```
MONTHS_BETWEEN   ('31-MAR-1995',   '28-FEB-1994')
==> 13
```

- Calculate when both dates fall in the same month:

```
MONTHS_BETWEEN   ('28-FEB-1994',   '15-FEB-1994')
==>  0
```

- Perform months_between calculations with a fractional component:
-
```
 MONTHS_BETWEEN   ('31-JAN-1994',   '1-MAR-1994')
==> -1.0322581
```

-
```
 MONTHS_BETWEEN   ('31-JAN-1994',   '2-MAR-1994')
==> -1.0645161
MONTHS_BETWEEN   ('31-JAN-1994',   '10-MAR-1994')
==> -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:

```
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:

```
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:

```
TO_CHAR (TRUNC (TO_DATE ('01-MAR-1994'), 'CC'), 'DD-
MON-YYYY')
==> 01-JAN-1900

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

Trunc to the first of the current year:

```
TRUNC (TO_DATE ('01-MAR-1994'), 'YYYY')
  ⇨  01-JAN-1994

TRUNC (TO_DATE ('01-SEP-1994'), 'YEAR')
==> 01-JAN-1994
```

Trunc to the first day of the quarter:

```
TRUNC (TO_DATE ('01-MAR-1994'), 'Q')
  ⇨  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:

```
new_time( date, zone1, zone2 )
```

zone1 and *zone2* can be any of the following values:

Value	Description
AST	Atlantic Standard Time
ADT	Atlantic Daylight Time
BST	Bering Standard Time
BDT	Bering Daylight Time
CST	Central Standard Time
CDT	Central Daylight Time
EST	Eastern Standard Time
EDT	Eastern Daylight Time
GMT	Greenwich Mean Time
HST	Alaska-Hawaii Standard Time
HDT	Alaska-Hawaii Daylight Time
MST	Mountain Standard Time
MDT	Mountain Daylight Time
NST	Newfoundland Standard Time
PST	Pacific Standard Time
PDT	Pacific Daylight Time
YST	Yukon Standard Time
YDT	Yukon Daylight Time

NEW_TIME FUNCTION

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

NEW_TIME('
16/03/2007

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

NEW_TIME(T
31/10/2003