Dates in Oracle

8 min. read · View original

When representing date-time information in Oracle it becomes absolutely critical to know with which version of the Oracle server you are working. From version 9 onwards it is possible to represent points in time and time intervals using ANSI SQL data types such as `timestamp` and `interval`. Earlier versions of Oracle represented points in time with the `date` datatype, which is precise to within one second, and time intervals as numbers (where 1 = one day).

We strongly recommend that you use the newly available ANSI data types when building new applications. These are cleaner and more powerful than the older Oracle-specific way of doing things and further will make it easier to port your application to another RDBMS if necessary.

If you are stuck using an older version of Oracle or are writing queries and transactions to an older data model, please visit http://philip.greenspun.com/sql/dates-pre-9.

Querying by Date

Suppose that we have the following table to record user registrations:

```
create table users (
        user_id                 integer primary
key,
        first_names             varchar(50),
        last_name               varchar(50) not
null,
        email                   varchar(100) not
null unique,
        -- we encrypt passwords using operating
system crypt function
        password                varchar(30) not
null,
        -- we only need precision to within one
second
        registration_date       timestamp(0)
);

-- add some sample data
insert into users
(user_id, first_names, last_name, email,
password, registration_date)
values
(1,'schlomo','mendelowitz','schlomo@mendelowitz.com','67xui2',
to_timestamp('2003-06-13 09:15:00','YYYY-MM-DD
HH24:MI:SS'));

insert into users
(user_id, first_names, last_name, email,
password, registration_date)
values
(2,'George Herbert Walker','Bush','former-
president@whitehouse.gov','kl88q',
to_timestamp('2003-06-13 15:18:22','YYYY-MM-DD
HH24:MI:SS'));
```

Let's query for people who registered during the last day:

```
column email format a35
column registration_date format a25

select email, registration_date
from users
where registration_date > current_date - interval
'1' day;

EMAIL
REGISTRATION_DATE
---------------------------------- -------------
------------
schlomo@mendelowitz.com             13-JUN-03
09.15.00 AM
former-president@whitehouse.gov     13-JUN-03
03.18.22 PM
```

Note how the registration date comes out in a non-standard format that won't sort lexicographically and that does not have a full four digits for the year. You should curse your database administrator at this point for not configuring Oracle with a more sensible default. You can fix the problem for yourself right now, however:

```
alter session
set nls_timestamp_format =
'YYYY-MM-DD HH24:MI:SS';

select email, registration_date
from users
where registration_date > current_date - interval
'1' day;

EMAIL
REGISTRATION_DATE
---------------------------------- -------------
---------
schlomo@mendelowitz.com             2003-06-13
09:15:00
former-president@whitehouse.gov     2003-06-13
15:18:22
```

You can query for shorter time intervals:

```
select email, registration_date
from users
where registration_date > current_date - interval
'1' hour;

EMAIL
REGISTRATION_DATE
---------------------------------- -------------
------------
former-president@whitehouse.gov     2003-06-13
15:18:22

select email, registration_date
from users
where registration_date > current_date - interval
'1' minute;

no rows selected

select email, registration_date
from users
where registration_date > current_date - interval
'1' second;

no rows selected
```

You can be explicit about how you'd like the timestamps formatted:

```
select email, to_char(registration_date,'Day,
Month DD, YYYY') as reg_day
from users
order by registration_date;

EMAIL                              REG_DAY
---------------------------------- -------------
----------------
schlomo@mendelowitz.com            Friday   ,
June      13, 2003
```

```
former-president@whitehouse.gov     Friday   ,
June      13, 2003
```

Oops. Oracle pads some of these fields by default so that reports will be lined up and neat. We'll have to trim the strings ourselves:

```
select
  email,
  trim(to_char(registration_date,'Day')) || ', '
||
   trim(to_char(registration_date,'Month')) || '
' ||
   trim(to_char(registration_date,'DD, YYYY')) as
reg_day
from users
order by registration_date;

EMAIL                               REG_DAY
----------------------------------- -------------
---------------
schlomo@mendelowitz.com             Friday, June
13, 2003
former-president@whitehouse.gov     Friday, June
13, 2003
```

Some Very Weird Things

One reason that Oracle may have resisted ANSI date-time datatypes and arithmetic is that they can make life very strange for the programmer.

```
alter session set nls_date_format = 'YYYY-MM-DD';

-- old
select add_months(to_date('2003-07-31','YYYY-MM-
DD'),-1) from dual;

ADD_MONTHS
----------
2003-06-30
```

```
-- new
select to_timestamp('2003-07-31','YYYY-MM-DD') -
interval '1' month from dual;

ERROR at line 1:
ORA-01839: date not valid for month specified

-- old
select to_date('2003-07-31','YYYY-MM-DD') - 100
from dual;

TO_DATE('2
----------
2003-04-22

-- new (broken)
select to_timestamp('2003-07-31','YYYY-MM-DD') -
interval '100' day from dual;

ERROR at line 1:
ORA-01873: the leading precision of the interval
is too small

-- new (note the extra "(3)")
select to_timestamp('2003-07-31','YYYY-MM-DD') -
interval '100' day(3) from dual;

TO_TIMESTAMP('2003-07-31','YYYY-MM-DD')-
INTERVAL'100'DAY(3)
---------------------------------------------------
------------
2003-04-22 00:00:00
```

Some Profoundly Painful Things

Calculating time intervals between rows in a table can be very painful because there is no way in standard SQL to refer to "the value of this column from the previous row in the report". You can do this easily enough in an imperative computer language, e.g., C#, Java, or Visual Basic, that is reading rows from an SQL database but doing it purely in SQL is tough.

Let's add a few more rows to our users table to see how this works.

```
insert into users
(user_id, first_names, last_name, email,
password, registration_date)
values
(3,'Osama','bin Laden','50kids@aol.com','dieusa',
to_timestamp('2003-06-13 17:56:03','YYYY-MM-DD
HH24:MI:SS'));

insert into users
(user_id, first_names, last_name, email,
password, registration_date)
values
(4,'Saddam','Hussein','livinlarge@saudi-
online.net','wmd34',
to_timestamp('2003-06-13 19:12:43','YYYY-MM-DD
HH24:MI:SS'));
```

Suppose that we're interested in the average length of time between registrations. With so few rows we could just query all the data out and eyeball it:

```
select registration_date
from users
order by registration_date;

REGISTRATION_DATE
-------------------------
2003-06-13 09:15:00
2003-06-13 15:18:22
2003-06-13 17:56:03
2003-06-13 19:12:43
```

If we have a lot of data, however, we'll need to do a self-join.

```
column r1 format a21
column r2 format a21
```

```
select
  u1.registration_date as r1,
  u2.registration_date as r2
from users u1, users u2
where u2.user_id = (select min(user_id) from
users
                    where registration_date >
u1.registration_date)
order by r1;

R1                   R2
-------------------- --------------------
2003-06-13 09:15:00   2003-06-13 15:18:22
2003-06-13 15:18:22   2003-06-13 17:56:03
2003-06-13 17:56:03   2003-06-13 19:12:43
```

Notice that to find the "next row" for the pairing we are using the `user_id` column, which we know to be sequential and unique, rather than the registration_date column, which may not be unique because two users could register at exactly the same time.

Now that we have information from adjacent rows paired up in the same report we can begin to calculate intervals:

```
column reg_gap format a21

select
  u1.registration_date as r1,
  u2.registration_date as r2,
  u2.registration_date-u1.registration_date as
reg_gap
from users u1, users u2
where u2.user_id = (select min(user_id) from
users
                    where registration_date >
u1.registration_date)
order by r1;

R1                   R2
```

```
REG_GAP
-------------------- -------------------- -----
---------------
2003-06-13 09:15:00   2003-06-13 15:18:22
+000000000 06:03:22
2003-06-13 15:18:22   2003-06-13 17:56:03
+000000000 02:37:41
2003-06-13 17:56:03   2003-06-13 19:12:43
+000000000 01:16:40
```

The interval for each row of the report has come back as days, hours, minutes, and seconds. At this point you'd expect to be able to average the intervals:

```
select avg(reg_gap)
from
(select
  u1.registration_date as r1,
  u2.registration_date as r2,
  u2.registration_date-u1.registration_date as
reg_gap
from users u1, users u2
where u2.user_id = (select min(user_id) from
users
                    where registration_date >
u1.registration_date))

ERROR at line 1:
ORA-00932: inconsistent datatypes: expected
NUMBER got INTERVAL
```

Oops. Oracle isn't smart enough to aggregate time intervals. And sadly there doesn't seem to be an easy way to turn a time interval into a number of seconds, for example, that would be amenable to averaging. If you figure how out to do it, please let me know!

Should we give up? If you have a strong stomach you can convert the timestamps to old-style Oracle dates through character strings before

creating the intervals. This will give us a result as a fraction of a day:

```
select avg(reg_gap)
from
(select
  u1.registration_date as r1,
  u2.registration_date as r2,
  to_date(to_char(u2.registration_date,'YYYY-MM-
DD HH24:MI:SS'),'YYYY-MM-DD HH24:MI:SS')
   - to_date(to_char(u1.registration_date,'YYYY-
MM-DD HH24:MI:SS'),'YYYY-MM-DD HH24:MI:SS')
   as reg_gap
from users u1, users u2
where u2.user_id = (select min(user_id) from
users
                    where registration_date >
u1.registration_date))

AVG(REG_GAP)
------------
   .13836034
```

If we're going to continue using this ugly query we ought to create a view:

```
create view registration_intervals
as
select
  u1.registration_date as r1,
  u2.registration_date as r2,
  to_date(to_char(u2.registration_date,'YYYY-MM-
DD HH24:MI:SS'),'YYYY-MM-DD HH24:MI:SS')
   - to_date(to_char(u1.registration_date,'YYYY-
MM-DD HH24:MI:SS'),'YYYY-MM-DD HH24:MI:SS')
   as reg_gap
from users u1, users u2
where u2.user_id = (select min(user_id) from
users
```

```
                    where registration_date >
u1.registration_date)
```

Now we can calculate the average time interval in minutes:

```
select 24*60*avg(reg_gap) as avg_gap_minutes from
registration_intervals;

AVG_GAP_MINUTES
---------------
     199.238889
```

Reporting

Here's an example of using the `to_char` function an GROUP BY to generate a report of sales by calendar quarter:

```
select to_char(shipped_date,'YYYY') as
shipped_year,
       to_char(shipped_date,'Q') as
shipped_quarter,
       sum(price_charged) as revenue
from sh_orders_reportable
where product_id = 143
and shipped_date is not null
group by to_char(shipped_date,'YYYY'),
to_char(shipped_date,'Q')
order by to_char(shipped_date,'YYYY'),
to_char(shipped_date,'Q');

SHIPPED_YEAR         SHIPPED_QUARTER
REVENUE
-------------------- -------------------- -------
---
1998                 2
1280
1998                 3
1150
1998                 4
350
1999                 1
210
```

This is a hint that Oracle has all kinds of fancy date formats (covered in their online documentation). We're using the "Q" mask to get the calendar quarter. We can see that this product started shipping in Q2 1998 and that revenues trailed off in Q4 1998.

More

Next: limits

---

philg@mit.edu

Reader's Comments

> You state: >> no way in standard SQL to refer to "the value of this column from the previous row in the report".
>
> At least in Oracle 8i SQL, there is a way in to refer to this, I'm sure it isn't standard, but useful nonetheless, and so I present it here.
>
> It is called an Analytic Function. There are several, but the one demonstrated in this example is LAST_VALUE.
>
> SELECT r1, r2, r2 - r1 reg_gap FROM (SELECT u1.update_date AS r1, LAST_VALUE (update_date) OVER (ORDER BY update_date ASC ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS r2 FROM users u1 WHERE u1.user_id > 100000) WHERE r1 <> r2 ORDER BY r1
>
> From the inside out, I take the update_date from the users table, and using the LAST_VALUE function, I ask for the last update_date value, including in the window the

current row and the next chronologically ordered row.

I used a higher level query to do the difference simply to avoid repeating the long function, but I could have done it in one.

The results are the same:

"R1" "R2" "REG_GAP" 11/10/2003 5:19:00 PM 11/10/2003 8:23:24 PM 0.128055555555556 11/10/2003 8:23:24 PM 11/12/2003 7:53:10 AM 1.47900462962963 11/12/2003 7:53:10 AM 2/13/2004 3:44:47 PM 93.3275115740741

Although, as I said, I'm using 8i so I don't have the interval type.

To find out more about Analytic Functions, check out the Oracle Documentation SQL Reference.

KSF

-- K SF, September 1, 2004

"Some Profoundly Painful Things -- Calculating time intervals between rows in a table" is very useful, thank you. Some people may need the following technique to establish a sequential numeric identifier. (In the example you assume "user_id column, which we know to be sequential and unique")

declare @tmp (registration_date datetime)

insert @tmp

> select identity(int,1,1) as Sequence, registration_date into #x from users order by registration_date
>
> ... (now use #x instead of users in the example)
>
> drop table #x
>
> -- Steve Davis, January 29, 2006

> You say: "Oops. Oracle pads some of these fields by default so that reports will be lined up and neat. We'll have to trim the strings ourselves." Not quite: one can use FM modifier in format string to instruct Oracle to trim whitespace from resulting string automatically, like this:
>
> ```
> SQL> select to_char(sysdate,'Day, Month DD, YYYY')
> from dual;
>
> TO_CHAR(SYSDATE,'DAY,MONTHDD,
> ----------------------------
> Monday   , May       22, 2006
>
> SQL> select to_char(sysdate,'FMDay, Month DD,
> YYYY') from dual;
>
> TO_CHAR(SYSDATE,'FMDAY,MONTHD
> ----------------------------
> Monday, May 22, 2006
> ```
>
> Note that FM is a switch - second FM in format string negates the effect of the first.
>
> -- Vladimir Zakharychev, May 22, 2006

> Not pretty at all but it works...
>
> ```
> CREATE OR REPLACE FUNCTION interval_to_seconds(x
> INTERVAL DAY TO SECOND ) RETURN NUMBER IS
> s VARCHAR2(26);
> days_s VARCHAR2(26);
> time_s VARCHAR2(26);
> ```

```
N NUMBER(10,6);
BEGIN
  s := TO_CHAR(x);
  days_s := SUBSTR(s,2,INSTR(s,' ')-2);
  time_s := SUBSTR(s,2+LENGTH(days_s)+1);
  N := 86400*TO_NUMBER(days_s) +
3600*TO_NUMBER(SUBSTR(time_s,1,2)) +
60*TO_NUMBER(SUBSTR(time_s,4,2)) +
TO_NUMBER(SUBSTR(time_s,7));
  IF SUBSTR(s,1,1) = '-' THEN
     N := - N;
  END IF;
  RETURN N;
END;
```

-- Andre Mostert, June 20, 2006

1.Find the first monday on every quater based on date ?

Select Next_day(trunc(to_date(sysdate,'DD-MON-YYYY'), 'Q')-1,'Monday') from dual

-- Mohamed Kaleel, April 13, 2007

Computing number of seconds in an interval:

FUNCTION seconds_from_interval(invInterval IN INTERVAL DAY TO SECOND) RETURN NUMBER IS BEGIN

RETURN EXTRACT (DAY FROM invInterval) * 86400 +

EXTRACT (HOUR FROM invInterval) * 3600 +

EXTRACT (MINUTE FROM invInterval) * 60 +

```
EXTRACT (SECOND FROM invInterval);

END seconds_from_interval;
```

-- Bob Jarvis, March 4, 2008

Add a comment