

Utilizing the Power of SQL to Improve Performance

Sep 10, 2001

By: Lev Moltyaner

Problem Definition

Here is a simple and common business problem.

We have 3 source tables, which store data about a person (t1, t2, t3). Each table has a person id (pid). Each table has a start date and end date. The start date is entered by the user, while the end date is derived by a trigger as (start date of next record - 1) or (a high date in the future). Assume this trigger exists and works to produce the source tables as you see them below. The high date is Jun 23, 2004 in my example.

This means that in each table, the timeline is continuous, and there are no gaps or overlaps in dates. However, the earliest record for each person in each table could be on the same date or on different dates.

The primary key of each table is (pid, start date). Each table has one or more mutually exclusive columns, which all need to be in the final table. All three tables also have change by column (cby), to indicate who change the source table.

Objective

Produce a derived table, which merges the data from the 3 tables into 1.

* There should be a record for each period created by a change in any of the 3 tables.

* The table should only contain periods for which there is data in ALL 3 tables. This means that if one source table has no records for a person, the person will not appear in the derived table.

Sample Data

See Appendix A for scripts to create test tables and sample data.

```
select * from t1;
  PID SD          ED          X  CBY
-----
   1 28-SEP-01 08-OCT-01    12 ann
   1 09-OCT-01 23-JUN-04    20 joe
   2 28-SEP-01 03-OCT-01    12 ann
   2 04-OCT-01 23-JUN-04    20 joe
```

```
select * from t2;
  PID SD          ED          Y  CBY
-----
   1 27-SEP-01 03-OCT-01  a frank
   1 04-OCT-01 06-OCT-01  b mike
   1 07-OCT-01 23-JUN-04  c joe
   2 28-SEP-01 03-OCT-01  a frank
   2 04-OCT-01 06-OCT-01  b mike
   2 07-OCT-01 23-JUN-04  c joe
```

```
select * from t3;
  PID SD          ED          Z  CBY
-----
   1 01-OCT-01 23-JUN-04    10 frank
   2 28-SEP-01 03-OCT-01    10 lucy
   2 04-OCT-01 23-JUN-04    30 dan
```

Desired Results

```
select * from d;
  PID SD          ED          X  Y          Z  CBY
-----
   1 01-OCT-01 03-OCT-01    12  a          10 frank
   1 04-OCT-01 06-OCT-01    12  b          10 mike
   1 07-OCT-01 08-OCT-01    12  c          10 joe
   1 09-OCT-01 23-JUN-04    20  c          10 joe
   2 28-SEP-01 03-OCT-01    12  a          10 ann
   2 04-OCT-01 06-OCT-01    20  b          30 joe
   2 07-OCT-01 23-JUN-04    20  c          30 joe
```

Solutions

Solution #1

The easiest solution is a "brute force" method. That is, a pl/sql procedure, which uses a cursor to select from the first table and inside the cursor query the other two tables using current date as part of the input. Although, conceptually this is easy, performance of this approach will be considerably slow.

Solution #2

Another solution is use a number of incremental steps which do inserts and updates. This solution will also be slow.

Solution #3

A third solution is to come up with an algorithm, which minimizes I/O (see Appendix A). In summary it performs a sort-merge join using PL/SQL. It defines a cursor per table and orders the rows by pid and start date. Then the code synchronizes the rows from the three cursors by using the least and greatest functions on dates to control which cursor(s) to fetch next. There are several ways to code the details but the idea is clear.

This solution is easy to understand and is relatively efficient because there are only 3 full table scans. The solution below should be enhanced to include bulk insert, which will improve performance by 30%.

Solution #4

The fourth alternative is a similar algorithm to the above which does a UNION ALL of the three tables so that the data is already sorted by pid and start date across all three tables. This is arguable is slightly simpler algorithm and will yield comparable performance to the above. Keep in mind that although this is only one cursor, it will return 3 records for the same start date for some people.

The important point so far is that both of these solutions are faster to develop, easier to maintain and are much more efficient. However, they demand that we take the time to come up with a structured design. If we jump into development, we will definitely end up with a "brute force" solution.

Solution #5

The next alternative is to solve the entire problem with a simple SQL statement:

```
select t1.pid person_id
      ,greatest(t1.sd,t2.sd,t3.sd) start_date
      ,least(t1.ed,t2.ed,t3.ed) end_date
      ,t1.x
      ,t2.y
      ,t3.z
      ,decode(greatest(t1.sd,t2.sd,t3.sd),t1.sd,t1.cby
              ,t2.sd,t2.cby
              ,t3.sd,t3.cby) changed_by
from t1, t2, t3
where t2.pid = t1.pid
and t3.pid = t1.pid
and greatest(t1.sd, t2.sd, t3.sd) <= least(t1.ed, t2.ed, t3.ed);
```

The last condition can also be re-written as (this is more intuitive):

```
and t1.sd <= t2.ed
and t1.ed >= t2.sd
and t1.sd <= t3.ed
and t1.ed >= t3.sd
and t2.sd <= t3.ed
and t2.ed >= t3.sd
```

Summary

Most developers would not think of the SQL solution because they think programmatically (one row at a time vs in sets). However, once you make the paradigm shift to sets of data, the SQL solution is easier to develop and maintain because it is only a few lines of code.

The SQL solution also performs significantly better than the best PL/SQL solution (3-4 times faster). In general, SQL performs better than PL/SQL (eg. context switches).

Appendix A:

Create Tables Script

```
create table t1 (pid number, sd date, ed date, x number, cby varchar2(10));
create table t2 (pid number, sd date, ed date, y varchar2(1), cby varchar2(10));
create table t3 (pid number, sd date, ed date, z number, cby varchar2(10));
alter table t1 add constraint t1pk primary key (pid,sd);
alter table t2 add constraint t2pk primary key (pid,sd);
alter table t3 add constraint t3pk primary key (pid,sd);
create table d (pid number, sd date, ed date, x number, y varchar2(1), z number, cby varchar2(10));
```

Inserts for Sample Data

```
-- data for pid=1
insert into t1 values (1, trunc(sysdate), trunc(sysdate)+10, 12, 'ann');
insert into t1 values (1, trunc(sysdate)+11, trunc(sysdate)+999, 20, 'joe');
insert into t2 values (1, trunc(sysdate)-1, trunc(sysdate)+5, 'a', 'frank');
insert into t2 values (1, trunc(sysdate)+6, trunc(sysdate)+8, 'b', 'mike');
insert into t2 values (1, trunc(sysdate)+9, trunc(sysdate)+999, 'c', 'joe');
insert into t3 values (1, trunc(sysdate)+3, trunc(sysdate)+999, 10, 'frank');

-- data for pid=2
insert into t1 values (2, trunc(sysdate), trunc(sysdate)+5, 12, 'ann');
insert into t1 values (2, trunc(sysdate)+6, trunc(sysdate)+999, 20, 'joe');

insert into t2 values (2, trunc(sysdate), trunc(sysdate)+5, 'a', 'frank');
insert into t2 values (2, trunc(sysdate)+6, trunc(sysdate)+8, 'b', 'mike');
insert into t2 values (2, trunc(sysdate)+9, trunc(sysdate)+999, 'c', 'joe');
insert into t3 values (2, trunc(sysdate), trunc(sysdate)+5, 10, 'lucy');
insert into t3 values (2, trunc(sysdate)+6, trunc(sysdate)+999, 30, 'dan');
```

Appendix B: Code for Solution #3

```
create or replace procedure t is
lv_commit_frequency integer := 500;
lv_uncommitted_inserts integer := 0;
lv_lst_pid t1.pid%type;
lv_lst_ed t1.ed%type;
lv_grt_sd t1.sd%type;
cursor c1 is
  select *
  from t1
  order by pid, sd;
cursor c2 is
  select *
  from t2
  order by pid, sd;
cursor c3 is
  select *
  from t3
  order by pid, sd;
r1 c1%rowtype;
r2 c2%rowtype;
r3 c3%rowtype;
begin
open c1;
fetch c1 into r1;
open c2;
fetch c2 into r2;
open c3;
fetch c3 into r3;
while c1%found and c2%found and c3%found
loop
  if r1.pid = r2.pid and r1.pid = r3.pid then
    lv_lst_ed := least(r1.ed,r2.ed,r3.ed);
    lv_grt_sd := greatest(r1.sd,r2.sd,r3.sd);
    if lv_lst_ed >= lv_grt_sd then
      insert into t4
        (pid,sd,ed,x,y,z,x_cby,y_cby,z_cby)
      values
        (r1.pid,lv_grt_sd,lv_lst_ed,r1.x,r2.y,r3.z,r1.cby, r2.cby,r3.cby);
      if lv_uncommitted_inserts = lv_commit_frequency then
        commit;
        lv_uncommitted_inserts := 0;
      else
        lv_uncommitted_inserts := lv_uncommitted_inserts + 1;
      end if;
    end if;
    if r1.ed = lv_lst_ed then
      fetch c1 into r1;
    elsif r2.ed = lv_lst_ed then
      fetch c2 into r2;
    else
      fetch c3 into r3;
    end if;
  elsif r1.pid <= r2.pid and r1.pid <= r3.pid then
    fetch c1 into r1;
  elsif r2.pid <= r1.pid and r2.pid <= r3.pid then
    fetch c2 into r2;
  else
    fetch c3 into r3;
  end if;
end loop;
commit;
end;
```