Best way to encapsulate complex Oracle PL/SQL cursor logic as a view?

Asked 16 years, 4 months ago Modified 5 years, 10 months ago Viewed 7k times

4

I've written PL/SQL code to denormalize a table into a much-easer-to-query form. The code uses a temporary table to do some of its work, merging some rows from the original table together.

The logic is written as a pipelined table function, following the pattern from the linked article. The table function uses a `PRAGMA AUTONOMOUS_TRANSACTION` declaration to permit the temporary table manipulation, and also accepts a cursor input parameter to restrict the denormalization to certain ID values.

I then created a view to query the table function, passing in all possible ID values as a cursor (other uses of the function will be more restrictive).

My question: is this all really necessary? Have I completely missed a much more simple way of accomplishing the same thing?

Every time I touch PL/SQL I get the impression that I'm typing way too much.

Update: I'll add a sketch of the table I'm dealing with to give everyone an idea of the denormalization that I'm talking about. The table stores a history of employee jobs, each with an activation row, and (possibly) a termination row. It's possible for an employee to have multiple simultaneous jobs, as well as the same job over and over again in non-contiguous date ranges. For example:

```
| EMP_ID | JOB_ID | STATUS | EFF_DATE    | other columns...
|      1 |     10 | A      | 10-JAN-2008 |
|      2 |     11 | A      | 13-JAN-2008 |
|      1 |     12 | A      | 20-JAN-2008 |
|      2 |     11 | T      | 01-FEB-2008 |
|      1 |     10 | T      | 02-FEB-2008 |
|      2 |     11 | A      | 20-FEB-2008 |
```

Querying that to figure out who is working when in what job is non-trivial. So, my denormalization function populates the temporary table with just the date ranges for each job, for any `EMP_ID`s passed in though the cursor. Passing in `EMP_ID`s 1 and 2 would produce the following:

```
| EMP_ID | JOB_ID | START_DATE  | END_DATE    |
|      1 |     10 | 10-JAN-2008 | 02-FEB-2008 |
|      2 |     11 | 13-JAN-2008 | 01-FEB-2008 |
```

```
|      1 |     12 | 20-JAN-2008 |             |
|      2 |     11 | 20-FEB-2008 |             |
```

(`END_DATE` allows `NULL`s for jobs that don't have a predetermined termination date.)

As you can imagine, this denormalized form is much, much easier to query, but creating it--so far as I can tell--requires a temporary table to store the intermediate results (e.g., job records for which the activation row has been found, but not the termination...yet). Using the pipelined table function to populate the temporary table and then return its rows is the only way I've figured out how to do it.

sql **oracle-database** **plsql**

Share

Improve this question

Follow

edited Sep 18, 2008 at 16:10

asked Aug 21, 2008 at 14:29

yukondude

24.6k ● 13 ● 50 ● 58

6 Answers

Sorted by: Highest score (default)

4

I think a way to approach this is to use analytic functions...

I set up your test case using:

```
create table employee_job (
    emp_id integer,
    job_id integer,
    status varchar2(1 char),
    eff_date date
    );

insert into employee_job values (1,10,'A',to_date('10-JAN-2008','DD-MON-
YYYY'));
insert into employee_job values (2,11,'A',to_date('13-JAN-2008','DD-MON-
YYYY'));
insert into employee_job values (1,12,'A',to_date('20-JAN-2008','DD-MON-
YYYY'));
insert into employee_job values (2,11,'T',to_date('01-FEB-2008','DD-MON-
YYYY'));
insert into employee_job values (1,10,'T',to_date('02-FEB-2008','DD-MON-
YYYY'));
insert into employee_job values (2,11,'A',to_date('20-FEB-2008','DD-MON-
YYYY'));

commit;
```

I've used the **lead** function to get the next date and then wrapped it all as a sub-query just to get the "A" records and add the end date if there is one.

```
select
    emp_id,
    job_id,
    eff_date start_date,
    decode(next_status,'T',next_eff_date,null) end_date
from
    (
    select
        emp_id,
        job_id,
        eff_date,
        status,
        lead(eff_date,1,null) over (partition by emp_id, job_id order by
eff_date, status) next_eff_date,
        lead(status,1,null) over (partition by emp_id, job_id order by
eff_date, status) next_status
    from
        employee_job
    )
where
    status = 'A'
order by
    start_date,
    emp_id,
    job_id
```

I'm sure there's some use cases I've missed but you get the idea. Analytic functions are your friend :)

```
EMP_ID   JOB_ID     START_DATE     END_DATE
  1        10       10-JAN-2008    02-FEB-2008
  2        11       13-JAN-2008    01-FEB-2008
  2        11       20-FEB-2008
  1        12       20-JAN-2008
```

Share Improve this answer Follow

answered Sep 18, 2008 at 21:19

Nick Pierpoint

17.8k ● 9 ● 47 ● 74

Very cool. I'd never heard of analytic functions but I'll definitely check them out. This looks much more simple that what I had tried. – yukondude Sep 18, 2008 at 22:01

It turns out that this will work just fine. There was a complication that it's possible to have multiple activations for a single termination, but by changing the decode line to: decode(next_status, 'A', next_eff_date - 1, next_eff_date) the problem was dealt with beautifully. Thanks a bunch. – yukondude Sep 30, 2008 at 21:36

1

Rather than having the input parameter as a cursor, I would have a table variable (don't know if Oracle has such a thing I'm a TSQL guy) or populate another temp table with the ID values and join on it in the view/function or wherever you need to.

The only time for cursors in my honest opinion is when you *have* to loop. And when you have to loop I always recommend to do that outside of the database in the application logic.

Share Improve this answer Follow

answered Aug 22, 2008 at 10:25

hollystyles

5,029 ● 2 ● 37 ● 38

1

It sounds like you are giving away some read consistency here ie: it will be possible for the contents of your temporary table to be out of sync with the source data, if you have concurrent modification data modification.

Without knowing the requirements, nor complexity of what you want to achieve. I would attempt

1. to define a view, containing (possibly complex) logic in SQL, else I'd add some PL/SQL to the mix with;
2. A pipelined table function, but using an SQL collection type (instead of the temporary table). A simple example is here: http://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:4447489221109

Number 2 would give you less moving parts and solve your consistency issue.

Mathew Butler

Share Improve this answer Follow

answered Sep 18, 2008 at 12:57

mathewbutler

1,039 ● 6 ● 7

> Thanks for the tip. I'm not going to worry too much about read consistency, since the view and its underlying table function are meant just for reports, and up-to-the-second consistency isn't a requirement. – yukondude Sep 18, 2008 at 15:49

1

The real problem here is the "write-only" table design - by which I mean, it's easy to insert data into it, but tricky and inefficient to get useful information out of it! Your "temporary" table has the structure the "permanent" table should have had in the first place.

Could you perhaps do this:

- Create a permanent table with the better structure
- Populate it to match the data in the first table
- Define a database trigger on the original table to keep the new table in sync from now on

Then you can just select from the new table to perform your reporting.

Share Improve this answer Follow

answered Sep 19, 2008 at 15:33

Tony Andrews

132k ● 23 ● 227 ● 264

> Unfortunately, the original table is at the core of a vendor-supplied system. I thought about the trigger idea myself, although it would have to reference the original table, and that causes other problems. The whole system has a "fragile" feeling to it, so I thought it better to go with a view – yukondude Sep 19, 2008 at 15:41

0

I couldn't agree with you more, HollyStyles. I also used to be a TSQL guy, and find some of Oracle's idiosyncrasies more than a little perplexing. Unfortunately, temp tables aren't as convenient in Oracle, and in this case, other existing SQL logic is expecting to directly query a table, so I give it this view instead. There's really no application logic that exists outside of the database in this system.

Oracle developers do seem to use cursors much more eagerly than I would have thought. Given the bondage & discipline nature of PL/SQL, that's all the more surprising.

Share Improve this answer Follow

answered Aug 22, 2008 at 15:36

yukondude

24.6k ● 13 ● 50 ● 58

0

The simplest solution is:

1. Create a global temporary table containing just IDs you need:

```
CREATE GLOBAL TEMPORARY TABLE tab_ids (id INTEGER)
ON COMMIT DELETE ROWS;
```

2. Populate the temporary table with the IDs you need.
3. Use EXISTS operation in your procedure to select the rows that are only in the IDs table:

```
SELECT yt.col1, yt.col2 FROM your\_table yt
 WHERE EXISTS (
    SELECT 'X' FROM tab_ids ti
     WHERE ti.id = yt.id
 )
```

You can also pass a comma-separated string of IDs as a function parameter and parse it into a table. This is performed by a single SELECT. Want to know more - ask me how :-) But it's got to be a separate question.

Share

Improve this answer

Follow

edited Feb 5, 2019 at 7:06

DJo

2,169 ● 4 ● 30 ● 48

answered Sep 18, 2008 at 1:04

Sergey Stadnik

3,210 ● 8 ● 29 ● 31

Note, you only need to create the temporary table once. – Matthew Watson Sep 18, 2008 at 2:54

The temporary table that I mentioned does essentially consist of IDs (four columns, making up the primary key of the original table). Trouble is, some of the original rows have to be merged together and so it's not quite as simple as you suggest. – yukondude Sep 18, 2008 at 3:27