Think simply and spare yourself a facepalm

Michal Simonik
Freelancer
Czech Republic

Keywords:

Oracle; SQL; PL/SQL; Development

Introduction

Have you ever come up with what you think to be a brilliant solution to a problem? Sure you have, and so have I. But after a brief moment of joy (maybe a day or two if you are lucky), comes that notorious moment of facepalm. Your genius solution crumbles to pieces from a simple comment from your colleague, or from your own discovery after you've had some rest. Your solution is overly complicated and bad. The correct one is, actually, very simple.

In this session, I'll make fun of myself and my mistakes to show listeners how to solve things simply. I'll be presenting examples of simplifying SQL that were given to me for tuning and presenting proper solutions for the problem.

In the following section you can find the examples we will go through in our session to find proper solutions.

That's a good MERGE, right?

My first example is from when I initially learned about MERGE. It came with Oracle 9i, and I loved it. I used it quite often, and I used it in places where it was not good at all. I had to change this exact SQL code two times. The first time was when we were migrating to Oracle 10g, and then again when we were migrating to Oracle 11g.

Sometimes it's good to view your SQL like a piece of art. When you look at it, and feel it, you must like it. You must look at it and say, "Yeah, I can put this on my wall". Some SQL feel horrible no matter what you do, but that's another story. When you look at this SQL, it does not FEEL right … and that can be said about every SQL you'll see in our session.

So use your inner force and tell me what you don't like about this particular query:

```
MERGE INTO metadata_tab met USING
( SELECT DISTINCT zp,
  obd_from_rb,
  obd_to_rb,
  obd_from_rz,
  obd_to_rz
FROM fakta_tab
WHERE module_code = 'P304'
  AND period      = 201209
) fak ON (    NVL(fak.zp, 0)          = NVL(met.zp, 0)
```

```
          AND NVL(fak.obd_from_rb, 0) = NVL(met.obd_from_rb, 0)
          AND NVL(fak.obd_to_rb, 0)   = NVL(met.obd_to_rb, 0)
          AND NVL(fak.obd_from_rz, 0) = NVL(met.obd_from_rz, 0)
          AND NVL(fak.obd_to_rz, 0)   = NVL(met.obd_to_rz, 0))
WHEN NOT MATCHED THEN
  INSERT
    (
      id_meta_ident,
      zp,
      obd_from_rb,
      obd_to_rb,
      obd_from_rz,
      obd_to_rz
    )
    VALUES
    (
      metadata_seq.NEXTVAL+100,
      fak.zp,
      fak.obd_od_rb,
      fak.obd_do_rb,
      fak.obd_od_rz,
      fak.obd_do_rz
    );
```

SELECT from SELECT from SELECT … and from SELECT

This horror story of SQL is actually not mine. It's from an employee of an "*unknown"* company. The listed query is from a validation module. Its purpose is to identify multiplicities in medical care data based on patient ID and the dates when medical care was provided, and then present them in tables like the following:

ID	Patient ID	Date	ID of duplicate
1	FWER241	1.1.2015	5
1	FWER241	1.1.2015	11
1	FWER241	1.1.2015	23
2	33GSW52	4.5.2015	632
2	33GSW52	4.5.2015	745

And, thus, was the underlying code born (table `patient_data` has millions of rows):

```
SELECT id_uni, patient_id_uni, procedure_date_uni, id_dpl
FROM
  (SELECT uni.procedure_date procedure_date_uni,
          uni.patient_id patient_id_uni,
          uni.id id_uni,
          dpl.procedure_date procedure_date_dpl,
          dpl.patient_id patient_id_dpl,
          dpl.id id_dpl
   FROM
    (SELECT master.*
     FROM patient_data master
```

```
     WHERE EXISTS
      (SELECT 1
       FROM
       (SELECT * FROM patient_data) source
        WHERE master.procedure_date = source.procedure_date
          AND master.patient_id     = source.patient_id
          AND master.ROWID          > source.ROWID
        )
      ) dpl,
      (SELECT *
       FROM patient_data
       WHERE ROWID IN (SELECT ID_ROW
                       FROM
                       (SELECT MIN(ROWID) AS ID_ROW,
                               procedure_date,
                               patient_id
                        FROM patient_data
                        GROUP BY procedure_date,
                                 patient_id
                       )
                      )
      ) uni
   WHERE dpl.procedure_date  = uni.procedure_date
     AND dpl.patient_id      = uni.patient_id
  ) unirowid,
  patient_data master_data
WHERE unirowid.patient_id_uni     = master_data.patient_id
  AND unirowid.procedure_date_uni = master_data.procedure_date
  AND unirowid.id_uni             = master_data.id
ORDER BY unirowid.patient_id_uni,
         unirowid.procedure_date_uni,
         unirowid.id_dpl;
```

This tricky problem can be solved with quite a simple … and much faster … piece of SQL. Let's see if we can figure it out …

Copy and paste INSERT

The following example is quite common. We have one set of data which we have to insert into multiple tables in certain forms. My only excuse for this extract is that it's from Oracle 8i, and I was very young at the time, not knowing what SQL and PL/SQL context were. But I did know that there was something like an execution plan!

```
PROCEDURE ...
  ...
  seq NUMBER;
BEGIN
  ...
  INSERT INTO dm_owner.ms_cs2vd501011_tmp (SELECT * FROM
csu2011.cs2vd501011@vm15dbv_ora112i WHERE stat_date =
p_stat_date_in);
```

```
  ...
  FOR sel IN (SELECT * FROM dm_owner.cs2vd501011_tmp)
  LOOP
    SELECT dmv_s_statobj.NEXTVAL INTO seq FROM dual;

    INSERT INTO dmv_t_statobj (id_statobj, … , module)
      VALUES (seq, … , p_module_in);
    INSERT INTO dmv_t_msr (kode_p, … , id_fak_meta_neident)
      VALUES ('ISEKTOR', … ,seq , 1);
    INSERT INTO dmv_t_msr (kode_p, … , id_fak_meta_neident)
      VALUES ('FORMA', … ,seq , 1);
    INSERT INTO dmv_t_msr (kode_p, … , id_fak_meta_neident)
      VALUES ('KATP', … ,seq , 1);
    INSERT INTO dmv_t_msr (kode_p, … , id_fak_meta_neident)
      VALUES ('MNACE', … ,seq , 1);
    INSERT INTO dmv_t_msr (kode_p, … , id_fak_meta_neident)
      VALUES ('POCP', … ,seq , 1);
  END LOOP;
  ...
END;
```

I bet you can do better!

I have to SELECT to see!

Our second to last real-life example is quite simple. We have an online shop, and every order consists of order items. The goal of the following procedure is to delete items from a particular order and write deleted items to output. Not only can the following code be written faster, there is also one nasty flaw in it. Can you find it?

```
PROCEDURE delete_order (p_id orders.id%TYPE) IS
BEGIN
  dbms_output.put_line('Deleting Order ID: '||p_id);

  FOR l_idx IN (SELECT * FROM mtg.order_items WHERE order_id = p_id)
  LOOP
    dbms_output.put_line('Deleting Item ID: '||l_idx.id);
  END LOOP;

  DELETE mtg.order_items WHERE order_id = p_id;
END;
```

Ultimate facepalm

After one of my training sessions I was presented with a simple query which looked like this.

```
SELECT company,
       COUNT(*)
FROM invoices
WHERE can_access( company ) = 1
GROUP BY company;
```

The function can_access contained a single SQL and some simple PL/SQL code. The purpose of this function was to filter records based on the privileges of the current user.

First, I told them that I don't like the usage of a PL/SQL function at all in this kind of situation. It's a CPU burner on SQL and PL/SQL context switches in the first place, and second, it's hiding some important information from Oracle optimizer (selectivity for example).

The response was that this was legacy stuff and that they had to deal with it somehow … ouch. The problem was that they knew that function could be run on a grouped result set (limiting calls a great deal), but they was unable to force Oracle to do so.

Want to hear what I proposed after a long day? It was not very good.

Contact address:

Michal Simonik
Charbulova 23
618 00, Brno
Czech Republic

Phone: +42(0)730-182935
Email misimonik@gmail.com
Internet: www.michalsimonik.com
LinkedIn: https://cz.linkedin.com/in/michalsimonik
Twitter: @MichalSimonik