SQL in Functions

This article was submitted for AIOUG's CONNECT Journal in November 2010

Author: Hemant K Chitale Oracle ACE.

About: Hemant K Chitale has more than 20 years Oracle DBA experience on a large number of platforms. He has been in production support and consulting roles and has worked for or with organisations in financial services, manufacturing, Government and not-for-profit sectors. He has provided Oracle DBA support for a wide variety of applications.

Introduction:
As DBAs and Developers, we are familiar with both SQL and PLSQL. Where the two "languages" meet is when SQL is used in PLSQL Blocks – as in Procedures, Triggers and Functions. The usage of SQL in custom Functions is a special case. ("custom Functions" are our own developed code as opposed to Oracle supplied Functions that are part of the kernel – examples are TO_DATE / TO_CHAR, DECODE, NVL). There are two issues that arise:

a. The SQL is executed each time the Function is called – thus including a Function call for each row in a "parent" SQL results in repeated executions of the Function's SQL
b. Oracle's Read Consistency model, being at the *statement* level is applied to each execution of the SQL separately. This means that data can "seem" to be inconsistent if it is simultaneously updated from another session while it is being retrieved repeatedly by a Function. I plan to demonstrate this in a subsequent article

Description:
In this paper, I walk through an example of the first type of issue with SQL in Functions. (I plan to demonstrate the second issue in a subsequent paper). I demonstrate a case where a PLSQL Function (containing an SQL statement) is used inside an SQL that operates on many rows. I show how the recursive calls have an impact on the total "effort" that Oracle undertakes. Repeated switching between PLSQL and SQL also has significant CPU overheads – "context switches" have to take place, although Oracle has merged the SQL and PLSQL engines considerably, beginning with 9i. As it is PLSQL code that loops "for each row" is a performance killer. Adding SQL recursive calls into the loop doesn't help.

Development Practices:
Common Development "Best Practices" generally include the specification of "Use ReUsable Code". In the context of Oracle this means Packages, Stored Procedures and Functions which can be called from various points in the overall Application. ReUsable Code is perfectly fine where it helps avoid

a. Overheads in Parse Time
b. Latch Contention and Space Management in the Shared Pool
c. Ease of Development (avoid re-writing the same logic repeatedly, in different sections of the application and, possibly, even share the same code across multiple applications).

Thus, it is easy to write a Function and then invoke the same Function wherever it is necessary in the application.

Case:

In this Test Case, I create:

Tables:
FX_RATES: This stores foreign exchange (“forex”) rates for 26 countries over the past 1000 days. The Currency Code and Conversion Rate Date together form the Primary Key.
TRANSACTIONS: This has 100,000 Transactions in the various currencies. The last two columns of this table "CONVERTED_AMT" and "UPDATED_AT" are NULL when the transactions are created. The Transaction ID (an increasing Sequence) is the Primary Key.

Function:
CONVERT_FX: This Function returns the Converted Amount for a given Transaction Amount based on the Transaction Date and Currency Code. It does a lookup on the FX Rates table to do so.
This function makes perfect sense when used in a Forms / Screen / Browser based application where a converted amount has to be presented to a user who is viewing 1 or 5 or 10 records only. Thus, multiple screens can invoke the same function and it does not need to be rewritten into each screen.

User Requirements:
The CONVERTED_AMT and UPDATED_AT columns in the TRANSACTIONS table must be populated by a Batch Program (possibly running weekly or monthly). This may later be used for reporting the summary of Debits and Credits.

The Design of the Application and Batch Program:
By design, the Function, as a stand-alone, is sound. The CONVERT_FX Function is a quick, seemingly, lightweight operation as the FX_RATES table is a small table and a lookup for any particular Currency Code + Conversion Rate Date is very fast. However, the issue in this Test Case is when the Lead Developer decides to use this Function in his Batch Program – in a large UPDATE. The Function is Row oriented – it can be invoked only for a particular combination of Amount + Date + Currency. The Function is then used in an UPDATE statement against the TRANSACTIONS table. Since the Function has to be executed against each row, the UPDATE, unfortunately, ends up invoking the SQL within the Function repeatedly -- once for each row that it is to be executed upon in the Transactions table. For simplicity, I have assumed that the Batch Job has to run against all the 100,000 rows in the Transactions Table.

Data Creation:

This is the creation of the data for this Test Case:

```
SQL>
SQL> -- create the FX_RATES table
SQL> drop table fx_rates;

Table dropped.

SQL> create table fx_rates
  2  ( currency_code varchar2(1) not null,
  3    conv_rate_date  date not null,
  4    conv_rate  number(9,4))
  5  /

Table created.

SQL> alter table fx_rates add constraint my_fx_rates_pk primary key
(currency_code, conv_rate_date);

Table altered.

SQL>
SQL>
SQL> -- create the TRANSACTIONS table
SQL> drop table transactions;

Table dropped.

SQL> create table transactions
  2  (transaction_id  number not null,
  3   transaction_type_code   varchar2(1),
  4   transaction_date date,
  5   transaction_amount number,
  6   transaction_currency_code varchar2(1),
  7   transactions_reference_text varchar2(250),
  8   converted_amount  number,
  9   updated_at  date )
 10  /

Table created.

SQL> alter table transactions add constraint my_transactions_pk primary key
(transaction_id);

Table altered.

SQL>
SQL>
SQL> -- populate the FX_RATES table
SQL> variable i  number;
SQL> variable j  number;
SQL> variable c  number;
SQL>
SQL> begin
  2   for i in 1..1000
  3   loop
  4     for c in 1..26
  5    loop
  6       insert into fx_rates values
  7        (
  8         chr(c+64),  -- 26 different Curency Codes
  9         trunc(sysdate)-i,  -- 1000 days
 10         trunc((dbms_random.value(4,100)),3)  -- Conversion Rates
 11        ) ;
 12     end loop;
 13   end loop;
 14  end ;
 15  /

PL/SQL procedure successfully completed.

SQL>
SQL>
SQL> -- populate the TRANSACTIONS table
SQL> begin
  2    for j in 1..100000
  3    loop
  4    insert into transactions values
  5    (
  6     j,
  7     decode(mod(j,2),0,'D',1,'C'),  -- Transaction Type
  8     trunc(sysdate)-(j*1000/100000),  -- days
  9     trunc(dbms_random.value(5,5000),2),  -- Transaction Amounts
 10     dbms_random.string('U',1),  -- One of 26 Currency Codes
 11     rpad(dbms_random.string('a',25),200,'X'),  -- Txn Reference (text)
 12     NULL,  -- Converted Amount
 13     NULL      -- Update date
 14    ) ;
 15    end loop;
 16  end;
 17  /

PL/SQL procedure successfully completed.

SQL>
SQL> commit;

Commit complete.

SQL>
SQL> exec dbms_stats.gather_table_stats(user,'FX_RATES',method_opt=>'FOR
ALL COLUMNS SIZE 1',estimate_percent=>100,cascade=>TRUE);

PL/SQL procedure successfully completed.

SQL> exec
dbms_stats.gather_table_stats(user,'TRANSACTIONS',method_opt=>'FOR ALL
COLUMNS SIZE 1',estimate_percent=>100,cascade=>TRUE);

PL/SQL procedure successfully completed.

SQL>
SQL> select count(*) from fx_rates;

  COUNT(*)
----------
     26000
```

```
SQL> select count(distinct(currency_code)) from fx_rates;

COUNT(DISTINCT(CURRENCY_CODE))
------------------------------
                            26

SQL> select count(*) from transactions;

  COUNT(*)
----------
    100000

SQL>
SQL> REM fx_rates has rates for 26 currencies and 1000 days
SQL> REM transactions has 100,000 transactions for different currencies
SQL>
SQL>
SQL> -- create the function to convert transaction amount
SQL> create or replace function
  2  convert_fx (currency_code_in in varchar2, amount_in in number,
txn_date_in in date)
  3  return number
  4  is
  5  converted_amt number;
  6  begin
  7    select amount_in*conv_rate into converted_amt from fx_rates where
currency_code=currency_code_in and conv_rate_date=txn_date_in ;
  8    return converted_amt;
  9  end;
 10  /

Function created.

SQL> show errors
No errors.
SQL> -- test the function
SQL> select convert_fx('A',1000,trunc(sysdate)-3) from dual;

CONVERT_FX('A',1000,TRUNC(SYSDATE)-3)
-------------------------------------
                                65617

SQL> select convert_fx('C',3000,trunc(sysdate)-8) from dual;

CONVERT_FX('C',3000,TRUNC(SYSDATE)-8)
-------------------------------------
                               186576

SQL> select currency_code, conv_rate_date, conv_rate from fx_rates
  2  where (currency_code = 'A' and conv_rate_date=trunc(sysdate)-3)
  3  OR    (currency_code = 'C' and conv_rate_date=trunc(sysdate)-8)
  4  /

C CONV_RATE  CONV_RATE
- --------- ----------
A 26-OCT-10     65.617
C 21-OCT-10     62.192

SQL>
SQL>
```

I have created the Test Data and Function.

UPDATE using the Function:
Now, I run the Batch Update, using the Function in an UPDATE statement.

```
SQL> select /*+ FULL (f) */ count(*) from fx_rates f;

  COUNT(*)
----------
     26000

SQL> select /*+ FULL (t) */ count(*) from transactions t;

  COUNT(*)
----------
    100000

SQL>
SQL>
SQL> -- create a new session so that I can report session statistics
SQL> connect hemant/hemant
Connected.
SQL>
SQL> set timing on
SQL> exec dbms_session.session_trace_enable();

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.00
SQL>
SQL> update transactions
  2  set converted_amount =
  3
convert_fx(transaction_currency_code,transaction_amount,transaction_date)
  4  /

100000 rows updated.

Elapsed: 00:00:13.77
SQL>
SQL> exec dbms_session.session_trace_disable();

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.00
SQL>
```

The Update of 100,000 rows took 13.77seconds.

If I run a tkprof against the trace, I get :

```
TKPROF: Release 11.2.0.1.0 - Development on Fri Oct 29 15:26:50 2010

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Trace file: Demo_SQL_in_Function.TRC
Sort options: default

********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************

SQL ID: abnb1tj3n04cv
Plan Hash: 0
BEGIN dbms_session.session_trace_enable(); END;


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           1
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        1      0.00       0.00          0          0          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 91

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                       1        0.00          0.00
  SQL*Net message from client                     1        0.00          0.00
********************************************************************************

update transactions
set converted_amount =
convert_fx(transaction_currency_code,transaction_amount,transaction_date)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1     11.54      12.12          0       3284     201977      100000
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2     11.54      12.13          0       3284     201977      100000

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 91

Rows     Row Source Operation
-------  ---------------------------------------------------
      0  UPDATE  TRANSACTIONS (cr=204326 pr=0 pw=0 time=0 us)
 100000      TABLE  ACCESS  FULL  TRANSACTIONS  (cr=3275  pr=0  pw=0  time=209332  us  cost=889
size=2800000 card=100000)


Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  Disk file operations I/O                       10        0.00          0.00
  control file sequential read                   42        0.00          0.00
  db file sequential read                         4        0.00          0.00
  Data file init write                           14        0.04          0.15
  db file single write                            2        0.00          0.00
  control file parallel write                     6        0.00          0.00
  log file switch (checkpoint incomplete)         1        0.00          0.00
  log buffer space                                1        0.05          0.05
  SQL*Net message to client                       1        0.00          0.00
  SQL*Net message from client                     1        0.00          0.00
********************************************************************************

SQL ID: gz1jkmjq5s5fg
Plan Hash: 3275614890
SELECT :B3 *CONV_RATE
FROM
 FX_RATES WHERE CURRENCY_CODE=:B2 AND CONV_RATE_DATE=:B1


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute 100000      1.18       1.09          0          0          0           0
Fetch   100000      0.57       0.54          0     201000          0        1000
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total   200001      1.76       1.63          0     201000          0        1000

Misses in library cache during parse: 1
Misses in library cache during execute: 1
Optimizer mode: ALL_ROWS
Parsing user id: 91     (recursive depth: 1)

Rows     Row Source Operation
-------  ---------------------------------------------------
      0  TABLE ACCESS BY INDEX ROWID FX_RATES (cr=2 pr=0 pw=0 time=0 us cost=2 size=15 card=1)
      0   INDEX UNIQUE SCAN MY_FX_RATES_PK (cr=2 pr=0 pw=0 time=0 us cost=1 size=0
card=1)(object id 74590)

********************************************************************************

SQL ID: 23d3sap7cask4
Plan Hash: 0
BEGIN dbms_session.session_trace_disable(); END;


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           1
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      0.00       0.00          0          0          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 91



********************************************************************************

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.00       0.00          0          0          0           0
Execute      3     11.54      12.12          0       3284     201977      100002
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        5     11.55      12.13          0       3284     201977      100002

Misses in library cache during parse: 1

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                       2        0.00          0.00
  SQL*Net message from client                     2        0.00          0.00
  Disk file operations I/O                       10        0.00          0.00
  control file sequential read                   42        0.00          0.00
  db file sequential read                         4        0.00          0.00
  Data file init write                           14        0.04          0.15
  db file single write                            2        0.00          0.00
  control file parallel write                     6        0.00          0.00
  log file switch (checkpoint incomplete)         1        0.00          0.00
  log buffer space                                1        0.05          0.05


OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse       12      0.00       0.00          0          0          0           0
Execute 100011      1.18       1.09          0          0          0           0
Fetch   100020      0.57       0.54          0     201042          0        1011
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total   200043      1.76       1.63          0     201042          0        1011

Misses in library cache during parse: 1
Misses in library cache during execute: 1

100003  user  SQL statements in session.
   11  internal SQL statements in session.
100014  SQL statements in session.
********************************************************************************
Trace file: Demo_SQL_in_Function.TRC
Trace file compatibility: 11.1.0.7
Sort options: default

       1  session in tracefile.
  100003  user  SQL statements in trace file.
      11  internal SQL statements in trace file.
  100014  SQL statements in trace file.
       6  unique SQL statements in trace file.
  700259  lines in trace file.
      13  elapsed seconds in trace file.
```

Wait Events : You will notice a number of Wait Events like “Disk file operations I/O”, “Data file init write”, “log buffer space”, “control file sequential read” and “control file parallel write”. These may appear occassionally in production environments depending on the pattern and nature of calls. Since the time for these Waits is not significant, they do not impact the focus of this Test Case.

Furthermore, selected Session Statistics and Waits (querying V$SESSTAT and V$SESSION_EVENT) are:

```
Statistic or Wait                                                Val or Cnt       Time MS
---------------------------------------------------------- ---------------- -------------
CPU : CPU used by this session                                            0     13,330.00
Recursive CPU : recursive cpu usage                                       0      8,350.00
Recursive CPU double counting: recursive cpu usage                        0     -8,350.00
Statistic : consistent gets                                         204,374
Statistic : execute count                                           100,029
Statistic : index fetch by key                                      100,000
Statistic : recursive calls                                         199,239
Statistic : table fetch by rowid                                      1,000
Statistic : table scan blocks gotten                                  3,291
Statistic : table scan rows gotten                                  100,212
Statistic : user calls                                                   26
Wait : Data file init write                                              14        159.71
Wait : Disk file operations I/O                                          11           .82
Wait : SQL*Net message from client                                       16          2.71
Wait : SQL*Net message to client                                         17           .02
Wait : control file parallel write                                        6          3.09
Wait : control file sequential read                                      42           .27
Wait : db file sequential read                                            4           .03
Wait : db file single write                                               2           .44
Wait : events in waitclass Other                                      1           .00
Wait : log buffer space                                               1         55.64
Wait : log file switch (checkpoint incomplete)                        1          6.30
Wait : log file sync                                                  1           .24
                                                                        -------------
sum                                                                         13,559.26
```

These are the critical things to note:

1. OF the total elapsed time of 13.77seconds, CPU Usage was 13.33seconds (tkprof reports a total CPU time of 13.31seconds) of which 8.35seconds were on recursive calls – the repeated calls to the Function, invoking an SQL for each row. Therefore, a very large portion of the total elapsed time was because Oracle had to use CPU time for the recursive calls. (The difference between 13.77seconds and 13.56 in the Statistics + Waits is because of rounding in computation of CPU and Waits. This difference is not significant).

2. Although the "`update transactions`" was executed only once, a "`SELECT :B3 *CONV_RATE FROM FX_RATES WHERE CURRENCY_CODE=:B2 AND CONV_RATE_DATE=:B1`" was executed 100,000 times. (Note how Oracle rewrites SQL in a PLSQL block (the CONVERT_FX function in this case) so that it is all uppercase and with Binds – this is how Oracle makes the SQL reusable and not requiring a hard parse at each execute).

3. The "`recursive calls`" statistic of 199,239 and "`execute count`" statistic of 100,029 are another indication of the number of context switches Oracle had to make between PLSQL and SQL. (I have excluded SYS calls from the tkprof listing).

Using a Correlated UPDATE:

Here is an alternative way of running the UPDATE, as a Correlated Update in SQL ***without invoking the Function itself:***

```
SQL>
SQL> select /*+ FULL (f) */ count(*) from fx_rates f;

  COUNT(*)
----------
     26000

SQL> select /*+ FULL (t) */ count(*) from transactions t;

  COUNT(*)
----------
    100000

SQL>
SQL> connect hemant/hemant
Connected.
SQL> exec dbms_session.session_trace_enable();
```

```
PL/SQL procedure successfully completed.

SQL>
SQL> set timing on
SQL> REM This is the SQL Update -- use a Correlated Subquery on the
PrimaryKey
SQL> update transactions mt
  2  set mt.converted_amount=
  3   (select st.transaction_amount*sf.conv_rate
  4    from transactions st, fx_rates sf
  5    where
  6       st.transaction_currency_code=sf.currency_code
  7    and
  8       st.transaction_date=sf.conv_rate_date
  9    and st.transaction_id=mt.transaction_id),
 10  mt.updated_at=sysdate;

100000 rows updated.

Elapsed: 00:00:04.22
SQL>
SQL> commit;

Commit complete.

Elapsed: 00:00:00.00
SQL> exec dbms_session.session_trace_disable();

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.01
SQL>
```

This Update took 4.22seconds.

The tkprof of the SQL Correlated Update shows:

```
TKPROF: Release 11.2.0.1.0 - Development on Fri Oct 29 15:31:42 2010

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Trace file: Run_Correlated_Update.TRC
Sort options: default

********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************

SQL ID: abnb1tj3n04cv
Plan Hash: 0
BEGIN dbms_session.session_trace_enable(); END;


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           1
```

```
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        1      0.00       0.00          0          0          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 91

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                       1        0.00          0.00
  SQL*Net message from client                     1        0.00          0.00
********************************************************************************

update transactions mt
set mt.converted_amount=
 (select st.transaction_amount*sf.conv_rate
    from transactions st, fx_rates sf
    where
       st.transaction_currency_code=sf.currency_code
    and
       st.transaction_date=sf.conv_rate_date
    and st.transaction_id=mt.transaction_id),
mt.updated_at=sysdate

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      3.65       4.21          0     429159     212502      100000
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      3.65       4.21          0     429159     212502      100000

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 91

Rows     Row Source Operation
-------  ---------------------------------------------------
      0  UPDATE  TRANSACTIONS (cr=417584 pr=0 pw=0 time=0 us)
 202868      TABLE  ACCESS  FULL  TRANSACTIONS  (cr=6645  pr=0  pw=0  time=308451  us  cost=889
size=1900000 card=100000)
   1027   NESTED LOOPS  (cr=422490 pr=0 pw=0 time=0 us cost=3 size=35 card=1)
 102868     TABLE ACCESS BY INDEX ROWID TRANSACTIONS (cr=215727 pr=0 pw=0 time=0 us cost=2
size=20 card=1)
 102868     INDEX UNIQUE SCAN MY_TRANSACTIONS_PK (cr=112859 pr=0 pw=0 time=0 us cost=1 size=0
card=1)(object id 74592)
   1027      TABLE  ACCESS  BY  INDEX  ROWID  FX_RATES  (cr=206763  pr=0  pw=0  time=0  us  cost=1
size=390000 card=26000)
   1027      INDEX UNIQUE SCAN MY_FX_RATES_PK (cr=205736 pr=0 pw=0 time=0 us cost=0 size=0
card=1)(object id 74590)


Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  Disk file operations I/O                       16        0.00          0.00
  reliable message                                2        0.00          0.00
  rdbms ipc reply                                 1        0.00          0.00
  control file sequential read                   84        0.01          0.01
  db file sequential read                         8        0.00          0.00
  Data file init write                           28        0.07          0.24
  db file single write                            4        0.00          0.00
  control file parallel write                    12        0.00          0.00
  log file switch completion                      2        0.00          0.00
  log buffer space                                1        0.30          0.30
  SQL*Net message to client                       1        0.00          0.00
  SQL*Net message from client                     1        0.00          0.00
********************************************************************************

SQL ID: 23wm3kz7rps5y
Plan Hash: 0
commit
```

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          1           0
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      0.00       0.00          0          0          1           0

Misses in library cache during parse: 0
Parsing user id: 91

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                       1        0.00          0.00
  SQL*Net message from client                     1        0.00          0.00
********************************************************************************

SQL ID: 23d3sap7cask4
Plan Hash: 0
BEGIN dbms_session.session_trace_disable(); END;


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           1
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      0.00       0.00          0          0          0           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 91



********************************************************************************

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        3      0.00       0.00          0          0          0           0
Execute      4      3.65       4.21          0     429159     212503      100002
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        7      3.66       4.21          0     429159     212503      100002

Misses in library cache during parse: 1

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                       3        0.00          0.00
  SQL*Net message from client                     3        0.00          0.00
  Disk file operations I/O                       16        0.00          0.00
  reliable message                                2        0.00          0.00
  rdbms ipc reply                                 1        0.00          0.00
  control file sequential read                   84        0.01          0.01
  db file sequential read                         8        0.00          0.00
  Data file init write                           28        0.07          0.24
  db file single write                            4        0.00          0.00
  control file parallel write                    12        0.00          0.00
  log file switch completion                      2        0.00          0.00
  log buffer space                                1        0.30          0.30


OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
```

```
Parse       28      0.00       0.00          0          0          0           0
Execute     28      0.00       0.00          0          0          0           0
Fetch       51      0.00       0.00          0        107          0          28
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      107      0.00       0.00          0        107          0          28

Misses in library cache during parse: 0

    4  user  SQL statements in session.
   28  internal SQL statements in session.
   32  SQL statements in session.
********************************************************************************
Trace file: Run_Correlated_Update.TRC
Trace file compatibility: 11.1.0.7
Sort options: default

       1  session in tracefile.
       4  user  SQL statements in trace file.
      28  internal SQL statements in trace file.
      32  SQL statements in trace file.
       6  unique SQL statements in trace file.
     486  lines in trace file.
       4  elapsed seconds in trace file.
```

Selected Session Statistics and Waits for the Correlated Update are:

```
Statistic or Wait                                          Val or Cnt       Time MS
---------------------------------------------------------- ---------------- -------------
CPU : CPU used by this session                                            0      3,670.00
Recursive CPU : recursive cpu usage                                       0        100.00
Recursive CPU double counting: recursive cpu usage                        0       -100.00
Statistic : consistent gets                                         429,314
Statistic : execute count                                                46
Statistic : index fetch by key                                      205,736
Statistic : recursive calls                                             387
Statistic : table fetch by rowid                                    103,895
Statistic : table scan blocks gotten                                  6,681
Statistic : table scan rows gotten                                  203,304
Statistic : user calls                                                   28
Wait : Data file init write                                              28        245.19
Wait : Disk file operations I/O                                          17          1.31
Wait : SQL*Net message from client                                       17          3.02
Wait : SQL*Net message to client                                         18           .02
Wait : control file parallel write                                       12          7.09
Wait : control file sequential read                                      84         13.54
Wait : db file sequential read                                            8           .07
Wait : db file single write                                               4           .95
Wait : events in waitclass Other                                          4           .08
Wait : log buffer space                                                   1        305.12
Wait : log file switch completion                                         2          9.78
                                                                            -------------
sum                                                                               4,256.16
```

The Correlated UPDATE ran in 4.22seconds --- much faster than the using the Function, which took 13.77seconds.

These are the critical things to note:

1. CPU time was only 3.66 or 3.67 seconds.

2. However, "`consistent gets`" is slightly more than twice that in the first method – at being in excess of 420thousand. Most of these are because of the index lookups. (Similarly "`index fetch by key`" is twice that in the first method). The TRANSACTIONS table is visited twice and indexed lookups on FX_RATES are in a

Nested Loop. Can the Correlated Update be improved further? You could take that as a further exercise.

3. The “`recursive calls`” is only 387 and “`execute count`” is only 46. Much lower than the 199thousand and 100thousand calls of the first method.

A Comparison of the Two Tests:

Statistic\TestCase	Update using Function	SQL Correlated Update	Remarks
Elapsed Time	13.77seconds	4.22seconds	
CPU Time	13.33seconds	03.67seconds	
Recursive CPU Time	08.35seconds	00.10seconds	This is where the major portion of the difference in elapsed time occurs
“recursive calls”	199,239	387	These cause additional for CPU usage
“execute count”	100,029	046	
“consistent gets”	204,374	429,314	This is where the Correlated Update needs improvement
“index fetch by key”	100,000	205,736	
“table fetch by rowid”	1,000	103,895	

Key Findings in this Test Case:

1. Context Switches caused by repeated SQL calls in PLSQL increase the CPU Utilisation. Significantly!

2. Function Calls that run SELECT statements (that retrieve more than a few blocks) can add overhead when executed for *each row*.

Recommendations:

1. Recognise the difference between PLSQL "Procedural" operations and SQL "Set" Operations. Developers who come from backgrounds with other programming languages (eg Java) typically overuse Procedural operations. It is more important to learn SQL first and well.

2. Functions are fine when calling sparingly for a few rows – e.g. in data validation or input screens in an OLTP environment or when operating on small data sets. If, however, a

Function is being invoked repeatedly for each row in the ResultSet, review the impact of the recursive calls to the SQL(s) in the Function.

3. Example of Functions that do not do lookups on tables are the inbuilt 'SYSDATE', 'TO_CHAR' and other functions, which themselves do not run SELECTs on table blocks.

4. Use SQL-only code wherever possible.
When SQL can execute an operation on multiple rows in one single call, it is best to use SQL.
(See Tom Kyte's "*mantra*"
http://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:760210800346068768)