The Devil is in the PeopleSoft MetaSQL

by David Kurtz, Go-Faster Consultancy Ltd.

As I may have mentioned in a previous article, PeopleSoft applications are 'platform agnostic'. As far as possible, they run the same code on different database platform. One of the ways that PeopleSoft achieve this was by adding macros, called MetaSQLs, to their proprietary language and development tools that evaluate differently on different database platform. So long as the developer uses the macro, then the same code will run on any supported database platform. However, some of the macros do not evaluate to sensible results on Oracle, and this can have some nasty side effects on Oracle's Cost-Based Optimiser.

%CurrentDateIn MetaSQL prevents Oracle CBO from correctly evaluating selectivity of predicate.

The cost based optimiser chooses a poor execution plan for a particular critical SQL statement in CRM because the expansion of the *%CurrentDateIn* macro is excessively complicated. The problem occurs in a delivered view PS_RBC_PACKAGE_VW and a related custom view PS_XX_RBCPKCLTR_VW. The views both contain a pair of date conditions, which are coded in line with PeopleSoft standards

```
AND A.FROM_DATE <= %CurrentDateIn
AND A.TO_DATE >= %CurrentDateIn
AND E.FROM_DATE <= %CurrentDateIn
AND E.TO_DATE >= %CurrentDateIn
```

On an Oracle RDBMS, this expands to

```
AND A.FROM_DATE <= TO_DATE(TO_CHAR(SYSDATE,'YYYY-MM-DD'),'YYYY-MM-DD')
AND A.TO_DATE >= TO_DATE(TO_CHAR(SYSDATE,'YYYY-MM-DD'),'YYYY-MM-DD')
AND E.FROM_DATE <= TO_DATE(TO_CHAR(SYSDATE,'YYYY-MM-DD'),'YYYY-MM-DD')
AND E.TO_DATE >= TO_DATE(TO_CHAR(SYSDATE,'YYYY-MM-DD'),'YYYY-MM-DD')
```

with the result that this statement took over 15 seconds to execute. However, if the view is recoded as follows

```
AND A.FROM_DATE <= TRUNC(SYSDATE)
AND A.TO_DATE >= TRUNC(SYSDATE)
AND E.FROM_DATE <= TRUNC(SYSDATE)
AND E.TO_DATE >= TRUNC(SYSDATE)
```

Then the execution time fell to less than 1 second.

To explain why, I shall use a very simple example that is easy to reproduce. In the following script, I have created a table with 1000 rows and a few columns. Column B is just for padding so that the rows are not unrealistically small. Columns C, D and E contain some dates. The data has a perfectly uniform distribution in columns C and D. In column E the distribution is deliberately uneven, there are more dates further into the past.

```
DROP TABLE t1;
CREATE TABLE t1
(a NUMBER NOT NULL
,b VARCHAR2(2000) NOT NULL
,c DATE NOT NULL
,d DATE NOT NULL
,e DATE NOT NULL);

INSERT INTO t1
SELECT rownum
, RPAD(TO_CHAR(TO_DATE(rownum,'J'),'Jsp'),1000,'.') padd
, SYSDATE-rownum+4.2
, SYSDATE-rownum+42
, SYSDATE-sqrt(rownum)+4.2
FROM dba_objects
WHERE rownum <= 1000;
```

Lets look at the execution plans of a few simple SQLs. Both of the following queries return 4 rows, but the cardinality calculated by the optimizer is very different. In the first statement I have used just the simple *TRUNC(SYSDATE)*, the optimizer has correctly worked out that the query will return 4 rows. However, in the second I have used the expansion of the *%CurrentDateIn* macro. Because the predicate contains a function, the optimizer uses a hard coded guess that the selectivity of the condition is 5% of the table.

```
SELECT * FROM t1 WHERE c > TRUNC(SYSDATE);
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=24 Card=4 Bytes=4112)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=24 Card=4 Bytes=4112)

SELECT * FROM t1 WHERE c > TO_DATE(TO_CHAR(SYSDATE,'YYYY-MM-DD'),'YYYY-MM-
DD');
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=24 Card=50 Bytes=51400)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=24 Card=50 Bytes=51400)
```

If I repeat the queries on column D, the cardinality goes up to 42, but the cardinality is still 50.

```
SELECT * FROM t1 WHERE d > TRUNC(SYSDATE);
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=24 Card=42 Bytes=43176)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=24 Card=42 Bytes=43176)

SELECT * FROM t1
WHERE d > TO_DATE(TO_CHAR(SYSDATE,'YYYY-MM-DD'),'YYYY-MM-DD');
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=24 Card=50 Bytes=51400)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=24 Card=50 Bytes=51400)
```

If you try this with column E you get different results again because the distribution of data is not uniform. The query actually returns 23 rows. The cardinality calculation is wrong for both constructions.

```
SELECT * FROM t1 WHERE e > TRUNC(SYSDATE);
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=24 Card=126 Bytes=129528)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=24 Card=126 Bytes=129528)

SELECT * FROM t1
WHERE e > TO_DATE(TO_CHAR(SYSDATE,'YYYY-MM-DD'),'YYYY-MM-DD');
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=24 Card=50 Bytes=51400)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=24 Card=50 Bytes=51400)
```

This is a case where histograms can help, but only with the TRUNC(SYSDATE) construction. The expansion of *%CurrentDateIn* still returns a cardinality of 50.

```
SELECT * FROM t1 WHERE e > TRUNC(SYSDATE);
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=24 Card=24 Bytes=24672)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=24 Card=24 Bytes=24672)

SELECT * FROM t1 WHERE e > TO_DATE(TO_CHAR(SYSDATE,'YYYY-MM-DD'),'YYYY-MM-
DD');
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=24 Card=50 Bytes=51400)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=24 Card=50 Bytes=51400)
```

So this test illustrates that the expansion of *%CurrentDateIn* prevents Oracle's CBO from evaluating the selectivity correctly, and that this causes it to use a guess that is almost certainly incorrect, and in some cases it will lead to an inappropriate execution plan.

There is, of course, a simple workaround. *TRUNC(SYSDATE)* is functionally equivalent to *TO_DATE(TO_CHAR(SYSDATE,'YYYY-MM-DD'),'YYYY-MM-DD')* and can simply be coded into the application as a customisation where necessary.

However, this test shows that if *%CurrentDateIn* evaluated to the more simple *TRUNC(SYSDATE)* the optimizer would have better information with which it might make a better decision. The PeopleSoft MetaSQLs exist so that platform specific code can be introduced into an otherwise platform generic code line. It should be a simple matter to change the expansion in PeopleTools so that this improvement can be made throughout the PeopleSoft applications.

%FirstRows MetaSQL generates FIRST_ROWS hint on Oracle instead of FIRST_ROWS(n)

The following SQL was identified as a long running query from a PeopleTools SQL trace. Note the *FIRST_ROWS* hint.

```
PSAPPSRV.904     1-22827  13.52.01    0.000 Cur#2.904.CRPRD01 RC=0 Dur=0.000
COM Stmt=SELECT  /*+ FIRST_ROWS */ CASE_ID, …
FROM PS_RC_CASE_HD_VW2
WHERE BUSINESS_UNIT = 'HRSUK'
AND RC_CONTACT_NAME = 'Hpolite'
AND RC_VERTICAL = 'HD'
AND MARKET = 'HHD'
ORDER BY  CASE_ID DESC
```

The SQL Statement comes from a SQL exec in the *DERIVEDRCSEARCH.SEARCH_BUTTON.FieldChange* PeopleCode, but it contains the *%FirstRows* MetaSQL which controls how the rows are fetched.

```
…%FirstRows(" | String(&rc_case_qry_rows + 51) | ") …
```

The *%FirstRows* MetaSQL is introducing the */*+ FIRST_ROWS */* hint on Oracle. On Microsoft SQLServer it evaluates to *TOP(n)*, causing the query to return only the first *n* rows of the data set. This is a change of behaviour in PeopleTools on

Oracle introduced in PeopleTools 8.45. In previous versions this MetaSQL evaluated to a null string, thus having no effect on the SQL.

The solution to my performance problem was simply to remove the hint. The following table shows timings in SQL*Plus for the statement with and without the hints.

Hint	Execution Time (CRUAT system)	Execution Time (CRPRD system)
FIRST_ROWS hint	69s	37.08s
No hint, any FIRST_ROWS(n), or ALL_ROWS	5s	<1s

This hint is deprecated in Oracle 9i having been replaced with *FIRST_ROWS(n)*. *FIRST_ROWS* is that it uses a mix of cost and heuristics to find a best plan for fast delivery of the first few rows. Oracle's Performance Tuning and Planning manual says that "using heuristics sometimes leads the CBO to generate a plan with a cost that is significantly larger than the cost of a plan without applying the heuristic. *FIRST_ROWS* is available for backward compatibility and plan stability".

The problem is that the *FIRST_ROWS* hint includes some rules that override the usual cost-based behaviour, including that an index can be used to avoid a sort operation, no matter how expensive the path may be. It most situations *FIRST_ROWS* is simply an inappropriate hint in Oracle 9i.

I have found two support cases (200991709 and 200769258) where this hint was causing a problem, and it is probably in response to these, that there is now a workaround available in PeopleTools 8.46. A new parameter *OracleDisableFirstRowsHint* has been added to the *Database Options*' section of both the Application Server configuration file (*psappsrv.cfg*) and the Process Scheduler configuration files (*psprcs.cfg*). This flag defaults to 0, and should be set to 1 in order to suppress this hint.

However, PeopleTools 8.46 is not certified on any release of Oracle prior to 9.2.0.6.0, and the *FIRST_ROWS* hint was deprecated in Oracle 9i when it was replaced by *FIRST_ROWS(n)*. MetaSQLs are there to produce appropriate platform specific code. It would have been much better if this MetaSQL generated a *FIRST_ROWS(n)* hint in the first place, perhaps resulting in this

```
...Stmt=SELECT /*+ FIRST_ROWS(100)*/ …
```

David Kurtz is a performance specialist working Enterprise PeopleSoft applications and Oracle (www.go-faster.co.uk). He is the chair of UKOUG UNIX SIG, and recently also became a deputy chair of the UKOUG's new PeopleSoft Technology SIG. He is also the author of 'PeopleSoft for the Oracle DBA', published by Apress (www.psftdba.com).