Performance Benefits of ReUse Statement Flag in Application Engine

by David Kurtz (david.kurtz@go-faster.co.uk)

I have achieved some significant performance improvements in some Application Engine programs by just enabling the ReUse Statement flag on certain steps. I thought I would share a recent example of how effective this can be.

I don't think I can improve on the description of this feature in *PeopleBooks*[1]:

"One of the key performance features of PeopleSoft Application Engine is the ability to reuse SQL statements by dedicating a persistent cursor to that statement.

Unless you select the ReUse property for a SQL action, %BIND fields are substituted with literal values in the SQL statement. The database has to recompile the statement every time it is executed.

However, selecting ReUse converts any %BIND fields into real bind variables (:1, :2, and so on), enabling PeopleSoft Application Engine to compile the statement once, dedicate a cursor, and re-execute it with new data multiple times. This reduction in compile time can result in dramatic improvements to performance.

In addition, some databases have SQL statement caching. Every time they receive SQL, they compare it against their cache of previously executed statements to see if they have seen it before. If so, they can reuse the old query plan. This works only if the SQL text matches exactly. This is unlikely with literals instead of bind variables."

In fact most databases do this, and Oracle certainly does.

On Oracle, you could enable CURSOR_SHARING. Then Oracle effectively replaces the literals with bind variables at parse time. However, I certainly would not recommend doing this database-wide. Whenever I have tried this on a PeopleSoft system, it has had severe negative effects elsewhere. I have enabled Cursor Sharing at session level for specific batch programs (using a trigger), but even then it is not always beneficial.

Instead, I do recommend using the *ReUse Statement* flag wherever possible. It cannot just be turned on indiscriminately, the same section in *PeopleBooks* goes on to describe some limitations (which is probably why the default value for the flag is false).

[1] http://download.oracle.com/docs/cd/E13292_01/pt849pbr0/eng/psbooks/tape/chapter.htm?File=tape/htm/tape05.htm%23d0e4240

To illustrate the kind of improvement you can obtain, here is a real-life example. This is an extract from the batch timings report at the end of the Application Engine trace file. We are interested in statements with the high compile count.

ReUse Statement is not enabled on these 4 steps. They account for more that 50% of the overall execution time.

```
                        PeopleSoft Application Engine Timings
                              (All timings in seconds)

           C o m p i l e    E x e c u t e    F e t c h        Total
SQL Statement                 Count   Time     Count   Time     Count   Time     Time
----------------------------- ------- -------- ------- -------- ------- -------- --------
99XxxXxx.Step02.S              8453      2.8    8453    685.6       0      0.0    688.4
99XxxXxx.Step03.S              8453      5.0    8453   2718.8       0      0.0   2723.8
99XxxXxx.Step05.S              8453      0.9    8453    888.4       0      0.0    889.3
99XxxXxx.Step06.S              8453      0.4    8453     17.4       0      0.0     17.8

-------------------------------------------------------------------------------------------
Total run time               :     8416.4
Total time in application SQL :     8195.0   Percent time in application SQL :       97.4%
Total time in PeopleCode      :      192.7   Percent time in PeopleCode      :        2.3%
Total time in cache           :        8.7   Number of calls to cache        :       8542
-------------------------------------------------------------------------------------------
```

Now, I have enabled ReUse Statement on these steps. I have not changed anything else.

```
                       C o m p i l e    E x e c u t e    F e t c h        Total
SQL Statement                 Count   Time     Count   Time     Count   Time     Time
----------------------------- ------- -------- ------- -------- ------- -------- --------
99XxxXxx.Step02.S                 1      0.0    8453    342.3       0      0.0    342.3
99XxxXxx.Step03.S                 1      0.0    8453     83.3       0      0.0     83.3
99XxxXxx.Step05.S                 1      0.0    8453      8.7       0      0.0      8.7
99XxxXxx.Step06.S                 1      0.0    8453      7.6       0      0.0      7.6
-------------------------------------------------------------------------------------------
Total run time               :     5534.1
Total time in application SQL :     5341.7   Percent time in application SQL :       96.5%
Total time in PeopleCode      :      190.8   Percent time in PeopleCode      :        3.4%
Total time in cache           :        1.1   Number of calls to cache        :         90
-------------------------------------------------------------------------------------------
```

Notice that::

- The number of compilations for each step has gone down to 1, though the number of executions remains the same
- The execution time for the first three statements has fallen by nearly 90%.
- The improvement in the 4th statement is quite modest because it did not contain any bind variables, but clearly some of the time reported by Application Engine as SQL execution time is associated with the preparation of a new SQL statement.

To emphasise the point, let's look at the effect on the database. The following are extracts from the TKPROF output for Oracle SQL trace files for these processes.

First the TKPROF without ReUse Statement:

```
OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse   101063   2600.60    2602.83       6197     661559          4           0
Execute 101232   1817.96    3787.17    1572333   73729207   10617830     4770112
Fetch    96186    385.41    1101.47     374425   25986600          0       96285
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total   298481   4803.97    7491.48    1952955  100377366   10617834     4866397

Misses in library cache during parse: 25498
Misses in library cache during execute: 90

Elapsed times include waiting on following events:
Event waited on                             Times   Max. Wait  Total Waited
----------------------------------------   Waited  ----------  ------------
db file sequential read                    1199472        0.36       2601.83
SQL*Net message from client                 130345        1.57        296.50
db file scattered read                        8816        0.39        171.47

OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse   100002     13.51      13.57         17        820         94           0
Execute 131495     30.00      31.31       7025      29277      21164       74315
Fetch   141837    218.77     295.49     159969    3039304         12      519406
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total   373334    262.28     340.38     167011    3069401      21270      593721

160446  user  SQL statements in session.
 70478  internal SQL statements in session.
230924  SQL statements in session.
```

And now with ReUse Statement set on only those four steps

```
OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse    67238     10.24      10.75         47       4415          9           0
Execute 101160   1650.25    4040.88    1766325  129765633   11160830     4781797
Fetch    96123    385.50    1024.50     372737   26097251          0      103844
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total   264521   2045.99    5076.14    2139109  155867299   11160839     4885641

Misses in library cache during parse: 73
Misses in library cache during execute: 21

Elapsed times include waiting on following events:
Event waited on                             Times   Max. Wait  Total Waited
----------------------------------------   Waited  ----------  ------------
db file sequential read                    1506834        0.61       2839.19
SQL*Net message from client                 130312        1.53        258.81
db file scattered read                        8782        0.37        147.01

OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse     1331      0.46       0.46          0        173         16           0
Execute   4044      2.72       5.82      12923      33374      24353      113323
Fetch     5697      8.38      13.43      15550      55431         12       13449
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    11072     11.56      19.72      28473      88978      24381      126772

67425  user  SQL statements in session.
 3154  internal SQL statements in session.
70579  SQL statements in session.
```

- Nearly all the saving is in parse time of non-recursive statements, the rest is the reduction of recursive SQL because there is less parsing.

- There is less parsing, because there are fewer different SQL statements submitted by Application Engine. The number of user statements has fallen from 160446 to 67425. There has also been a huge reduction in recursive SQL.
- The number of misses on the library cache has fallen from 25498 to just 73.
- There has been a reduction in SQL*Net message from client (database idle time) from 296 seconds to 253 because the Application Engine program spends less time compiling SQL statements.

Conclusion

Enabling ReUse Statement can have a very significant effect on the performance of Application Engine batches. It is most effective when SQL statements with *%BIND()* variables are executed within loops. Otherwise, for each execution of the loop, Application Engine must recompile the SQL statement with different bind variable values, which the database will then treat as a new statement that must be hard parsed.

SQL parsing is CPU intensive. Reducing excessive parse also reduces CPU consumption on the database server. It can also reduce physical I/O to the database catalogue. On PeopleSoft 8.x applications that use Unicode, the overhead of parsing is magnified by the use of length checking constraints on all character columns[2]. This is no longer an issue in PeopleSoft version 9 applications which use character semantics.

If you use Oracle's Automatic Memory Management, excessive parsing can cause the database to allocate more memory to the Shared Pool at the expense of the Block Buffer Cache. This in turn can increase physical I/O and can degrade query performance.

Bind Variables are a good thing. You should use them. Therefore, ReUse Statement is also a good thing. You should use that too!

David Kurtz (david.kurtz@go-faster.co.uk) is a performance specialist working with Oracle RDBMS and Enterprise PeopleSoft systems. He is a regular presenter at Oracle conferences, and is the author of 'PeopleSoft for the Oracle DBA', published by Apress (www.psftdba.com).

[2] http://www.go-faster.co.uk/bugs.htm#unicode_oddity.pps