TECHNICAL NOTE

# USE OF ORACLE PLAN STABILITY (STORED OUTLINES) IN PEOPLESOFT GLOBAL PAYROLL

Prepared By David Kurtz, Go-Faster Consultancy Ltd.
Technical Note
Version 1.00
Monday 19 April 2010
(E-mail: david.kurtz@go-faster.co.uk, telephone +44-7771-760660)
File: gp.stored_outlines.doc, 19 April 2010

## Contents

# Introduction

In PeopleSoft for the Oracle DBA I wrote a page (p. 291) explaining why stored outlines were not suitable for use in PeopleSoft. Five years later, my view has not significantly changed. Essentially, stored outlines work best with shared SQL, and there isn't much shared SQL in PeopleSoft, because a lot of it is dynamically generated.

- Code generated by the component processor is dynamically generated. At save time, only fields that have changed are updated.
- PeopleCode can written in such a way that where clauses are dynamically assembled
- nVision reports have variable numbers of criteria on literal tree node IDs in the queries.
- By default in Application Engine, bind variables are converted to literals before the SQL is submitted to the database. Even if this is overridden by enabling *ReUseStatement* in Application Engine or by using Cursor Sharing in the database, the code still wouldn't be sharable. Different instances of Application Engine executing the same program can use different instances on non-shared temporary records, so the tables in otherwise identical SQL statements are not the same. You would get one version of the statement per temporary table instance.

However, there are very limited exceptions to this rule. The SQL in COBOL and SQR programs are more likely to be shareable. Although some programs are coded to generate SQL dynamically, bind variables are passed through to SQL statements, and they use regular tables for working storage and not PeopleSoft temporary records

A Global Payroll customer came to me with a problem where the payroll calculation (GPPDPRUN) would usually run well, but sometimes, the execution plan of a statement would change and the calculation would take additional several hours. It is significant that the Global Payroll engine is written in COBOL. My usual response to this sort of problem in Global Payroll is to add a hint to the stored statement. Usually, I find that only a few statements that are affected. However, after this happened a couple of times in production, it was clear that we couldn't continue to react to these problems. We needed to proactively stop this happening again. This is exactly what stored outlines are designed to do.

## A Simple Example

First, I will give a simple example of using a Stored Outline. In order to demonstrate the control over the optimiser, I am going to use the outline to force the optimiser to do the wrong thing, rather that to ensure it does the right thing.

I will create a table, and I will put a reasonable number of rows into it. Note that initially I am not going to create any index.

```
set autotrace off lines 110
DROP TABLE t PURGE;
CREATE TABLE t(a NUMBER, b VARCHAR2(1000));
TRUNCATE TABLE t;

INSERT INTO t
SELECT rownum, TO_CHAR(TO_DATE(rownum,'J'),'Jsp')
FROM dba_objects
WHERE rownum <= 10000;
```

When I set *create_stored_outlines* to a value other than false, Oracle will collect stored outlines for every SQL statement submitted in that session. Because there is no index on this table, Oracle has no choice but to use a full table scan.

```
set autotrace off
ALTER SESSION SET create_stored_outlines = wibble;
select * from t where a = 42;
ALTER SESSION SET create_stored_outlines = FALSE;
```

I can query the outlines back from USER_OUTLINES

```
SELECT name, sql_text FROM user_outlines WHERE category = 'WIBBLE';

NAME
------------------------------------------------------------------------------
SQL_TEXT
------------------------------------------------------------------------------
SYS_OUTLINE_10032423060495305
select * from t where a = 42
```

Now that I have my outline, I will create an index on the table, and collect statistics

```
CREATE INDEX t ON t(a);
execute sys.dbms_stats.gather_table_stats(ownname=>user,tabname=>'T');
```

So, now, without the benefit of outlines, if I run the same query, I get a new execution plan that uses my new index

```
SQL_ID  7yj1xfjzasaah, child number 1
-------------------------------------
select * from t where a = 42

Plan hash value: 2795797496

-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |      |       |       |     2 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| T    |     1 |    38 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | T    |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("A"=42)
```

If I enable the stored outline category, the query for the same SQL goes back to the full scan specified by the outline. Notice that I needed to flush the Shared Pool because outlines are applied during hard parse.

```
ALTER SYSTEM FLUSH SHARED_POOL;
ALTER SESSION SET statistics_level = ALL;
ALTER SESSION SET use_stored_outlines = WIBBLE;

SQL> select * from t where a = 42;

         A B
---------- ----------------------------------------
        42 Forty-Two

SQL> select * from table(dbms_xplan.display_cursor(NULL,NULL,'TYPICAL'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
SQL_ID  7yj1xfjzasaah, child number 0
-------------------------------------
select * from t where a = 42

Plan hash value: 1601196873

------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |       |       |    16 (100)|          |
|*  1 |  TABLE ACCESS FULL| T    |     1 |    38 |    16   (0)| 00:00:01 |
------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("A"=42)

Note
-----
   - outline "SYS_OUTLINE_10032423060495305" used for this statement
```

Now, if I do something to change the text of the SQL, the outline will not apply and the execution plan will go back to the expected index scan. So in this case my fake GO-FASTER hint appears to make the plan improve[1], but now there is no note to say which outline was used.

```
select /*+GO-FASTER*/ * from t where a = 42;
select * from table(dbms_xplan.display_cursor(NULL,NULL,'TYPICAL'));

SQL> select /*+GO-FASTER*/ * from t where a = 42;

         A B
---------- ----------------------------------------
        42 Forty-Two

SQL> select * from table(dbms_xplan.display_cursor(NULL,NULL,'TYPICAL'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
SQL_ID  b8a9mu8xxws1p, child number 0
-------------------------------------
select /*+GO-FASTER*/ * from t where a = 42

Plan hash value: 2795797496

-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |      |       |       |     2 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| T    |     1 |    38 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | T    |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("A"=42)
```

[1] This fake hint is taken from a humorous demo done by Jonathan Lewis (www.jlcomp.demon.co.uk). The demo is done before any mention is made of store outlines. The note in dbms_xplan which reports use of the outline rather spoils the joke. It can be suppressed with –NOTE option thus

```
select * from table(dbms_xplan.display_cursor(NULL,NULL,'TYPICAL -NOTE'));
```

## Using Stored Outlines in the PeopleSoft GP Engine

Earlier I said that we could apply stored outlines to the Global Payroll calculation engine (GPPDPRUN) because it generally doesn't use dynamic code with embedded literal values.

While outlines are being created, the following privilege needs to be granted. It can be revoked later.

```
GRANT CREATE ANY OUTLINE TO SYSADM;
```

We can create a trigger to collect the stored outlines for a payroll calculation, thus:

- The trigger fires when a payroll calculation process starts or finishes.
- At the start a payroll process it starts collecting stored outlines in a category called the same as the process; GPPDPRUN.
- When the process finishes, outline collection is disabled by setting it back to false.

```
CREATE OR REPLACE TRIGGER sysadm.gfc_create_stored_outlines
BEFORE UPDATE OF runstatus ON sysadm.psprcsrqst
FOR EACH ROW
WHEN (new.prcsname = 'GPPDPRUN' AND (new.runstatus = 7 OR old.runstatus = 7))
DECLARE
 l_sql VARCHAR2(100);
BEGIN
  l_sql := 'ALTER SESSION SET create_stored_outlines = ';
  IF :new.runstatus = 7 THEN
    EXECUTE IMMEDIATE l_sql||:new.prcsname;
  ELSIF :old.runstatus = 7 THEN
    EXECUTE IMMEDIATE l_sql||'FALSE';
  END IF;
EXCEPTION WHEN OTHERS THEN NULL; --because I don't want to crash the process scheduler[2]
END;
/
```

The exact number of outlines that are collected during this process will vary depending upon configuration, and which employees are processed as different payroll rules are invoked.

If no more outlines are to be collected the CREATE ANY OUTLINE privilege can be revoked. This does not prevent the outlines from being used.

```
REVOKE CREATE ANY OUTLINE FROM SYSADM;
```

[2] I am deliberately suppressing any exception raised by this trigger because I do not want to crash any process or the process scheduler under any circumstances. I would rather this trigger doesn't function correctly.

Then, the category of outlines can be used in subsequent executions by replacing the trigger above with the one below, and the execution plans cannot change so long as the SQL doesn't change.

```
CREATE OR REPLACE TRIGGER sysadm.gfc_use_stored_outlines
BEFORE UPDATE OF runstatus ON sysadm.psprcsrqst
FOR EACH ROW
WHEN (new.prcsname = 'GPPDPRUN' AND (new.runstatus = 7 OR old.runstatus = 7))
DECLARE
 l_sql VARCHAR2(100);
BEGIN
  l_sql := 'ALTER SESSION SET use_stored_outlines = ';
  IF :new.runstatus = 7 THEN
    EXECUTE IMMEDIATE l_sql||:new.prcsname;
  ELSIF :old.runstatus = 7 THEN
    EXECUTE IMMEDIATE l_sql||'FALSE';
  END IF;
EXCEPTION WHEN OTHERS THEN NULL; --because I don't want to crash the process scheduler
END;
/
```

After running an identify-and-calc and a cancel, we can see how many of the outlines are actually used.

```
SELECT category, count(*) outlines
, sum(decode(used,'USED',1,0)) used
FROM user_outlines
GROUP BY category
ORDER BY 1
/
```

I have a large number of unused outlines because of additional recursive SQL generated because OPTIMIZER_DYNAMIC_SAMPLING was set 4. This does not occur if this parameter is set to the default of 2.

```
CATEGORY                       OUTLINES       USED
------------------------------ ---------- ----------
GPPDPRUN                              572        281
```

I can then remove the unused outlines.

```
EXECUTE dbms_outln.drop_unused;
```

Used flags on the outlines can be reset, so we later we can see the outlines being used again.

```
BEGIN
 FOR i IN (SELECT * FROM user_outlines WHERE category = 'GPPDPRUN') LOOP
  dbms_outln.clear_used(i.name);
 END LOOP;
END;
/
```

If I want to go back running without outlines, I just disable the trigger

```
ALTER TRIGGER sysadm.stored_outlines DISABLE;
```

To re-enable outlines, just re-enable the trigger.

```
ALTER TRIGGER sysadm.stored_outlines ENABLE;
```

# Test Results

## Test 1: Stored Outline Overhead

I have run various payroll identify & calculations on with and without outlines in two environments with the same code-line but with differing volumes of data.

### Environment 1

| | |
|---|---|
| 281 | Outlines |
| 2770 | Payees |
| 9041 | Segments |

| Test | Duration | Comment |
|---|---|---|
| Collecting Outlines | 00:09:17 | Didn't flush shared pool |
| Using Outlines | 00:08:18 | Flushed Shared Pool prior to recalc |
| Baseline | 00:07:16 | Flushed Shared Pool prior to recalc |

### Environment 2

| | |
|---|---|
| 353 | Outlines |
| 6804 | Payees |
| 17625 | Segments |

| Test | Duration |
|---|---|
| Collecting Outlines | 00:38:17 |
| Using Outlines | 00:39:04 |
| Baseline | 00:42:51 |

This shows that stored outlines do not have an excessive run-time overhead, probably less than an operational variances in this test. The overhead does not seem related to the data volumes being processed. I speculate that the overhead of running with outlines might be outweighed by a saving in parse.

## Test 2: Stabilising Small Payrolls/Group Lists

I have experienced unstable execution plans with processing of small Payrolls in an environment with a much larger streamed payroll, and with Group Lists in environments where partitioning has been introduced to support.

The Global Payroll result tables need to be range partitioned to match the stream definitions. The problem lies with the small payroll having employees scattered across many partitions, while the large payroll only works in a single partition.

A set of stored outlines were created for a full payroll identification and calculation process for the larger payroll, and applied to all subsequent payrolls. Now, I want to prove not only that the outlines were used, but that they have a beneficial effect.

I have three test scenarios.

1. A large streamed payroll calculation was run. It ran without using outlines for 2h 42m, which can considered to be good performance (in fact I used this process to collect the stored outlines).

2. Executed a small non-streamed payroll calculation without outlines. This ran for over 8 hours before it was cancelled. Hence, I don't have data for all statements for this scenario.

3. Execute a small non-streamed payroll calculation again, but this time with outlines enabled. It ran for 2h5m. Not great, considering it has a lot fewer payees than a single stream of the large payroll, but better than scenario 2.

Ideally I should disable outlines again and prove that performance reverts.

I can use the ASH[3] data to see whether the execution plan changed, and what effect that had on performance.

| SQL_ID | SCENARIO 1 | ASH_SECS | SCENARIO 2 | ASH_SECS | SCENARIO 3 | ASH_SECS |
|---|---|---|---|---|---|---|
| 4uzmzh74rdrnz | 2514155560 | 280 | 3829487612 | 28750 | **SAME** | 5023[4] |
| 4n482cm7r9qyn | 1595742310 | 680 | 869376931 | 140 | **SAME** | 889[5] |
| 2f66y2u54ru1v | 1145975676 | 630 | | | **SAME** | 531 |
| 1n2dfvb3jrn2m | 1293172177 | 150 | | | **SAME** | 150 |
| 652y9682bqqvp | 3325291917 | 30 | | | **SAME** | 110 |
| d8gxmqp2zydta | 1716202706 | 10 | 678016679 | 10 | **SAME** | 32 |
| 2np47twhd5nga | 3496258537 | 10 | | | **SAME** | 27 |
| 4ru0618dswz3y[6] | 2621940820 | 10 | | | 539127764 | 22 |
| 4ru0618dswz3y | 539127764 | 100 | | | **SAME** | 22 |
| 4ru0618dswz3y | 3325291917 | 10 | | | 539127764 | 22 |
| 4ru0618dswz3y | 1403673054 | 110 | | | 539127764 | 22 |
| gnnu2hfkjm2yd | 1559321680 | 80 | | | **SAME** | 19 |
| fxz4z38pybu3x | 1478656524 | 30 | | | 4036143672 | 18 |
| 2xkjjwvmyf99c | 1393004311 | 20 | | | **SAME** | 18 |
| a05wrd51zy3kj | 2641254321 | 10 | | | **SAME** | 15 |

[3] Timings for Scenarios 1 and 2 are only accurate to 10 seconds because this is based on the ASH repository exposed in *DBA_HIST_ACTIVE_SESS_HISTORY*, whereas Scenario 3 is accurate to 1 second because it was based on recent history in *v$active_session_history*. See Practical ASH presentation and paper on http://www.go-faster.co.uk.

[4] On the small payroll calculation, without outlines, this statement move than 100 times longer. It had not completed by this stage – the process was cancelled. With outlines enabled this statement used the same execution plan as in scenario 1. It didn't perform that well compared to the large payroll calculation; clearly more work is required for this statement. However, at least it did complete and it did result in improved performance for the small payroll.

[5] This is an example of a statement that performed better on the small payroll without an outline. So, sometimes it is better to let the optimiser change the plan!

[6] This statement executed with 4 different execution plans during the large payroll, but once the outline was applied only one was used, and this seems to be

## Conclusions

Stored Outlines have very limited application in a PeopleSoft system. However, they can easily be collected and used with the PeopleSoft Global Payroll engine. It is just a matter of granting a privilege and using the database triggers on the process request table.

Testing that they actually have the desired effect is quite difficult, because you are trying to prove a negative. I don't think it is adequate simply to say that the outline has been used.

- First you would need an environment where payroll calculation performs well, where you could collect outlines.
- Then you would need a payroll calculation that performs poorly because the execution plan for at least one SQL statement is different
  - Either, on a second environment with exactly the same code.
  - Or in the same environment on a different set of data.
- Then, it would be possible to demonstrate that applying the outline causes the execution plan to revert and restores the performance. This can be confirmed by comparison of the ASH data for the various scenarios.

Even if you don't want to use a stored outline immediately, it might be advantageous to collect them, and have them available when you do encounter a problem.