A Compilation of Technical Tips from the Independent Oracle Users Group

www.ioug.org

Eighth Edition
Leveraging SQL Tuning Techniques for Active Data Guard Databases
By Jim Czuprynski

An Oracle DBA colleague recently presented me with an opportunity to stretch my Oracle 11g Database SQL tuning skills while helping his team of application developers fine-tune the performance of their SQL workloads. “Our developers are pretty sophisticated,” my colleague told me, not without a trace of pride, “and so are their queries. Many of them are several hundred lines long, and the EXPLAIN PLANs are not much shorter, because the business logic behind the application is so complex. Our DBA team is already overwhelmed with work, so we’d really like to leverage Oracle 11g’s SQL tuning tools to help the developers help themselves to build better SQL statements that consume fewer database resources.”

Mentally, I began to construct a game plan: Why not use Oracle 11g’s extensive SQL tuning capabilities, especially SQL Tuning Advisor (STA) and SQL Plan Management (SPM), to capture the most common execution plans for these queries and preserve the “best” plans for reuse by queries with similar ranges of bind variables? “Well, that may not work,” my colleague replied, “because we’re using Business Objects (BO) to actually generate the SQL statements, and, unfortunately, BO cannot take full advantage of bind variables for the SQL statements it issues. And anyway,” my client’s DBA continued, “we need to leave the database’s CURSOR_SHARING initialization parameter set to its default value of EXACT so the myriad ad hoc queries that other application developers are writing can take advantage of hard parsing for their statements as well.”

Undaunted, I pressed ahead, telling my client we could still leverage SPM in these cases to capture the best execution plans within the database itself. “Well, there’s just one other wrinkle,” my colleague DBA replied. “We
followed Oracle’s best practices and implemented Active Data Guard on our physical standby database. Because the standby database is open in READ ONLY mode, we can’t take advantage of SPM because the SYSAUX tablespace is inaccessible.”

**Activating Adaptive Cursor Sharing**

Adaptive Cursor Sharing (ACS), one of the most powerful features of Oracle 11g Release 1, has a relatively unknown benefit: Whenever a session’s CURSOR_SHARING parameter is set to FORCE, ACS will attempt to rewrite an SQL statement that contains literal values for predicates so they instead contain bind variables. ACS thus yielded a method for my colleague’s application developers to leverage bind variables without changing any of their application code.

After identifying which application user accounts could leverage shared cursors most effectively, we forced these user’s sessions to use cursor sharing via an AFTER LOGON trigger, as shown in Figure 1. We also set up SQL tracing for any statements they wanted to trace for possible performance tuning. Note the use of the tracefile_identifier attribute statement to “tag” to each SQL file with an appropriate identifier (APPDEVTUNING) in each trace file we wanted to capture for possible SQL tuning analysis. And, of course, we made sure to build a BEFORE LOGOFF trigger for these accounts so SQL tracing would be deactivated whenever the application user concluded the session.

```
-----
-- AFTER LOGON Trigger to activate CURSOR SHARING=FORCE and trace any SQL
-- statements for specific application developers
-----
CREATE OR REPLACE TRIGGER tr_al_appdevtuning
AFTER LOGON ON DATABASE
BEGIN
  IF USER = 'APPDEV' THEN
    BEGIN
      EXECUTE IMMEDIATE 'ALTER SESSION SET max_dump_file_size = UNLIMITED';
      EXECUTE IMMEDIATE 'ALTER SESSION SET tracefile_identifier = ''DEVTRACES''';
    END
  END IF;
END;
```
EXECUTE IMMEDIATE 'ALTER SESSION SET cursor_sharing = FORCE';
DBMS_MONITOR.SESSION_TRACE_ENABLE(
  session_id => NULL
, serial_num => NULL
, waits => TRUE
, binds => TRUE
, plan_stat => 'ALL_EXECUTIONS'
);
END;
END IF;
END;
/

-----
-- BEFORE LOGOFF Trigger to deactivate SQL tracing
-----
CREATE OR REPLACE TRIGGER tr_bl_appdevtuning
BEFORE LOGOFF ON DATABASE
BEGIN
  IF USER = 'APPDEV' THEN
    BEGIN
      DBMS_MONITOR.SESSION_TRACE_DISABLE(
        session_id => NULL
, serial_num => NULL
      );
    END;
  END IF;
END;
/

Figure 1: LOGON Triggers to Enable Cursor Sharing and SQL Tracing

Once we’d put this infrastructure in place, we were ready to capture any SQL tuning targets via trace files into SQL Tuning Sets. Our first step was to capture the trace files from the physical standby database’s standard diagnostic directory and transfer them to another OS directory specifically created for the purposes of retaining these trace files on the primary database. We then created a directory reference to that new trace directory containing all the SQL trace files and created an empty SQL Tuning Set on the primary database, as shown in Figure 2.
-----
-- Create new directory object and set permissions
-----
DROP DIRECTORY devtraces;
CREATE DIRECTORY devtraces AS '/systraces/appdev/project01';
GRANT READ ON DIRECTORY devtraces TO PUBLIC;

-----
-- Create new empty SQL Tuning Set
-----
BEGIN
  DBMS_SQLTUNE.CREATE_SQLSET('DEVTRACES_STS', 'Developer SQL Traces');
END;
/

Figure 2: Building DIRECTORY Object and Empty SQL Tuning Set

Next, we pointed the DBMS_SQLTUNE package at that directory to capture the SQL statements — including all the bind variables, bind values, execution statistics and wait events — from the SQL trace files, as shown in Figure 3.

-----
-- Clean out the SQL Tuning Set from prior runs
-----
BEGIN
  DBMS_SQLTUNE.DELETE_SQLSET('DEVTRACES_STS');
END;
/

-----
-- Load the SQL Tuning Set from trace files in specified directory
-----
DECLARE
  c_tracestmts SYS_REFCURSOR;
BEGIN
  OPEN  c_tracestmts FOR
    SELECT VALUE(stmts) FROM TABLE(
      DBMS_SQLTUNE.SELECT_SQL_TRACE(
        directory => 'DEVTRACES'
      )
    )
  ;
Figure 3: Loading the SQL Tuning Set on the Primary Database

Figure 4 shows some of the attributes of the SQL statements we’d captured into the DEVTRACES_STS SQL Tuning Set after it was loaded successfully.

<< add query >>
<< add query results >>
/

Figure 4: SQL Statements Captured Within SQL Tuning Set

Bravo! We’re now ready to use all the marvelous features of the SQL Tuning Advisor (STA) to identify any potential tuning advice, as well as generate SQL profiles for the tunable statements.

Closing the Circle: Generating SQL Profiles

We completed our tuning loop by instructing the SQL Tuning Advisor to construct a SQL profile for any SQL statement in the SQL Tuning Set we’d generated previously, as shown in Figure 5.

VARIABLE sts_taskname VARCHAR2(64);
BEGIN
 :sts_taskname := DBMS_SQLTUNE.CREATE_TUNING_TASK(
   sqlset_name => 'DEVTRACES_STS',
   sqlset_owner => 'SYS');
  DBMS_SQLTUNE.EXECUTE_TUNING_TASK(
    task_name => :sts_taskname
);
Figure 5: Using SQL Tuning Advisor to Recommend Tuning Opportunities

Figure 6 shows that DBMS_SQLTUNE recommended accepting SQL profiles for three SQL statements to improve their performance.

```sql
SQL> SET LONG 1000
SQL> SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK('TASK_142') FROM dual;
```

<table>
<thead>
<tr>
<th>GENERAL INFORMATION SECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuning Task Name</td>
</tr>
<tr>
<td>Tuning Task Owner</td>
</tr>
<tr>
<td>Workload Type</td>
</tr>
<tr>
<td>Scope</td>
</tr>
<tr>
<td>Time Limit(seconds)</td>
</tr>
<tr>
<td>Completion Status</td>
</tr>
<tr>
<td>Started at</td>
</tr>
<tr>
<td>Completed at</td>
</tr>
<tr>
<td>SQL Tuning Set (STS) Name</td>
</tr>
<tr>
<td>SQL Tuning Set Owner</td>
</tr>
<tr>
<td>Number of Statements in the STS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUMMARY SECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global SQL Tuning Result Statistics</td>
</tr>
<tr>
<td>Number of SQLs Analyzed</td>
</tr>
<tr>
<td>Number of SQLs in the Report</td>
</tr>
<tr>
<td>Number of SQLs with Findings</td>
</tr>
<tr>
<td>Number of SQLs with SQL profiles recommended</td>
</tr>
</tbody>
</table>
### SQLs with Findings Ordered by Maximum (Profile/Index) Benefit, Object ID

<table>
<thead>
<tr>
<th>object ID</th>
<th>SQL ID</th>
<th>statistics profile(benefit)</th>
<th>index(benefit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>10tpxxybxy9v9</td>
<td>62.17%</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>08qrfms0bg72t</td>
<td>46.96%</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8vthbj9v982vm</td>
<td>43.26%</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 6: Recommended SQL Profiles**

We therefore completed this tuning iteration by executing the ACCEPT_SQL_PROFILE procedure of the DBMS_SQLTUNE package to accept the profiles, as shown in Figure 7.

```
BEGIN
  DBMS_SQLTUNE.ACCEPT_SQL_PROFILE(
    task_name => 'TASK_142',
    object_id => 6,
    task_owner => 'SYS',
    replace => TRUE,
    force_match => TRUE
  );

  DBMS_SQLTUNE.ACCEPT_SQL_PROFILE(
    task_name => 'TASK_142',
    object_id => 7,
    task_owner => 'SYS',
    replace => TRUE,
    force_match => TRUE
  );

  DBMS_SQLTUNE.ACCEPT_SQL_PROFILE(
    task_name => 'TASK_142',
    object_id => 8,
    task_owner => 'SYS',
    replace => TRUE,
    force_match => TRUE
  );
END;
```

**Figure 7: Accepted SQL Profiles**
Accepting these recommendations meant that the new SQL profiles would become available immediately as an alternate SQL profile in the Active Data Guard environment just as soon as the online redo log entries that recorded this change to the corresponding objects in the primary database's SYSAUX tablespace were transmitted to the physical standby.

About the Author

Jim Czuprynski has accumulated more than 30 years of experience during his career in information technology. He has served diverse roles at several Fortune 1000 companies in those three decades — mainframe programmer, applications developer, business analyst and project manager — before becoming an Oracle database administrator in 2001. He currently holds OCP certification for Oracle 9i, 10g and 11g. Czuprynski teaches the core Oracle University database administration courses on behalf of Oracle and its education partners throughout the United States and Canada, instructing several hundred Oracle DBAs since 2005. He was selected as Oracle Education Partner Instructor of the Year in 2009. He continues to write a steady stream of articles that focus on myriad facets of Oracle database administration, with nearly 100 articles to his credit since 2003 at databasejournal.com. His monthly blog, “Generally . . . It Depends” (http://jimczuprynski.wordpress.com), contains his regular observations on all things Oracle and Exadata. He can also be found on Twitter @jczuprynski.

Did You KNOW...

The highest rated member benefit is the IOUG SELECT Journal, IOUG’s quarterly technical journal that provides in-depth technical articles from recognized authorities in Oracle technology.