Why This Session?

- Why Performance Assessment in PL/SQL is Different?
  - Module-oriented
  - Depths of modules
  - Need to know the collective timings

- Traditional assessment methods
  - not relevant for modules

- Three tools
  - PL/SQL Profiler
  - PL/SQL Trace
  - PL/SQL Hierarchical Profiler
Setup

• Install the profiler packages/libraries, etc.
• As SYS, execute
  SQL> @$OH/rdbms/admin/profload.sql
• Create the profiler tables in the user schema
  SQL> @$OH/rdbms/admin/proftab.sql

General Structure

• Start Profiler
  SQL> var r number
  SQL> execute :r := dbms_profiler.start_profiler;
• Execute PL/SQL
• Stop Profiler
  SQL> execute :r := dbms_profiler.stop_profiler;
Check Results

- Get the RUN ID of the profiler run
  - Table PLSQL_PROFILER_RUNS
  - Column: RUNID
- Get the data from PLSQL_PROFILER_DATA and PLSQL_PROFILER_UNITS

Script

```sql
SELECT
    u.unit_number u#,
    u.unit_type,
    u.unit_owner,
    u.unit_name,
    d.line#,
    d.total_occur,
    d.total_time,
    d.min_time,
    d.max_time
FROM   plsql_profiler_units u,
       plsql_profiler_data d
WHERE  u.runid = 2
       and u.unit_number = d.unit_number
       and u.runid = d.runid
ORDER BY u.unit_number, d.line#
```
### Detailed Analysis

- Get the Text from the Line#

```sql
select line, text
from user_source
where name = upper('&name')
order by line;
```
Steps

- Create the Trace Tables
- As SYS, execute
  
  SQL> @OH/rdbms/admin/tracetab.sql

- Grant privs and create synonyms

  create public synonym plsql_trace_runs for plsql_trace_runs;
  create public synonym plsql_trace_events for plsql_trace_events;
  create public synonym plsql_trace_runnumber for plsql_trace_runnumber;
  grant select, insert, update, delete on plsql_trace_runs to public;
  grant select, insert, update, delete on plsql_trace_events to public;
  grant select on plsql_trace_runnumber to public;
General Steps

- Start Trace
  
  \[\text{dbms_trace.set_plsql_trace}\]
  \[\text{(dbms_trace.trace_all_calls)};\]

- Execute PL/SQL

- Stop Trace
  
  \[\text{dbms_trace.clear_plsql_trace;}\]

- Get Trace Run ID
  
  - Table: PLSQL_TRACE_RUNS
  
  - Column: RUNID

Check Trace

\[
\text{select}
\]
\[
\text{event_seq,}
\]
\[
\text{event_time,}
\]
\[
\text{event_unit_owner,}
\]
\[
\text{event_unit,}
\]
\[
\text{event_unit_kind,}
\]
\[
\text{proc_line,}
\]
\[
\text{event_comment}
\]
\[
\text{from plsql_trace_events}
\]
\[
\text{where runid = &runid}
\]
\[
\text{order by event_seq}
\]
Controlling Trace

- Parameters to SET_PLSQL_TRACE
  - dbms_trace.trace_all_calls – trace all the calls
  - dbms_trace.trace_all_exceptions – only exceptions
  - dbms_trace.trace_enabled_calls – trace all calls to enabled functions

- Enable:
  SQL> alter procedure p compile debug;

Hierarchical Profiler
Create the Tables

- Create the tables for profiler
  - in the schema where you want the testing to be done
    SQL> /$OH/rdbs/admin/dbmshptabs.sql
- It creates three tables
  - DBMSHP_FUNCTION_INFO
  - DBMSHP_PARENT_CHILD_INFO
  - DBMSHP_RUNS

Setting Up

- Grant execute on DBMS_HPROF to the user
  SQL> grant execute on dbms_hprof to scott;
- Create a directory to hold the profiler files
  SQL> create directory plsql_dir as 'C:\Arup\Presentations\NYOUG15 - Tracing\plsql_hp';
  SQL> grant all on directory plsql_dir to scott;
**Start / Stop the Profiler**

- **Start**
  begin
  dbms_hprof.start_profiling (  
    location => 'PLSQL_DIR',  
    filename => 'prof.trc'
  );
  end;

- **Stop**
  begin
  dbms_hprof.stop_profiling ;
  end;

**General Steps**
- **Start Profiling**
- **Run the app**
- **Stop Profiling**

**Analyze the Trace**

- **SQL**
  var r number
  begin
    :r := dbms_hprof.analyze(  
      location=>'PLSQL_DIR',  
      filename=>'prof.trc');
  end;
  /
  print r

- **This populates the tables**
Create the Output Files

- Create the reports as HTML files
  
  ```
  $ plshprof -output upd_int prof.trc
  ```

- This creates several html files with prefix upd_int

Example

![Function Elapsed Time (microsecs) Data sorted by Total Subtree Elapsed Time (microsecs)](image)
Difference

- Pass two prof files for analyzing the differences
  
  ```bash
  $ plshprof -output diff <Trace1> <Trace2>
  ```

Profiling Parameters

- Issue: too many recursive calls
  
  proc1
  - proc2
  - proc3
  - Proc4
  - ...and so on

- Parameter `max_depth` in `start_profiling`
- Limits the number it can dive down to
Too many Procedures

- In the ANALYZE procedure, use the TRACE
  begin
  \r := dbms_hprof.analyze(
    location=> 'PLSQL_DIR',
    filename=> 'prof.trc',
    trace   => '"SCOTT"."CALC_INT"."CALC_INT"
  );
  end;
- It performs the tracing at the root CALC_INT

Conclusion

- Find out which specific component in a PL/SQL routine is taking the maximum time
- If you have 11g, use Hierarchical Profiler
- Otherwise, Traditional Profiler
- Focus your tuning efforts on that
- Use PL/SQL Tracing to trace the lines of code in the module
- Use Tracing to identify exceptions
Thank You!

Download Scripts: proligence.com/pres/nyoug15
My Blog: arup.blogspot.com
My Tweeter: @arupnanda