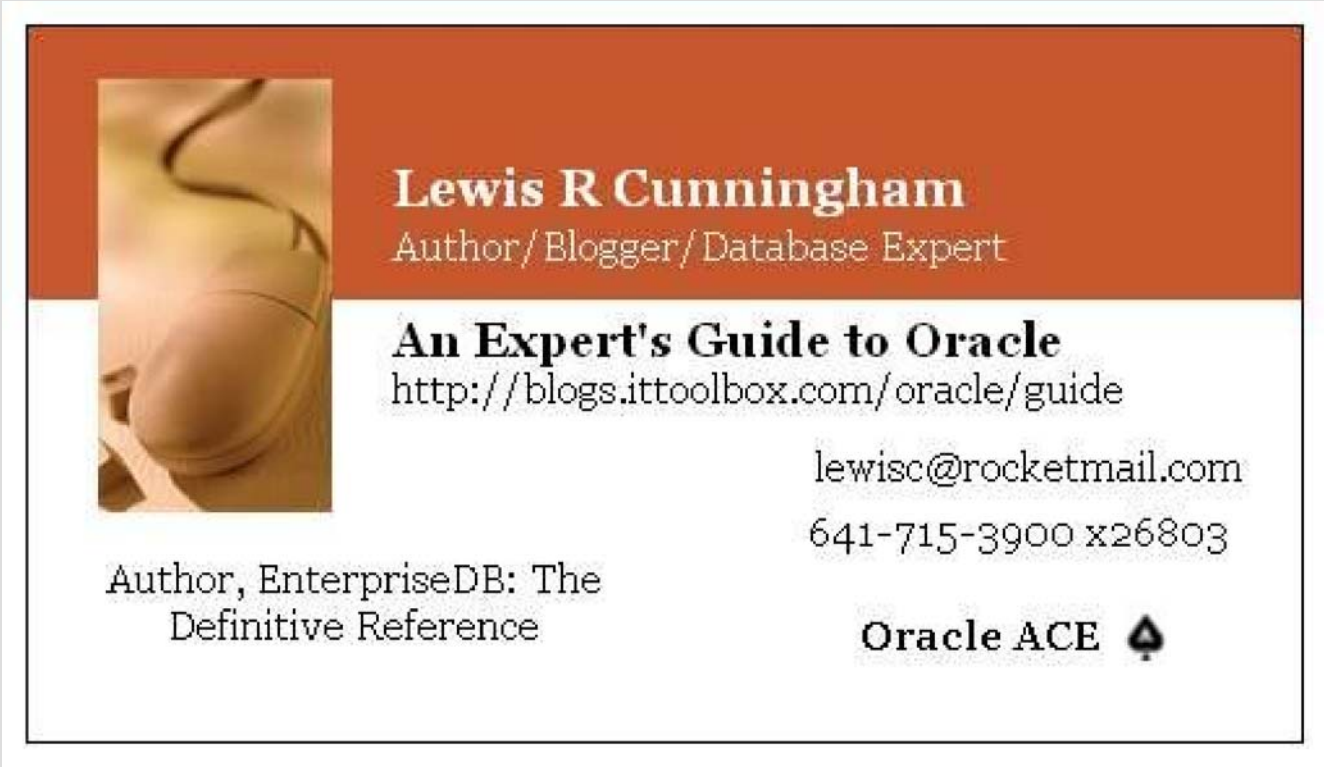
Profiling for Fun and Profit

- Who am I?
  - Lewis Cunningham
  - lewisc@rocketmail.com



Profiling for Fun and Profit

- What are we talking about?
  - SQL Trace
  - DBMS_SESSION
  - DBMS_APPLICATION_INFO
  - DBMS_MONITOR
  - TRCSESS
  - TKPROF
  - ORASrp
- Putting it All Together
- Tips
- Hotsos ILO

Why profile?

We profile code to see how often a certain piece of code executes and how long that piece of code takes to execute.

Profiling can tell you where your bottlenecks are in a process or it can tell you where a well performing piece of code is getting executed 1,000,000 times an hour.

Profiling code helps debug issues as well as proactively gathering performance statistics.

Oracle Profiling

Oracle provides several tools to assist in profiling your code.

We will cover the most common features today.

We will not cover DBMS_PROFILER.

DBMS_PROFILER only gives you basic timing information and is not query-able during a running process.

What is SQL Trace?

- The Oracle database is highly instrumented.
  - Oracle has included the ability to print out detailed processing information
  - Trace is primarily for DML but with the right assistance, it can be used to profile PL/SQL
  - Information gathered:
    - Parse, execute, and fetch counts
    - CPU and elapsed times
    - Physical reads and logical reads
    - Number of rows processed
    - Misses on the library cache
    - Username under which each parse occurred
    - Each commit and rollback
    - Wait event data for each SQL statement, and a summary for each trace file

What is SQL Trace?

- Some database parameters should be set to make the most of SQL Trace
  - TIMED_STATISTICS: This enables and disables the collection of timed statistics, such as CPU and elapsed times, by the SQL Trace facility, as well as the collection of various statistics in the dynamic performance tables.
  - MAX_DUMP_FILE_SIZE: When the SQL Trace facility is enabled at the instance level, every call to the server produces a text line in a file in the operating system's file format.
  - USER_DUMP_DEST: This must fully specify the destination for the trace file according to the conventions of the operating system.

DBMS_SESSION

DBMS_SESSION has many useful utilities but the one we are interested in for profiling is:

SET_INDENTIFIER

- Set a user or session name
- Set to application user if possible
- Can be set via logon trigger, JDBC or OCI
- 64 bytes of case-sensitive, free form text

DBMS_SESSION.SET_IDENTIFIER ('Lewis');

DBMS_APPLICATION_INFO

DBMS_APPLICATION_INFO provides several procedures that are useful for profiling:

SET_MODULE

- Set a module name
- Module is similar to business activity
- MODULE_NAME is 48 bytes of free form text
- Can be NULL
- Can also set an optional ACTION_NAME at the same time

DBMS_APPLICATION_INFO

DBMS_APPLICATION_INFO provides several procedures that are useful for profiling:

SET_ACTION

- An action within a module
- Actions are the individual steps within a procedure
- Every line in the procedure does NOT need to be an action, but can be
- ACTION_NAME is 32 bytes of free form text
- Can be NULL

DBMS_APPLICATION_INFO

```
BEGIN
  DBMS_APPLICATION_INFO.set_module(
    module_name=>'HR_Add_a_bunch_of_loops',
    action_name=>'Begin');

  DBMS_APPLICATION_INFO.set_action(
    action_name=>'Loop 1000000 times');

  -- Loop 1000000 times
  FOR i IN 1..1000000
  LOOP
    v_num_variable := v_num_variable + 1;
  END LOOP;
END;
```

DBMS_MONITOR

DBMS_MONITOR offers several procedures to assist with profile and statistics gathering. While there are several useful procedures, I will concentrate on just two.

CLIENT_ID_STAT_ENABLE

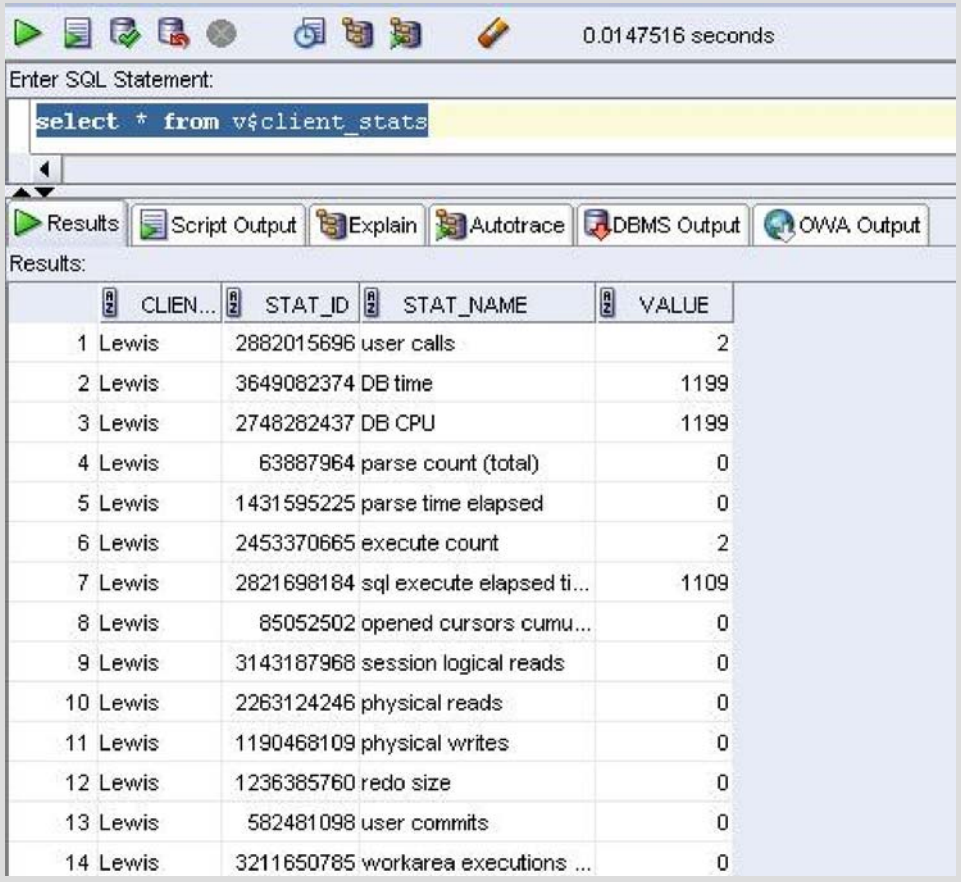
- Gathers statistics for job monitoring
- Dynamic performance stats

```
DBMS_MONITOR.
    client_id=>client_id_stat_enable( 'Lewis');
```

DBMS_MONITOR

CLIENT_ID_STAT_ENABLE

select * from v$client_stats



	CLIEN...	STAT_ID	STAT_NAME	VALUE
1	Lewis	2882015696	user calls	2
2	Lewis	3649082374	DB time	1199
3	Lewis	2748282437	DB CPU	1199
4	Lewis	63887964	parse count (total)	0
5	Lewis	1431595225	parse time elapsed	0
6	Lewis	2453370665	execute count	2
7	Lewis	2821698184	sql execute elapsed ti...	1109
8	Lewis	85052502	opened cursors cumu...	0
9	Lewis	3143187968	session logical reads	0
10	Lewis	2263124246	physical reads	0
11	Lewis	1190468109	physical writes	0
12	Lewis	1236385760	redo size	0
13	Lewis	582481098	user commits	0
14	Lewis	3211650785	workarea executions ...	0

DBMS_MONITOR

CLIENT_ID_TRACE_ENABLE

- Turns tracing on
- Equivalent of
  - alter session set events '10046 trace name context forever, level 8' (or 12, depending)
- Tracing puts performance and diagnostic information in a text file

```
DBMS_MONITOR.
     DBMS_MONITOR.client_id_trace_enable(
        client_id=> 'Lewis',
        waits=>TRUE,
        binds=>TRUE);
```

DBMS_MONITOR

CLIENT_ID_TRACE_ENABLE

Sample Trace File

```
EXEC #3:c=15625,e=9140,p=0,cr=1,cu=5,mis=0,r=1,dep=1,og=4,tim=21133870381
*** 2007-08-20 13:17:23.353
*** CLIENT ID:(Lewis) 2007-08-20 13:17:23.353
WAIT #4: nam='reliable message' ela= 237 channel context=864782644 channel
*** ACTION NAME:(Loop 1000000 times) 2007-08-20 13:17:28.540
*** MODULE NAME:(HR_Add_a_bunch_of_loops) 2007-08-20 13:17:28.540
WAIT #4: nam='PL/SQL lock timer' ela= 4999398 duration=500 p2=0 p3=0 obj#=
*** 2007-08-20 13:17:34.837
*** ACTION NAME:(Loop 10000000 times) 2007-08-20 13:17:34.837
WAIT #4: nam='PL/SQL lock timer' ela= 4999348 duration=500 p2=0 p3=0 obj#=
*** 2007-08-20 13:17:47.291
*** ACTION NAME:(Loop 100000000 times) 2007-08-20 13:17:47.291
```

trcsess

Combines multiple trace files into a single file

Useful in shared server environment
- Multiple processes can trace the same session
- Each process creates its own trace file

Binds can also be created in a separate file, even in a dedicated server environment

Example syntax (search for client_id = Lewis across all trace files in a directory):

```
trcsess clientid=Lewis
```

trcsess

trcsess [output=output_file_name] [session=session_id] [clientid=client_id] [service=service_name] [action=action_name] [module=module_name] [trace_files]

where

output specifies the file where the output is generated. If this option is not specified, then standard output is used for the output.
session consolidates the trace information for the session specified.
clientid consolidates the trace information given client Id.
service consolidates the trace information for the given service name.
action consolidates the trace information for the given action name.
module consolidates the trace information for the given module name.
trace_files is a list of all the trace file names. The wild card character * can be used to specify the trace file names.

tkprof

Tkprof is the “pretty print” for trace files

Most useful when you have a large trace or many DML statements

Not particularly useful for PL/SQL debugging

Can use tkprof to load trace information into the database for statistical research

Will generate explain plans for your SQL

tkprof

tkprof filename1 filename2 [waits=yes|no]
[sort=option] [print=n] [aggregate=yes|no]
[insert=filename3] [sys=yes|no]
[table=schema.table] [explain=user/password]
[record=filename4] [width=n]

Short format: tkprof filename1 filename2
[waits=yes|no] [aggregate=yes|no]

Example: tkprof second_ora_712.trc trace_out.txt
waits=yes aggregate=no

tkprof

```
TKPROF: Release 10.2.0.1.0 - Production on Tue Aug 21 10:13:44 2007

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

Trace file: second_ora_712.trc
Sort options: default

********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
--------------------------------------------------------------------------------

*** SESSION ID:(141.561) 2007-08-20 13:16:15.180

********************************************************************************

delete from wri$_aggregation_enabled
where
 trace_type = :1 AND primary_id = :2 AND qualifier_id1 IS NULL AND
  qualifier_id2 IS NULL AND instance_name IS NULL


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          1          5           1
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      0.00       0.00          0          1          5           1

Misses in library cache during parse: 0
```

Third Party trc Parsers

Session Summary

Instance Name: second
Oracle Version: 10.2.0.1.0
Session ID: 141.561
Total SQL Statements: 2 (0 user statements, 2 internal statements)
Total Transactions: 0 (0 rollback(s), 0 read-only)

Report Shortcuts

Session Flat Profile
Top 5 Statements per Event
Session Call Graph
Events Histograms
Statements

Session Flat Profile

Event Name	% Time	Seconds	Calls	- Time per Call -		
				Avg	Min	Max
unaccounted-for time	98.8%	5,970.3092s				
PL/SQL lock timer	1.2%	74.9940s	15	4.9996s	4.9992s	4.9999s
EXEC calls [CPU]	0.0%	0.0468s	12	0.0039s	0.0000s	0.0156s
reliable message	0.0%	0.0028s	11	0.0002s	0.0002s	0.0003s
PARSE calls [CPU]	0.0%	0.0000s	12	0.0000s	0.0000s	0.0000s
Total	100.0%	6,045.3529s				

Top 5 Statements per Event

PL/SQL lock timer

SQL Statement Id	% Time	Seconds	Calls	- Time per Call -		
				Avg	Min	Max
644964095	80.0%	59.9947s	12	4.9996s	4.9992s	4.9999s
Cursor #5	20.0%	14.9993s	3	4.9998s	4.9996s	4.9999s

Putting it all together

Set the DBMS_SESSION.CLIENT_ID at the beginning of you session (via logon trigger, OCI or JDBC)

Instrument your code using DBMS_APPLICATION_INFO

Turn on statistics and tracing using DBMS_MONITOR

Use TRCSESS, if required, to consolidate trace files

View the raw trc file or use TKPROF or another parser to format the output file

Tips

Use the DBMS_APPLICATION_INFO procedures READ_MODULE and READ_ACTION to save off previous values at the beginning of your routine

Reset module and action to NULL or to their prior values at the end of your procedure

Don’t forget to reset the values in your error handlers

You can use DBMS_APPLICATION_INFO SET_SESSION_LONGOPS before a long running query

View long ops values in V$SESSION_LONGOPS

Sample Code

```
create or replace
PROCEDURE do_a_bunch_of_loops
AS
  v_old_action VARCHAR2(32);
  v_old_module VARCHAR2(48);

BEGIN
```

Sample Code

```
DBMS_APPLICATION_INFO.
  READ_MODULE(
    module_name=>v_old_module,
    action_name=>v_old_action);

DBMS_APPLICATION_INFO.
   set_module(
     module_name=>
                ‘HR_Add_a_bunch_of_loops',
     action_name=>'Begin');
```

Sample Code

```
DBMS_APPLICATION_INFO.
  set_action(
     action_name=>'Loop 1000000 times');

-- Loop 1000000 times
FOR i IN 1..1000000 LOOP
  v_num_variable := v_num_variable + 1;
END LOOP;
```

Sample Code

```
 DBMS_OUTPUT.PUT_LINE(
     'v_num_variable=' ||
     to_char( v_num_variable ) );

 DBMS_APPLICATION_INFO.
     set_module(v_old_module,v_old_action);
END;
```

Call the procedure example

```
 set serveroutput on
BEGIN
  DBMS_SESSION.SET_IDENTIFIER ('Lewis');

  DBMS_MONITOR.client_id_stat_enable( 'Lewis');
  DBMS_MONITOR.client_id_trace_enable(
        'Lewis', TRUE, TRUE);

  do_a_bunch_of_loops;

  DBMS_MONITOR.client_id_stat_disable( 'Lewis');
  DBMS_MONITOR.client_id_trace_disable( 'Lewis');
END;
```

Hotsos ILO

- What is ILO?
  - Open Source Instrumentation Library for Oracle
- What does ILO do?
  - ILO abstracts the DBMS_APPLICATION_INFO and DBMS_SESSION calls into a higher level library
  - Developer's don't need to worry about when to set a trace
- What are the downsides to ILO?
  - Requires SYSDBA to install/configure
  - Uses internal objects (such as SYS.Dbms_System.ksdddt;)
  - You don't own the code; it is released as LGPL

Hotsos ILO

- Why use it? (from the read me)
- With HOTSOS_ILO, you can track the performance of any business task on your system, and you can report on system performance in exactly the language your management wants to hear: response time for specific business tasks.
- With HOTSOS_ILO, your Oracle trace files will be properly time-scoped and won't contain unwanted events like the 'SQL*Net message from client‘s events that commonly plague trace files.
- HOTSOS_ILO contains hooks to other Hotsos tools that allow you to do things like store response time histories and report on performance trends for specific tasks, and profile specific tasks or subtasks within large trace files.

Hotsos ILO

Is ILO that different from directly calling the Oracle procedures?

- Not really. This blurb is from the readme:

All the application developer needs to do is mark the beginnings and endings of business tasks. For example, imagine that a developer is writing code that a user will later regard as code that adds an employee to the HR database. Using HOTSOS_ILO, the developer will mark the code path as follows:

```
BEGIN
    HOTSOS_ILO_TASK.BEGIN_TASK(
         'HR', 'Add employee', '', rec.name);
     -- code to add the employee goes here
     HOTSOS_ILO_TASK.END_TASK;
END;
```

- That doesn't look much different, does it?
- ILO does not provide a trace file parser.

Summary

- Instrument your code!
- Instrumentation allows you to proactively look for performance issues
- Instrumentation allows you to better debug your applications
- Oracle provides plenty of instrumentation support
- Third parties provide plenty of assistance in instrumenting and parsing the resultant trace files
- It’s really not hard once you do it

Profiling for Fun and Profit

- Who am I?
  - Lewis Cunningham
  - lewisc@rocketmail.com