COLLABORATE 12

TECHNOLOGY AND APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

Dynamic Debugging and Instrumentation of Production PL/SQL

COLLABORATE 12
TECHNOLOGY AND APPLICATIONS FORUM FOR THE ORACLE COMMUNITY

IOUG
independent oracle users group

Bill Who?

- RMOUG, IOUG and UTOUG. 10 yrs
- PL/SQL enthusiast. 16 yrs
  - Andersen Consulting – SF, Denver
  - New Global Telecom – Golden
  - Structure Consulting Group – Houston
  - Church of Jesus Christ of Latter Day Saints
  - DBArtisans.com (place to keep my stuff)

Some Lessons Learned

- There's always another bug.
- Involve users early and often.
- Is it redundant? Simplify.
- Test with dirty data and plenty of it.
- Get another pair of eyes.
- Document the code well.
- If it isn't simple, back up, reduce, reuse.
- Have compassion on your successor.

Survey Says!

- Strictly DBA? Strictly developers? Hybrids?
- Written PL/SQL that was release to Prod?
- Who has never had anything go wrong in that production PL/SQL?
- When things do go wrong, how long does it take to find out what it is doing, what it did, why it did what it did?
- How did the users/mgmt appreciate your handling of the issue?

OR

Agenda

- Typical Production Problem Lifecycle
- Define instrumentation
- Oracle built-ins for instrumentation
- Develop requirements of good instrumentation
- Existing instrumentation libraries
- Demos: Debugging & adding Instruments

Lifecycle of Production Problem

- Become aware of a problem
- Find the source of the problem
- Fix the source of the problem
- Repair issues the problem caused
- Rebuild trust (costliest, longest step)
- Improve so problem doesn’t happen again

Awareness of Problem

- Non-instrumented:
  - Silent Fester
  - Side Effect
  - New Guy
  - Phone Call
  - Email
  - Pink slip
- Instrumented: Proactive monitoring

Finding the Problem Source

- Options without instrumentation:
  - Hunt, poke, prod, peek, query, hope, trace, explain, waits, OEM, TOAD, AskTom, …
- Options with instrumentation:
  A. Review what happened
  B. Replicate and monitor in real-time
  C. Add to proactive monitor

Agenda

- Typical Production Problem Lifecycle
- Define instrumentation
- Oracle built-ins for instrumentation
- Develop requirements of good instrumentation
- Existing instrumentation libraries
- Demos: Debugging & adding instruments

Instrumentation

- Big word, but more familiar than it seems
  - Dashboard of car and airplane
  - Task Manager/Process Explorer
  - Network Operations Center
  - What do they have in common?
- Instrumentation: the process of fitting applications with code that directs runtime context to some destination where it can be useful.

Pics

Runtime Context

- Who, when, what was passed in, what changed, time taken, errors and warnings encountered, etc.
- Four categories of runtime insight:
  - Debugging – *disabled* by default
  - Logging – *enabled* by default
    - Error, warning, informational, metric
  - Column-level audit data
  - Monitor and Trace

Destination

- Direct runtime context to stdout (screen), V$ views, a logging table, a log file on the database host, a queue for asynchronous re-direction, a DBMS pipe or alert, and other slower, more complex alternatives like HTTP, FTP and UDP callouts.
- IMHO: Best option is writing to heap table within an anonymous transaction

Agenda

- Typical Production Problem Lifecycle
- Define instrumentation
- Oracle built-ins for instrumentation
- Develop requirements of good instrumentation
- Existing instrumentation libraries
- Demos: Debugging & adding instruments

Oracle Instrumentation APIs?

- Column-level Auditing
  - 11g Flashback Data Archive (Total Recall)
  - Most build custom triggers to capture change, and tables to hold the history.
- Metrics
  - DBMS_UTILITY.get_time {DEMO}
  - DBMS_PROFILER {DEMO}

Oracle Instrumentation APIs?

- Logging/Debugging
  - DBMS_OUTPUT (Dev only)
  - DBMS_DEBUG & DBMS_DEBUG_JDWP (Yes)
  - ORADEBUG (Rarely)
  - DBMS_ERRLOG (No)
  - DBMS_ALERT (No)
  - DBMS_PIPE (Possibly) {DEMOS}

Oracle Instrumentation APIs?

- Logging/Debugging
  - DBMS_SYSTEM {DEMO}

```
<msg time='2012-02-03T18:30:40.283-07:00' org_id='oracle' comp_id='rdbms'
 client_id='bcoulam' type='UNKNOWN' level='16'
 host_id='R9AXR65' host_addr='fe80::cd94:25d3:ee1a:9777%11' module='PL/SQL
   Developer'
 pid='15156'>
 <txt>WARNING! Here is my real-time msg logged to alert.log
 </txt>
</msg>
```

Oracle Instrumentation APIs?

- Logging/Debugging
  - UTL_FILE
  - UTL_HTTP
  - UTL_TCP

Oracle Instrumentation APIs?

- Monitor and Trace Metadata
  - DBMS_SESSION.set_identifier
    - Sets client_identifier seen in V$SESSION, AUDIT, trace and elsewhere.
  - DBMS_APPLICATION_INFO
    - set_module(), set_action(), set_client_info()
    - set_session_longops()
  - USERENV namespace and V$SESSION

{DEMO}

Agenda

- Typical Production Problem Lifecycle
- Define instrumentation
- Oracle built-ins for instrumentation
- Develop requirements of good instrumentation
- Existing instrumentation libraries
- Demos: Debugging & adding instruments

Sweet Instrumentation

- Simple API to clock and record metrics
  - Should handle nested timers
- Simple API to tag sessions and long operations
  - Should handle nested tagging
- Simple API to write files
- Simple API for static & dynamic log messages
  - Must be independent of the calling transaction
- Standard method of handling exceptions
- Routines to gather client & session metadata so the APIs can remain simple
- Column-level auditing structures and triggers

Sweet Instrumentation

- **Dynamic Logging**
  - Off by default, and low overhead, so insightful debug lines can remain in Prod code
  - Can be turned on and off without Prod interruption
  - Toggles kept in a table or application context
  - Turn on for a PL/SQL unit or list of units, session, end user or named process, IP address, domain

Sweet Instrumentation

- **Simple**

```
dbg('Calling X with '||i_parm);
info('BEGIN: Nightly Reconcile');
warn('X took '||l_diff||' s too long');
err();
tag();
startT(); <stuff> stopT(); elapsed();
```

- **Origin Metadata Transparently Derived**
  - Time, unit, line, caller identifiers
  - End user identifiable from end-to-end

Sweet Instrumentation

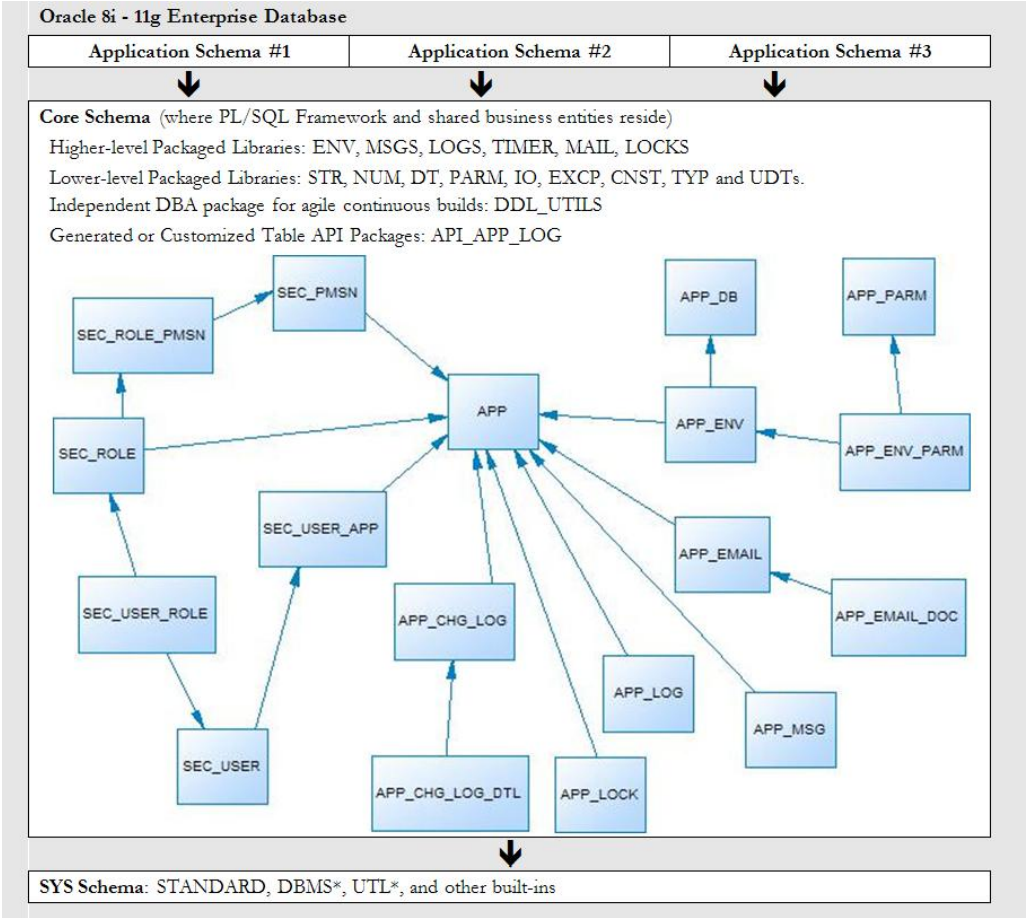
- **Choice of Output**
  - Minimally: to table and screen
  - Optionally: to file
  - Nice to have: ftp, pipe, AQ, http, etc.
  - Output must be transaction-independent

Agenda

- Typical Production Problem Lifecycle
- Define instrumentation
- Oracle built-ins for instrumentation
- Develop requirements of good instrumentation
- Existing instrumentation libraries
- Demos: Debugging & adding instruments

Resource Name	License	Purpose	Location & Notes
Google Code	Free	Library of libraries	http://code.google.com/hosting/search?q=label:plsql
Feuerstein's PL/SQL Obsession	Free	Repository of all things SF and PL/SQL	http://www.toadworld.com/sf
QCGU (Quest CodeGen Utility)	Free	Full framework Standards, Scripts, Template Factory, Code Generation, + more	http://codegen.inside.quest.com/index.jspa Latest incarnation of Feuerstein's vast reservoir of experience. (successor of QXNO, PL/Vision, and PL/Generator.)
PL/SQL Starter	Free	Author's full framework.	http://sourceforge.net/projects/plsqlframestart
Simple Starter	Free	Logging, Timing, Auditing, Debugging, Error Handling, + more	Simplified PL/SQL Starter to just logging, timing and auditing components (and the low-level packages they depend on). Designed to be used in one schema. Install and begin using in under a minute.
GED Toolkit	$120-$1200	Almost full framework	http://gedtoolkit.com Includes APEX UI to administer jobs and tables. Monitor processing.
PL/Vision	Free	Framework, API Generator, + more	http://toadworld.com/Downloads/PLVisionFreeware/tabid/687/Default.aspx Replaced by QXNO and then QCGU. Not supported.
Log4ora	Free	Logging	http://code.google.com/p/log4ora/ Fresh, full-featured logging library. Alerts. AQ. Easy to use. Good stuff.
ILO	Free	Timing and Tuning	http://sourceforge.net/projects/ilo From the sharp minds at Hotsos
Quest Error Manager	Free	Error Handling	http://www.toadworld.com/LinkClick.aspx?link=685&tabid=153 Included in QCGU. But offered separately as well. Not supported.
Plsql-commons	Free	Collection of utilities, including logging	http://code.google.com/p/plsql-commons
Log4oracle-plsql	Free	Logging	http://code.google.com/p/log4oracle-plsql Seems like an active project, but could not find code to download…
Log4PLSQL	Free	Logging	http://sourceforge.net/projects/log4plsql Popular, but aging and complex log4j analog in PL/SQL
Logger	Free	Logging	http://sn.im/logger1.4 Recently orphaned when Oracle decommissioned its samplecode site. Simple. Easy to use.
Orate	Free	Logging	http://sourceforge.net/projects/orate Never used it, but has been around a while. Still active.

PL/SQL Starter Framework

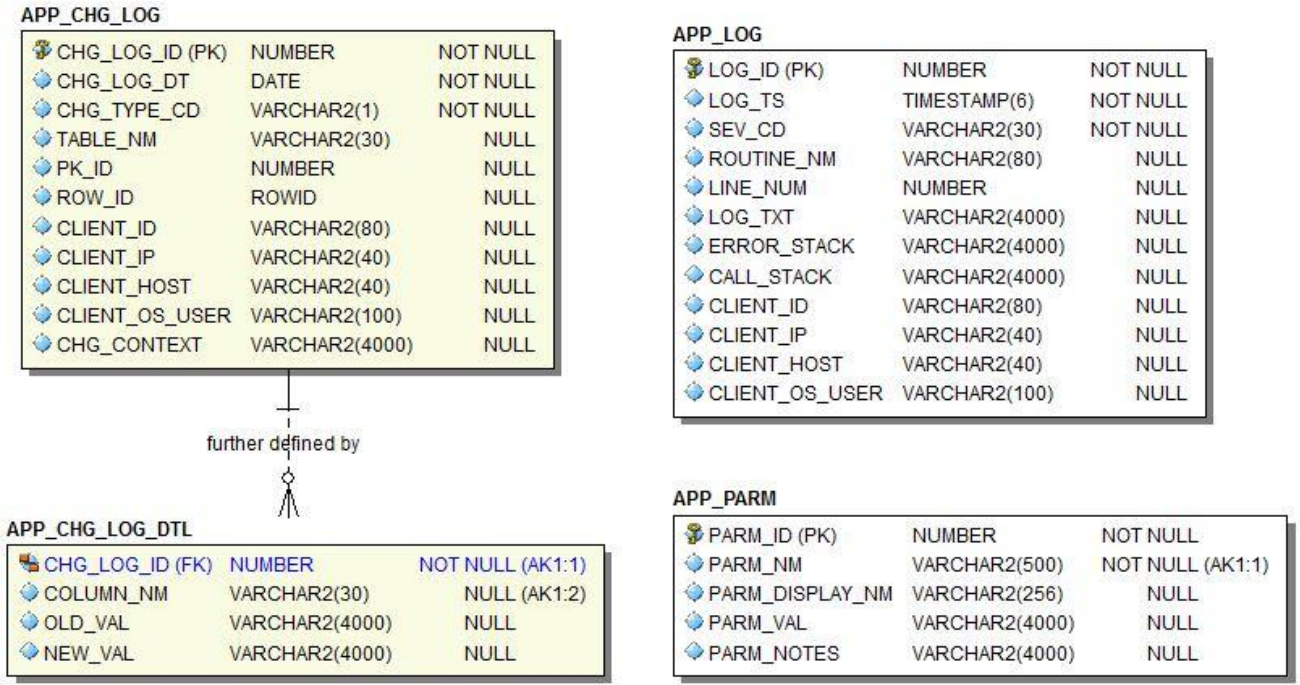


“Starter” too much?

- Thousands of downloads, but not much feedback or developer contributions.
- 21 services and 55 objects
- Some shops only have one major app schema per DB
- 60 page doc and days to week learning curve
- Security often done in directory server now
- Common messages almost never used
- Email-from-DB tables rarely used
- Locking always needs customization

Simple Starter

- LOGS, TIMER, ENV, gen_audit_triggers.sql



Library	Main Routines	Supporting Components and Notes
Auditing: **gen_audit_triggers.sql**		APP_CHG_LOG, APP_CHG_LOG_DTL (tables)
Metrics: **TIMER** (package)	**startme()** **stopme()** **elapsed()**	Uses DBMS_UTILITY
Debugging, Logging and Error Handling: **LOGS** (package) EXCP (package meant to be used only by LOGS) APP_LOG_API (pkg meant to be used only by LOGS)	**err()** **warn()** **info()** **dbg()**	APP_LOG (table) TRIM_APP_LOG (scheduled job)
Connection Metadata: **ENV** (package)	**init/reset_client_ctx()** **tag/untag()** **tag_longop()**	Uses DBMS_DB_VERSION, DBMS_APPLICATION_INFO, DBMS_SYSTEM, v$session and v$mystat.
File Operations: IO (meant to be used primarily by LOGS)	write_line() write_lines(0 p()	Uses UTL_FILE, DBMS_LOB
Dynamic (Table-Driven) Parameters/Properties: PARM (package)	get_val()	APP_PARM (table)
Extras (required for the seven libraries above to function): CNST, TYP, DDL_UTILS, DT, STR, NUM (packages)	These are libraries of application-wide constants and subtypes, build utility functions; date, string and number manipulation routines.	

Simple: Auditing

- Run *gen_audit_triggers.sql*. Generates trigger for every table in your schema.
- Remove triggers not needed. Remove auditing on columns not needed. Done.
- Audited changes are recorded to APP_CHG_LOG and APP_CHG_LOG_DTL
- May need view or materialized view to simplify access to audit data.

Simple: Metrics

- TIMER package
  - `startme(`*`timer name`*`)`
  - `stopme(`*`timer name`*`)`
  - `elapsed(`*`timer name`*`)`
- Log elapsed times
- Create separate automated processes to monitor metrics, learn from them over time, and notify when anomalies are detected.

Simple: Log & Debug

- LOGS package
  - `info(msg), warn(msg), err(msg)`
    - record important data, expected and unexpected error conditions
  - `dbg(msg)`
    - to document code and leave hooks for dynamic, real-time logging
  - `set_dbg (boolean and directed)`

Simple: Log Destination

- Screen (10K msgs = 1 sec)
  - Quick-and-dirty testing and debugging.
- Log Table (10K msgs = 4 sec)
  - A default job keeps the table trimmed to a couple weeks of data.
- File (10K msgs = 15 sec)
- Pipe (10K msgs = 8 sec + 4 sec to log them)

Simple: Debug Parameters

- Parameters in APP_PARM
  - *Debug* (on/off, session, unit, user)
  - *Debug Toggle Check Interval* (in minutes)
  - *Default Log Targets* (Screen=N,Table=Y,File=N)
- Parameter values table-driven
- Parameters can be temporarily overridden through logs.set* routines

Simple: Monitor and Trace

- ENV offers:
  - `tag/untag` to modify module, action and client_info
  - `tag_longop` to track long operations
  - `init_client_ctx(), reset_client_ctx()`
    - Front end client should pass the user's ID to the DB through init_client_ctx, and reset_client_ctx upon returning the connection to the pool.

Simple Framework: Install

- Go to Sourceforge.net
- Search for PL/SQL framework. First option.
- Select Browse All Files.
  - Drill to plsqlfmwksimple/2.1.
  - Download and unzip Simple.zip
- Start SQL*Plus as SYS
  - If installing to existing schema, remove DROP and CREATE USER statements.
  - Run __InstallSimpleFmwk.sql
- Done.

Agenda

- Typical Production Problem Lifecycle
- Define instrumentation
- Oracle built-ins for instrumentation
- Develop requirements of good instrumentation
- Existing instrumentation libraries
- Demos: Debugging & adding instruments

Putting it all Together

- Solution Manager just called.
  - After last night's release, she is not getting her daily file about the critical problem/solution repository.

{LIVE DEMO real-time debugging, monitoring, and adding instrumentation to two pages of code}

Putting it all Together

- Write and document public interface.
- Write tests that all fail.
- Write body in pseudo-code.
- Fill in the algorithm, making sure routine does one thing and one thing well. Ensure it uses assertions to check assumptions. Clean. To standard. Formatted.
- Wrap pseudo-code with log and debug calls, adding a little runtime context. Voila! 3-birds with one stone.
- Then I run the tests until they all work, using the instrumentation and metrics if there is trouble.

Conclusion

- Instrumentation should be in place *before* production problems occur.
- But it can be added easily *after* as well.
- Adopt or build a standard library.
  - It must be simple and easy to use.
- Encourage or enforce its use.
- Do it today! It's easy and rewarding.

vs.

Q&A

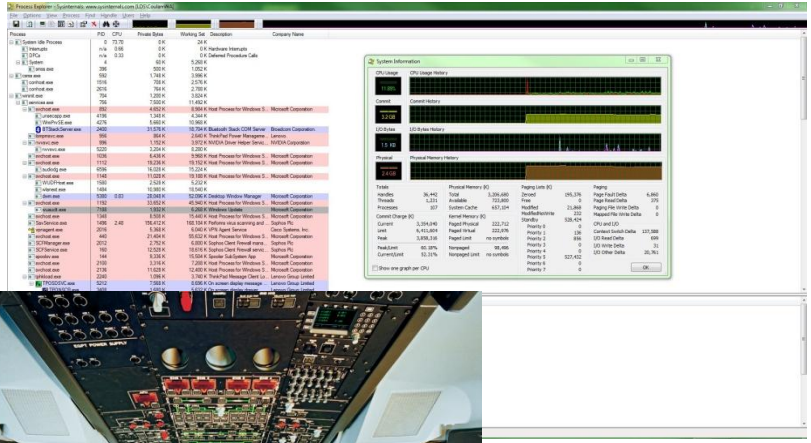
- Questions?

Contact: bcoulam@yahoo.com

Framework:
sourceforge.net/projects/plsqlframestart/

Instrumentation: Dials, Graphs, Guages

Car



Computer

Network Operations Center

Passenger Jet

Return