Oracle PL/SQL Best Practices

Bullet Proof Your PL/SQL

For those few times when something doesn't go as planned....

Audio will be available through your computer/headset.
You do not need to dial in (and access that way is limited).
If you do dial in, security code is 1111.

Download this PPT from oracle.com/oll, search for PL/SQL, visit the PL/SQL Learning Library.

Steven Feuerstein
Architect, Applied PL/SQL
email: steven.feuerstein@oracle.com
blog: stevenfeuersteinonplsql.blogspot.com
twitter: sfonplsql

ORACLE

Bullet Proof PL/SQL

- The #1 fantasy in software development: everything goes as planned.
  - Users gave us accurate specifications
  - Users use the application properly
  - There are no bugs in our code, etc., etc., etc.
- As with most fantasies, not much connection with reality.
- So if you want to write high quality, *popular* applications, you need to anticipate problems and make your code "bullet proof".
- Let's take a whirlwind trip through features in and techniques with PL/SQL to achieve that.

All referenced code available in my demo.zip file from the PL/SQL Learning Library: oracle.com/oll, search on "PL/SQL".

All that we can talk about

- Handling exceptions
  - Built in functionality in DBMS_UTILITIES and UTL_CALL_STACK
  - Where and how to best handle
- FORALL with SAVE EXCEPTIONS: suppress errors at statement level
- LOG ERRORS for DML: suppress errors at row level
- AFTER SERVERERROR trigger: instance wide error handling
- Validating assumptions with assertions: no surprises!
- Tracing execution to help diagnose bugs: how did that surprise happen?
- Wow! That's a lot...let's get going

Handling Exceptions

- The EXCEPTION section consolidates all error handling logic in a block.
  - But only traps errors raised in the executable section of the block.
- SQLCODE and SQLERRM:
  - Oracle suggests that you not use SQLERRM.
- DBMS_UTILITY.FORMAT_CALL_STACK: "how did I get here?"
- DBMS_UTILITY.FORMAT_ERROR_STACK: "what was the error?"
- DBMS_UTILITY.FORMAT_ERROR_BACKTRACE: "where in my code was the error raised?"
- And, new to 12.1, UTL_CALL_STACK

backtrace.sql
bt.pkg

UTL_CALL_STACK (12.1 only)

- This new package provides a granular API to the call stack and back trace.
- It also now provides complete name information, down to the nested subprogram.
- But the bottom line is that generally you will call your functions in the exception section (thru a generic error logging procedure, I hope!) and then write that information to your error log.
- When you need to diagnose an error, go to the log, grab the string, and analyze.

utl_call_stack*.*

Where to use all those functions

- NOT in *each exception handler*.
- Instead, put them inside a generic error logger. Call the logger in each handler, passing application-specific information.
  - Inside the procedure, call all the built-in functions – and throw it all in a table.
- And at what level should you trap those exceptions?
  - Some trap exceptions at top-most block – simpler and easier.
- I suggest trapping exceptions in block where they are raised.
  - It's more work, but then you can log the values of local variables.

FORALL and DML Errors

- FORALLs typically execute a large number of DML statements.
- When an exception occurs in one of those DML statement, the default behavior is:
  - *That* statement is rolled back and the FORALL stops.
  - All (previous) successful statements are *not* rolled back.
- What if you want the FORALL processing to continue, even if an error occurs in one of the statements?
- Just add the EXCEPTIONS clause!

forall_examples.sql

FORALL with SAVE EXCEPTIONS

```
PROCEDURE upd_for_dept (newsal_in IN NUMBER, list_of_emps_in IN pkg.array_type) IS
BEGIN
   FORALL indx IN list_of_emps_in.FIRST .. list_of_emps_in.LAST
      SAVE EXCEPTIONS
      UPDATE employees SET salary = newsal_in
       WHERE employee_id = list_of_emps_in (indx);
END;
```

- FORALL is key feature for improving performance of DML inside loops.
- Add SAVE EXCEPTIONS so that Oracle *saves* exception information and continue processing *all* of the DML statements.
- When the FORALL statement completes, if at least one exception occurred, Oracle then raises ORA-24381.
- You then check the contents of SQL%BULK_EXCEPTIONS.

bulkexc.sql

Row level suppression of errors with LOG ERRORS

- Default behavior: each DML statement is "all or nothing". If an error occurs on any row, changes to all rows are rolled back.
- Add the LOG ERRORS clause to your DML statement to preserve changes to rows.
- Errors are written out to table defined through call to DBMS_ERRLOG.
- Things to keep in mind:
  - No exception is raised when statement finishes!
  - Error log table missing key information, but you can add it!

dbms_errlog.*
dbms_errlog_helper.sql
save_exc_vc_dbms_errlog.sql
cfl_to_bulk7.sql

The AFTER SERVERERROR trigger

- Provides a relatively simple way to use a single table and single procedure for exception handling in an entire instance.
- Drawbacks:
  - Error must go unhandled out of your PL/SQL block for the trigger to kick in.
  - Does not fire for all errors (NO: -600, -1403, -1422...)
- Most useful for non-PL/SQL front ends executing SQL statements directly.

afterservererror.sql

Bullet proof your blocks

- Every block of PL/SQL code has the same structure:
  - Declaration, Execution, Exception
- I suggest that to write high quality, bullet proof code, you should write blocks with a slightly more elaborate "structure".
- Key technique: nested subprograms.

```
DECLARE/IS
   <declarations>

   PROCEDURE initialize
   IS
   BEGIN
      <validate assumptions>
      <other start up code>
   END;

   PROCEDURE cleanup...
BEGIN
   initialize;

   <your code>

   cleanup;
EXCEPTION
   WHEN <error> | OTHERS
   THEN
      <log error>
      cleanup;
END:
```

Validating assumptions with assertion routines

- Every program makes assumptions.
  - "The department ID will never be NULL."
  - "There will always be a row of data for that value."
- When an assumption is violated, programs can break in ways that are difficult to diagnose.
- "No problem!" you might say.
- I'll just write an IF statement:

```
BEGIN
   IF dept_id_in IS NULL
   THEN
      RAISE_APPLICATION_ERROR (
        -20000, 'Dept ID cannot be NULL);
   END IF;

   <"normal" code>
END:
```

Making assertions practical

- We are not going to write all that code.
  - And if we do, we *expose* the mechanism for asserting – best to hide the internal mechanics.
- Instead, use pre-defined assertion routines.
  - Declarative, quick to insert, easier to integrate into error logging, etc.

```
BEGIN
   assert.is_not_null (dept_id_in, 'Dept ID cannot be NULL');

   <"normal" code>
END:
```

assert.pkg

Tracing execution of your code

- Something *is* going to go wrong.
- And if it happens in production, you will most likely find:
  - You can't even believe it's happening. "That shouldn't be possible."
  - You can't reproduce it in dev or test. "Oh well, let us know if it happens again."
- What's a developer to do?
  - Utilize built-in tracing and profiling, but these are not "application aware". So in addition:
- Trace (or *instrument*) your code!
  - You must insert calls to grab application state and save it to a table/file/screen.

Key tips for tracing

- Never put calls to DBMS_OUTPUT.PUT_LINE in your application code.
- Always put calls to trace inside a Boolean expression to minimize overhead in production.
- Do not comment out tracing when you move to production.
- Make it easy to "switch" between sending trace info to table, screen, etc.
- Build in the ability to turn on/off tracing in your app without modifying code.
- Consider using the DBMS_APPLICATION_INFO package, the Logger utility, or build your own.

Bullet Proof PL/SQL – It's not that hard!

- Oracle PL/SQL offers many features to help you manage how errors are both raised and handled.
- The key step for you is to set up guidelines about how these features should be applied.
  - Decide where and when and how you will handle exceptions.
  - Decide standards for execution tracing: parameter values, key algorithmic steps, etc.
  - Build or install error logging and execution tracing utilities.
  - Establish a new standard for block structure. Use nested subprograms to isolate and routinely build initialization, cleanup and assertion "sections".


ORACLE®