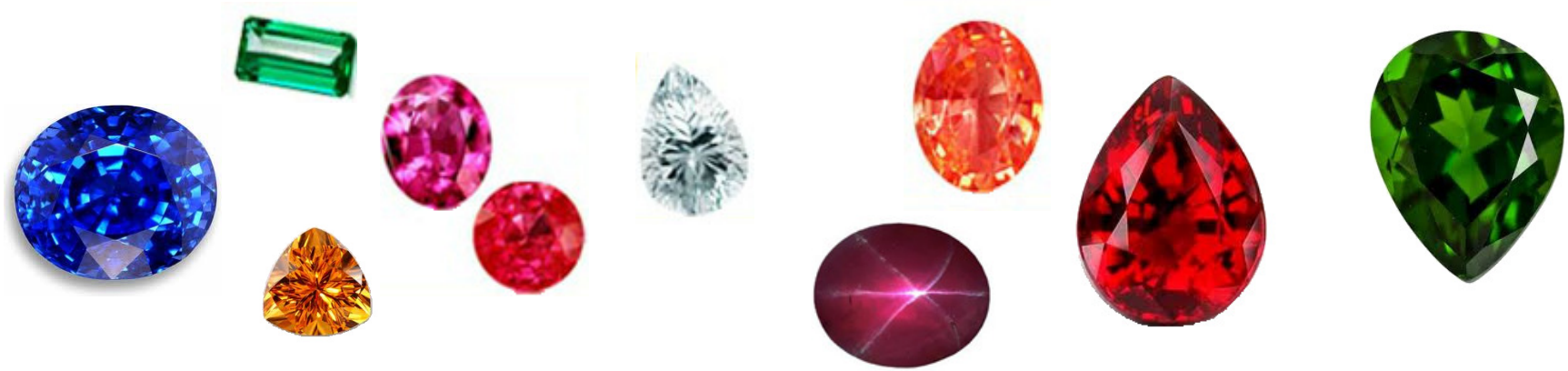
Oracle Gems for SQL, Java and Dot Net Developers



Daniel A. Morgan | damorgan12c@gmail.com | www.morganslibrary.org

Daniel A. Morgan

- Oracle ACE Director
- Consultant to Harvard University
- University of Washington Oracle Instructor, ret.
- The Morgan of Morgan's Library on the web
- Board Member: Western Washington OUG
- Upcoming Presentations
  - Feb 24-15: Rocky Mountain Oracle Users Group
  - Mar 13-14: Utah Oracle Users Group
  - Apr 16-18: Oracle User Group Norway
  - Apr 19-20: Oracle User Group Finland
  - May 13: Oracle User Group Turkey
  - May 15 Oracle User Group Azerbaijan
  - May 19 Bulgarian Oracle User Group

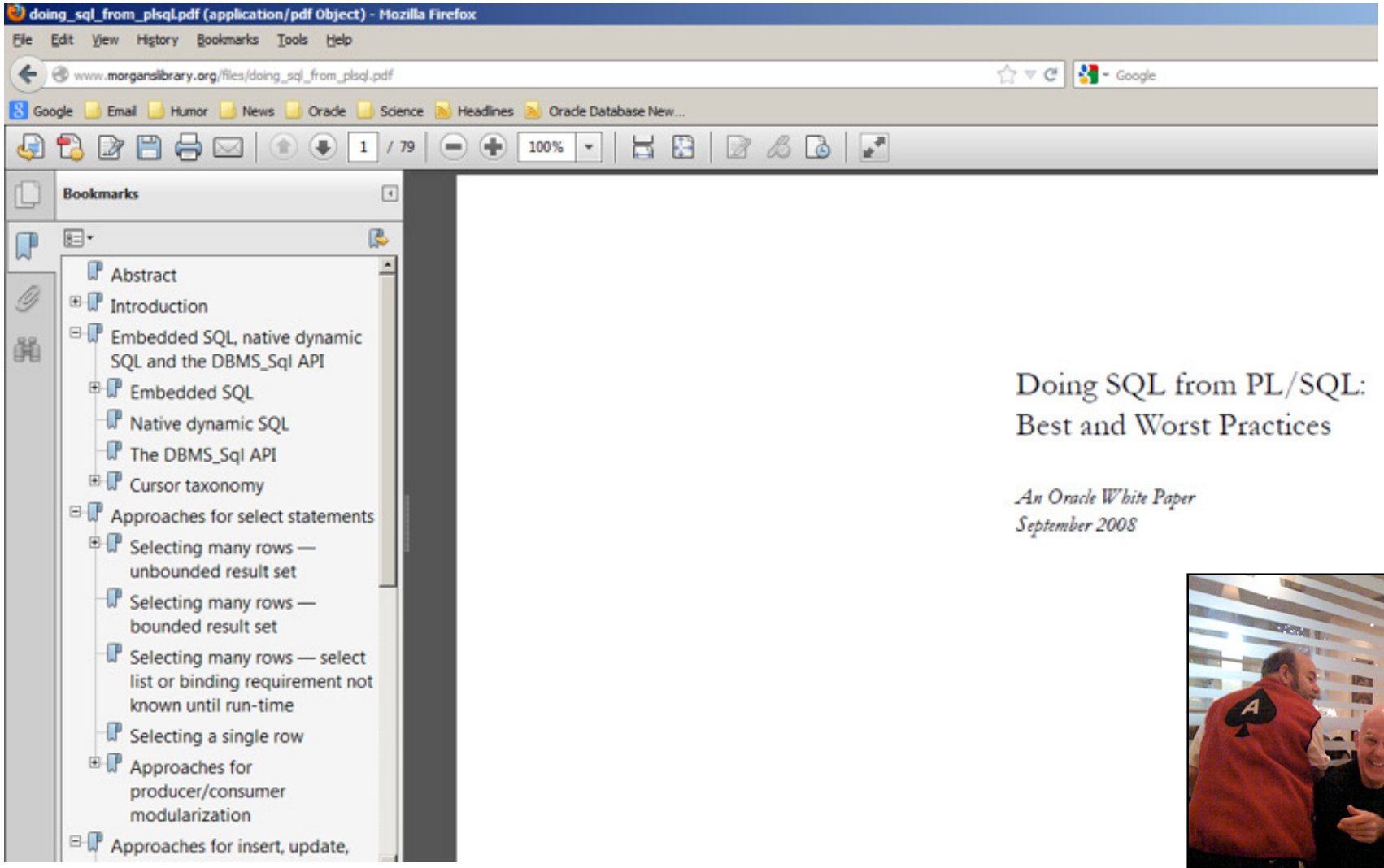
Official Beta Site ORACLE DATABASE 11g

ORACLE RACSIG

International zSeries Oracle SIG

Approach new experiences with care

Bryn Llewellyn's White Paper



Bryn Llewellyn's White Papers

Caveat

Prescribing best practice principles for programming any 3GL is phenomenally difficult. One of the hardest challenges is the safety of the assumption that the reader starts out with these qualities:

- Has chosen the right parents[3].
- Has natural common sense coupled with well-developed verbal reasoning skills.
- Has an ability to visualize mechanical systems.
- Requires excellence from self and others.
- Has first class negotiating skills. (Good code takes longer to write and test than bad code; managers want code delivered in aggressive timeframes.)
- Has received a first class education.
- Can write excellent technical prose. (How else can you write the requirements for your code, write the test specifications, and discuss problems that arise along the way?)

Then, the reader would be fortunate enough to work in an environment which provides intellectual succor:

- Has easy access to one or several excellent mentors.

Bryn Llewellyn's White Papers

Library News

- Morgan's ~~Notepad~~ vi (Blog)
- Join the Western Washington OUG
- Morgan's Oracle Podcast
- DBA Best Practice Guidelines
- Bryn Llewellyn's PL/SQL White Paper
- Bryn Llewellyn's Editioning White Paper
- Troubleshooting Performance

Doing SQL from PL/SQL: Best and Worst Practices

An Oracle White Paper
September 2008

PL/SQL-specific syntax.

```
-- Code 3
<<b>>declare
  Some_Value t.PK%type := 42;
  The_Result t%rowtype;
begin
  select    a.*
  into      b.The_Result
  from      t a
  where     a.PK = b.Some_Value;

  The_Result.v1 := 'New text';

  update  t a
  set     row = b.The_Result
  where   a.PK = b.The_Result.PK;

  The_Result.PK := -Some_Value;
  insert  into t
  values  The_Result;
end;
```

Resolution of names in embedded SQL statements.

SQL

No Objects Named with Keywords

- Oracle provides an list of all keywords. Words that should never be used in naming objects or columns. Following is an example of how to test a word to determine whether it can be used

```
SELECT keyword
FROM v$reserved_words
WHERE keyword LIKE 'ID%';

KEYWORD
------------------------------
ID
IDENTIFIED
IDGENERATORS
IDLE_TIME
IDENTITY
IDENTIFIER

6 rows selected.
```

Column must never be named "ID." Best practice is to identify the nature of the ID, for example, MEMBER_ID or ORDER_ID. This holds true for columns such as COMMENTS and DATES.

Not Null Constraints

- Many operations in Oracle are optimized when Oracle knows that the columns involved are not nullable. Always create tables with NOT NULL columns whenever possible but do not name Primary Key columns as NOT NULL: this creates additional overhead .

```
CREATE TABLE t (

order_id NUMBER,
order_date DATE) NOT NULL;


ALTER TABLE t
ADD CONSTRAINT pk_t
PRIMARY KEY;
```

Implicit Casts

- Code that does not correctly define data types will either fail to run or run very inefficiently

  The following example shows both the correct way and the incorrect way to work with dates. The correct way is to perform an explicit cast.

```
orabase> create table t (
  2  datecol date);

Table created.

orabase> insert into t values ('01-JAN-2012');

1 row created.

orabase> insert into t values (TO_DATE('01-JAN-2012'));

1 row created.
```

In Oracle dates are dates ... not strings. Similarly numbers should either be explicitly cast with TO_NUMBER or only processed using numeric variables.

Statement Labeling

- When SQL statements are labeled it makes it very easy for the DBA team to trace activity and help with performance tuning and debugging.
- The following example shows how easy this is. In this example "mypackage.myprocedure" is the name of the PL/SQL package and procedure into which the statements are embedded.

```
SELECT /* mypackage.myprocedure */ COUNT(*)
FROM all_tables;

INSERT /* mypackage.myprocedure */ INTO servers
(srvr_id)
VALUES
(11111);

UPDATE /* mypackage.myprocedure */ servers
SET srvr_id = 11111
WHERE srvr_id = 1;

DELETE /* mypackage.myprocedure */ FROM servers
WHERE srvr_id = 11111;
```

PL/SQL

DDL

- DDL should never be written as part of the operational code of a database application. It is especially hard on the system when utilizing Real Application Clusters (RAC) which is the standard platform for e-Commerce. The following statements are DDL.
  - ALTER
  - CREATE
  - DROP
  - GRANT
  - REVOKE
  - TRUNCATE
- If you think you require the use of one of these statement work with the DBA team to identify an alternative strategy.

Specify Constants as Constants

- Oracle has code paths that make for more efficient processing when CONSTANTS are specifically identified as shown below. When specifying a constant you must assign a value at the time it is declared as it must always have a single value.

```
CREATE OR REPLACE FUNCTION calc_area(pRadius IN NUMBER) RETURN NUMBER
AUTHID DEFINER IS
 pi    CONSTANT NUMBER(8,7) := 3.1415926;
 area           NUMBER;
BEGIN
  area := pi * POWER(pRadius, 2);
  RETURN area;
END calc_area;
/
```

Flag Data Type

- If you are using a variable solely for the purpose of acting as a flag use the SIGNTYPE data type which can only hold one of three possible values: -1, 0, or 1

```
DECLARE
 myFlag  SIGNTYPE := 0;
 myCount PLS_INTEGER;
BEGIN
  SELECT COUNT(*)
  INTO myCount
  FROM user_objects;

  IF myCount <> 0 THEN
    myFlag := 1;   -- indicates success
  ELSE
    myFlag := -1;  -- indicates failure
  END IF;
END;
/
```

SIGNTYPE is a PL/SQL data type and can not be used to define a table or view column.

... create a data type that will only hold 0s and 1s

```
DECLARE
 SUBTYPE flagtype IS PLS_INTEGER RANGE 0..1;
 x flagtype;
BEGIN
  BEGIN
    x := 0;
    dbms_output.put_line('Success: ' || TO_CHAR(x));
  EXCEPTION
    WHEN others THEN
      dbms_output.put_line('Can not assign 0 to flagtype');
  END;

  BEGIN
    x := 1;
    dbms_output.put_line('Success: ' || TO_CHAR(x));
  EXCEPTION
    WHEN others THEN
      dbms_output.put_line('Can not assign 1 to flagtype');
  END;
  ....
END;
/
```

Cursor Loops

- Cursor loops in the forms demonstrated below have been obsolete in Oracle for more than 14 years. If you have an issue where you are tempted to use one instead use the more efficient BULK COLLECT and FORALL syntax unless the number of rows will never exceed 100.

```
OPEN mycursor;
LOOP
  FETCH mycursor INTO myrecord;
  EXIT WHEN mycursor%NOTFOUND;
  ...
END LOOP;
CLOSE mycursor;


FOR myrecord IN mycursor LOOP
  ...
END LOOP;


FOR myrecord IN (SELECT * FROM mytable) LOOP
  ...
END LOOP;
```

NOCOPY Compiler Hint

- If your PL/SQL procedure or function returns a value using an OUT or IN OUT parameter consider use of the NOCOPY compiler hint if the OUT parameter is a CLOB, BLOB, XML, REF CURSOR, or a non-scalar data type. The following examples show how to do this.

```
CREATE OR REPLACE PROCEDURE nocopy_out(retval OUT NOCOPY VARCHAR2)
AUTHID DEFINER AS
BEGIN
  retval := dbms_crypto.randomBytes(32);
END nocopy_out;
/


CREATE OR REPLACE PROCEDURE noocpy_io(x IN OUT NOCOPY NUMBER)
AUTHID DEFINER AS
BEGIN
  x := x + dbms_crypto.randomInteger;
END;
/
```

AUTHID

- All PL/SQL objects, functions, package headers, procedures, and types default to AUTHID DEFINER
- The alternative, AUTHID CURRENT_USER is more restrictive and provides better protection for the database
- Always explicitly define the AUTHID value when creating objects

```
CREATE OR REPLACE FUNCTION how_many_tables AUTHID DEFINER RETURN NUMBER IS
 i NUMBER;
BEGIN
  SELECT COUNT(*)
  INTO i
  FROM user_all_tables;
  RETURN i
END is_number;
/

CREATE OR REPLACE FUNCTION how_many_tables AUTHID CURRENT_USER RETURN
NUMBER IS
 i NUMBER;
BEGIN
  SELECT COUNT(*)
  INTO i
  FROM user_all_tables;
  RETURN i
END is_number;
/
```

Predefined Inquiry Directives

- Version 11.2.0.3
  - $$plsql_line
  - $$plsql_unit
- Version 12.1.0.1
  - $$plsql_unit_owner
  - $$plsql_unit_type
  - $$plsql_ccflags
  - $$plsql_code_type
  - $$plsql_optimize_level
  - $$plsql_warnings
  - $$plsql_nls_length_semantics
  - $$plscope_settings

```
CREATE OR REPLACE PROCEDURE test AUTHID DEFINER IS
 BEGIN
   dbms_output.put_line('I am ' || $$plsql_unit);
 END test;
 /

exec test;
```

Pragma Inline

- Very strongly encourages the PL/SQL compiler to replace the subprogram call with the program unit itself

```
DECLARE
  l_loops  NUMBER := 10000000;
  l_start  NUMBER;
  l_return NUMBER;

  FUNCTION add_numbers(p_1 IN NUMBER, p_2 IN NUMBER) RETURN NUMBER AS
  BEGIN
    RETURN p_1 + p_2;
  END add_numbers;
 BEGIN
   l_start := dbms_utility.get_time;

   FOR i IN 1 .. l_loops LOOP
     PRAGMA INLINE(add_numbers, 'YES');
     l_return := add_numbers(1, i);
   END LOOP;

   dbms_output.put_line('Elapsed Time: ' || (dbms_utility.get_time -
l_start) || ' hsecs');
 END;
 /
```

Instead Of Triggers

- A trigger on a view that defines the action that should be taken INSTEAD OF the default action

```
CREATE OR REPLACE TRIGGER abcd
INSTEAD OF INSERT
ON payments
FOR EACH ROW
BEGIN
  INSERT /* IOTV1 */ INTO credit_cards
  (credit_card_id, credit_card_type_id, billing_addr_id, card_number_hint,
   expiration_month, expiration_year, cardholder_name, credit_card_token,
   member_id, credit_card_number, clear_number, deleted, create_date,
   update_date, flags, status, credit_card_bin)
  VALUES
  (:NEW.credit_card_id, :NEW.credit_card_type_id, :NEW.billing_addr_id,
   :NEW.card_number_hint,:NEW.expiration_month, :NEW.expiration_year,
   :NEW.cardholder_name, :NEW.credit_card_token, :NEW.member_id,
    CASE WHEN (:NEW.credit_card_type_id IN (1, 2, 5, 6, 8)) THEN
      NEW.credit_card_number
    ELSE '???' END,
   :NEW.clear_number, :NEW.deleted, :NEW.create_date,
   :NEW.update_date, :NEW.flags, :NEW.status, :NEW.credit_card_bin);

  IF :NEW.credit_card_type_id NOT IN (3, 4, 7) THEN
    pay_pkg.process_cc(:NEW.credit_card_id, NEW.credit_card_number);
  END IF;
END abcc;
/
```

PL/SQL Warnings

- 11g New Warnings
- Invaluable for finding suboptimal code, use of reserved words, inappropriate usages, and orphans
  - Severe
    - 5018 - omitted optional AUTHID clause
    - 5019 - deprecated language element
    - 5020 - parameter name must be identified
  - Informative
    - 6016 - native code generation turned off (size/time)
    - 6017 - operation will raise an exception
    - 6018 - an infinity or NaN value computed or used
  - Performance
    - None

Explicit Commits & Rollbacks

- An explicit commit must be present at the end of any transaction. You can not rely on ODBC drive configuration to commit your changes in a reliable and consistent way.
- Similarly ... you must explicitly rollback in exception handlers as the ODBC driver may perform an unintended commit.

```
CREATE OR REPLACE PROCEDURE test AUTHID DEFINER IS
BEGIN
  INSERT INTO t
  (testcolumn)
  VALUES
  (1);

  COMMIT;
EXCEPTION
  WHEN dup_val_on_index THEN
    ROLLBACK;
END;
/
```

SQL Injection

- All code containing dynamic SQL must use bind variables
- All code containing dynamic SQL where schema, object, or column names are dynamic must utilize calls to DBMS_ASSERT.ENQUOTE_NAME to prevent SQL Injection attack

```
set serveroutput on

DECLARE
 x NUMBER(10);
BEGIN
  FOR i IN 1 .. 50000
  LOOP
    EXECUTE IMMEDIATE 'SELECT ' || i || ' INTO :b1 FROM dual'
    INTO x;
  END LOOP;
END;
/

DECLARE
 tname user_tables.table_name%TYPE := 'SERVERS';
BEGIN
  dbms_output.put_line(tname);
  tname := dbms_assert.enquote_name(tname);
  dbms_output.put_line(table_name);
END;
/
```

Setting Client Info

- CLIENT_INFO is a way to poke into database instance memory the name of the application that is being run. In the following example, based on a PL/SQL package, the application is named "MEMBER_MGMT."

```
CREATE OR REPLACE PACKAGE webstore.member_mgmt AUTHID DEFINER IS
  PROCEDURE myproc;
END member_mgmt;
/


CREATE OR REPLACE PACKAGE BODY webstore.member_mgmt IS
 PROCEDURE myproc IS
 BEGIN
   dbms_application_info.set_client_info('WEBSTORE.MEMBER_MGMT');
   ... your code here
   dbms_application_info.set_client_info('-');
 EXCEPTION
   WHEN OTHERS THEN
     dbms_application_info.set_client_info('-');
 END myproc;
END member_mgmt;
/
```

Set Module and Action

- MODULE is a way to poke into database instance memory the name of the procedure or function that is being run while ACTION can be used to specify the specific activity being performed which, again, aids in debugging and tuning.

```
CREATE OR REPLACE PACKAGE webstore.member_mgmt AUTHID DEFINER IS
  PROCEDURE myproc;
END member_mgmt;
/

CREATE OR REPLACE PACKAGE BODY webstore.member_mgmt IS
 PROCEDURE myproc IS
 BEGIN
   dbms_application_info.set_client_info('WEBSTORE.MEMBER_MGMT');
   dbms_application_info.set_module('MYPROC', 'Loading Data');
   ... load data here
   dbms_application_info.set_module('MYPROC', 'Verifying Data Quality');
   ... verify data quality here
   dbms_application_info.set_module(NULL, NULL);
   dbms_application_info.set_client_info('-');
 END myproc;
END member_mgmt;
/
```

Exception Handling

Exception Handling

- For Oracle database exceptions should consist of a single table, per application, that collects, at a minimum, the following information
  - database name
  - schema name
  - instance number
  - the name of the connected user or application user
  - the code module executing
  - the line of code that failed
  - the Oracle exception number and message
- For application-based exceptions, exceptions that are generated by the application, for example “no credit card on file,” the following information collected must include
  - database name
  - schema name
  - the name of the connected user or application user
  - an identifier for the specific transaction that generated the exception
  - the code module executing (Dot Net or Database)
  - the line of code that generated the exception

Exception Handling

- The following example is a typical 11g exception capture table.

```
CREATE TABLE util_event_log (
instance_id       NUMBER(2),
database_name     VARCHAR2(9),
schema_name       VARCHAR2(30),
package_name      VARCHAR2(30),
object_name       VARCHAR2(30),
object_type       VARCHAR2(19),
line_number       NUMBER,
beg_date          TIMESTAMP(6),
end_date          TIMESTAMP(6),
host_name         VARCHAR2(64),
instance_name     VARCHAR2(16),
active_instances  NUMBER(3),
service_name      VARCHAR2(30),
module_name       VARCHAR2(48),
action_name       VARCHAR2(32),
client_info       VARCHAR2(64),
severity          NUMBER(2),
sql_errno         NUMBER(5),
event_text        VARCHAR2(256),
log_comment       VARCHAR2(256))
PARTITION BY RANGE (instance_id)
INTERVAL (1) (
PARTITION root_par VALUES LESS THAN (2));
```

Exception Handling

- The following examples how to capture calling module information including name and line number. More robust handling is possible with FORMAT_ERROR_STACK and FORMAT_CALL_STACK, and FORMAT_ERROR_BACKTRACE.

```
CREATE OR REPLACE PROCEDURE child AUTHID DEFINER IS
 oname all_objects.owner%TYPE;         -- the source schema name
 pname all_objects.object_name%TYPE;   -- the source object name
 lnumb all_source.line%TYPE;           -- the source line number
 callr all_objects.object_type%TYPE;   -- the source object type
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  owa_util.who_called_me(oname, pname, lnumb, callr);
  dbms_output.put_line(oname);
  dbms_output.put_line(pname);
  dbms_output.put_line(lnumb);
  dbms_output.put_line(callr);
END;
/

CREATE OR REPLACE PROCEDURE parent_p AUTHID DEFINER IS
BEGIN
  child;
END parent_p;
/

exec parent_p
```

Exception Handling

- The following code shows a typical generic exception handler

```
PROCEDURE Log_Event_Start(
 pBegDate    IN util_event_log.beg_date%TYPE DEFAULT SYSDATE,
 pSeverity   IN util_event_log.severity%TYPE DEFAULT NULL,
 pLogComment IN util_event_log.log_comment%TYPE) AUTHID DEFINER IS
 cDBName   CONSTANT util_event.log.database_name%TYPE := sys_context('USERENV', 'DB_NAME;);
 cInstID   CONSTANT util_event_log.instance_id%TYPE := sys_context('USERENV', 'INSTANCE');
 cInstName CONSTANT util_event_log.instance_name%TYPE := sys_context('USERENV', 'INSTANCE_NAME');
 cHostName CONSTANT util_event_log.host_name%TYPE := sys_context('USERENV', 'SERVER_HOST');
 cModName  CONSTANT util_event_log.module_name%TYPE := sys_context('USERENV', 'MODULE');
 cActName  CONSTANT util_event_log.action_name%TYPE := sys_context('USERENV', 'ACTION');
 cCliInfo  CONSTANT util_event_log.client_info%TYPE := sys_context('USERENV', 'CLIENT_INFO');
 cServName CONSTANT util_event_log.service_name%TYPE := sys_context('USERENV', 'SERVICE_NAME');
 vInstCount  util_event_log.active_instances%TYPE := 1;
 vInstTab    dbms_utility.instance_table;
 vLineNumber all_source.line%TYPE;
 vObjType    all_objects.object_type%TYPE;
 vPkgName    all_objects.object_name%TYPE;
 vSchemaName all_objects.owner%TYPE;
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  owa_util.who_called_me(vSchemaName, vPkgName, vLineNumber, vObjType);

  IF dbms_utility.is_cluster_database THEN
    dbms_utility.active_instances(vInstTab, vInstCount);
  END IF;

  INSERT /* mlib_utils.change_config05 */ INTO util_event_log
  (instance_id, database_name, schema_name, package_name, object_type, line_number,
   beg_date, host_name, instance_name, active_instances, service_name,
   module_name, action_name, client_info, severity, log_comment)
  VALUES
  (cInstID, cDBName, vSchemaName, vPkgName, vObjType, vLineNumber,
   pBegDate, cHostName, cInstName, vInstCount, cServName,
   cModName, cActName, cCliInfo, pSeverity, pLogComment);

  COMMIT;
  -- this procedure intentionally does not contain exception handling: do not add one.
END Log_Event_Start;
/
```

Other Built-ins

Packages and Functions

- SYS_CONTEXT function
- DBMS_ASSERT
- DBMS_CHANGE_NOTIFICATION
- DBMS_COMPARISON
- DBMS_ERRLOG
- DBMS_METADATA_DIFF
- DBMS_PCLXUTIL
- DBMS_SPACE

Questions

ERROR at line 1:
ORA-00028: your session has been killed

Thank you