Oracle PL/SQL: Transactions , Dynamic SQL & Debugging

Introduction

Pankaj Jain

@twit_pankajj

pluralsight
hardcore dev and IT training

Content

DBMS_SQL

Debugging

Transaction Management

Autonomous Transactions

Native Dynamic SQL

Pre-requisites

Oracle PL/SQL Fundamentals - Part 1

Oracle PL/SQL Fundamentals - Part 2

Equivalent Programming Knowledge

Audience

Oracle Database Programmers

Web Developers

Other Programmers

Tools

- Oracle Express Edition
- SQL Developer
- SQLPLUS
- Toad
- SQL Navigator

Transaction Management in PL/SQL

Pankaj Jain

@twit_pankajj

pluralsight
hardcore dev and IT training

ACID

Atomicity
All or Nothing

Consistency
Transition From One Valid State to Another

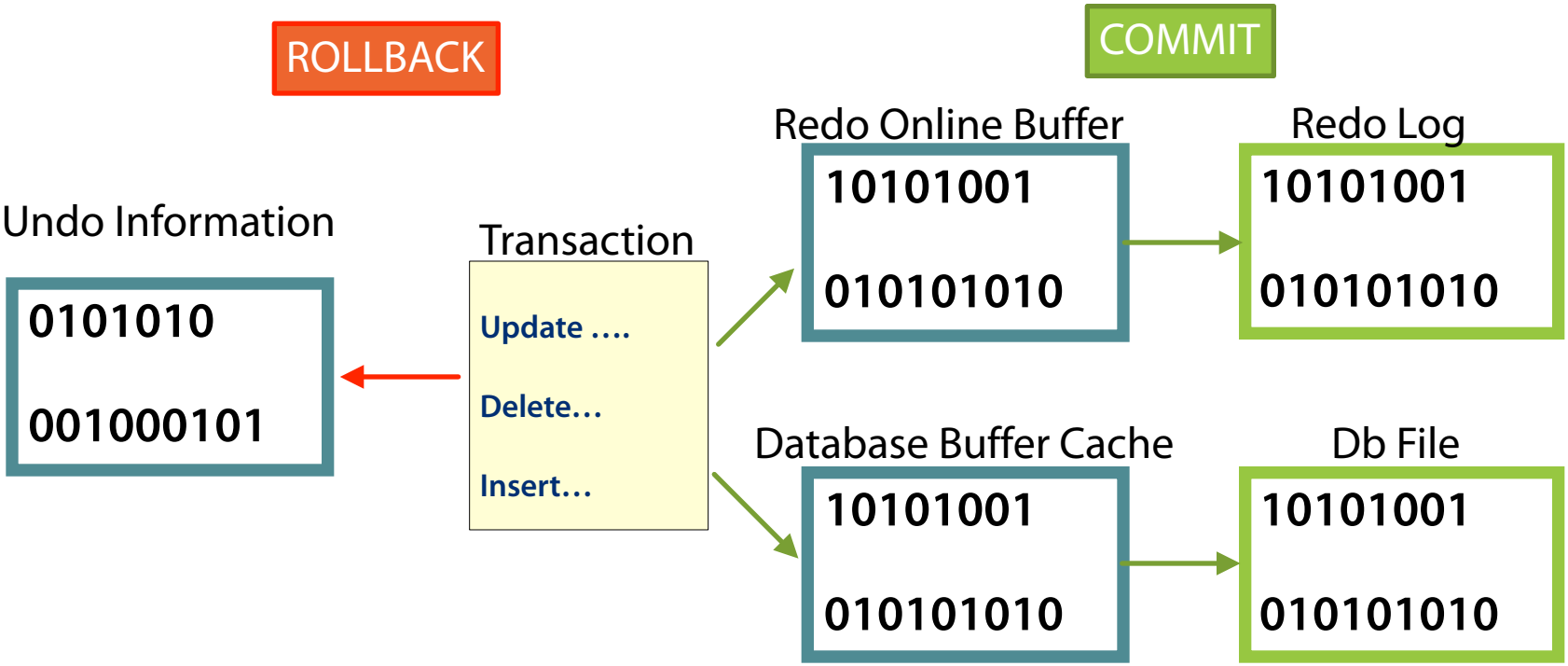
Isolation
Concurrent Transactions Leading to Same State as Serial Execution

Durability
Commit Makes Changes Permanent

What Is a Transaction?

- **Atomic Unit of Work**
- **Begins With**
  - DML Statement
  - DDL Statement
  - Set Transaction Statement
- **Ends With**
  - Commit Statement
  - Rollback Statement
  - DDL Statement
  - Normal User Exit / Abnormal Termination

Transaction Management in Oracle



Transaction

```
CREATE OR REPLACE PROCEDURE
                    process_order (p_act_id accounts.act_id%TYPE,
                                   p_item_id items.item_id%TYPE,
                                   p_item_value items.item_value%TYPE) IS
 BEGIN
  -- Debit Account
  UPDATE accounts
   SET act_bal = act_bal - p_item_value
   WHERE act_id = p_act_id;
 -- Place Order
   INSERT INTO orders(order_id,
                      order_item_id,
                      order_act_id)
              VALUES( order_seq.NEXTVAL,
                      p_item_id,
                      p_act_id);
   COMMIT;
 EXCEPTION
 WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
  DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
  ROLLBACK;
  RAISE;
END process_order;
```

Transaction Begins

Transaction Ends

Transaction Ends

Transaction Names

```
SET TRANSACTION NAME 'txn_name';
```

- User Specified Name
- First Statement of a Transaction
- Optional
- Useful for Monitoring Long Running Transactions
- Redo Logs
- V$TRANSACTION

Transaction Names

```
CREATE OR REPLACE PROCEDURE
                    process_order (p_act_id accounts.act_id%TYPE,
                                   p_item_id items.item_id%TYPE,
                                   p_item_value items.item_value%TYPE) IS
 BEGIN
   SET TRANSACTION NAME 'place_order';
  -- Debit Account
  UPDATE accounts …..
 -- Place Order
   INSERT INTO orders(order_id, …..
   COMMIT;
 EXCEPTION
 ….
  ROLLBACK;
  RAISE;
END process_order;
```

```
EXEC process_order(1, 2, 500);
```

```
SELECT name, status FROM V$TRANSACTION;

name            status
————————————————-
place_order     ACTIVE
```

Transaction Names

```
SET TRANSACTION NAME 'mytxn';

INSERT INTO demo.items(item_id, item_name,item_value) VALUES (2, 'Treadmill', 500);

SELECT name, status FROM V$TRANSACTION;

name              status
————————————————-
mytxn            ACTIVE
```

Read Only Transactions

```
SET TRANSACTION READ ONLY NAME  'txn_name';
```

- Read From the Same Snapshot
- First Statement
- Ends With Commit or Rollback

```
BEGIN
   SET TRANSACTION READ ONLY  'Current Month Order Count';
  -- Select customer id for a given customer name
     SELECT customer_name FROM customers WHERE  …..
  -- Find account id for the customer
     SELECT acct_id FROM accounts WHERE …
  -- Find count of orders for the current month for the customer
    SELECT COUNT(order_id) FROM orders WHERE …
  -- End Transaction
    COMMIT;
END
```

Commit

Changes Made Permanent

System Change Number (SCN) is Generated

Locks are Released

Savepoints are Deleted

Commit

- **Uncommitted Changes are Not Visible to Other Sessions**

```
DECLARE
l_item_name items.item_name%TYPE;
BEGIN
INSERT INTO items(item_id, item_name,item_value)
                VALUES (3, 'Treadmill', 400);
SELECT item_name INTO l_item_name
FROM items
WHERE item_id = 3;

DBMS_OUTPUT.PUT_LINE(l_item_name);

COMMIT;

END;
```

Treadmill

```
SELECT item_name
FROM items
WHERE item_id = 3;

item_name
................
```

```
SELECT item_name
FROM items
WHERE item_id = 3;

item_name
................
Treadmill
```

Commit

- DDL Statements Issue an Implicit Commit

```
CREATE OR REPLACE PROCEDURE
                    process_order (p_act_id accounts.act_id%TYPE,
                                   p_item_id items.item_id%TYPE,
                                   p_item_value items.item_value%TYPE) IS
 BEGIN
  -- Debit Account
  UPDATE accounts
   SET act_bal = act_bal - p_item_value
   WHERE act_id = p_act_id;

    EXECUTE IMMEDIATE 'DROP TABLE TEMP_TABLE' ;



-- Place Order
   INSERT INTO orders(order_id,
                      order_item_id,
                      order_act_id)
…..
   COMMIT;
 EXCEPTION
 WHEN OTHERS THEN …
  ROLLBACK;
  RAISE;
END process_order;
```

Transaction Begins

Transaction Ends / New Transaction Begins/ New Transaction Ends

Transaction Begins

Transaction Ends

Transaction Ends

User Exit

- SQLDeveloper
- Sqlplus

Rollback

Changes Undone

Ends Transaction

Releases Locks & Resources

Erases Savepoints

Rollback

```
CREATE OR REPLACE PROCEDURE
                    process_order (p_act_id accounts.act_id%TYPE,
                                   p_item_id items.item_id%TYPE,
                                   p_item_value items.item_value%TYPE) IS
 BEGIN
  -- Debit Account
  UPDATE accounts
   SET act_bal = act_bal - p_item_value
   WHERE act_id = p_act_id;
 -- Place Order
   INSERT INTO orders(order_id,
                      order_item_id,
                      order_act_id)
              VALUES( order_seq.NEXTVAL,
                      p_item_id,
                      p_act_id);
   COMMIT;
 EXCEPTION
 WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
  DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
  ROLLBACK;
  RAISE;
END process_order;
```

Transaction Begins

Transaction Ends

Transaction Ends

Statement Level Atomicity

- Statement is an Atomic Unit of Work
- Statement Error Causes Only Work Done by the Statement to Be Rolled Back

Statement Level Atomicity

```
CREATE OR REPLACE PROCEDURE
                    process_order (p_act_id accounts.act_id%TYPE,
                                   p_item_id items.item_id%TYPE,
                                   p_item_value items.item_value%TYPE) IS
 BEGIN
  -- Debit Account
  UPDATE accounts
   SET act_bal = act_bal - p_item_value
   WHERE act_id = p_act_id;
 -- Place Order
  INSERT INTO orders(order_id,
                     order_item_id,
                     order_act_id)
            VALUES( order_seq.NEXTVAL,
                     p_item_id,
                     p_act_id);
  COMMIT;
 EXCEPTION
 WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
  DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
  COMMIT;
  RAISE;
END process_order;
```

Transaction Begins

Transaction Ends

Other Rollback Considerations

- Abnormal Termination
- Package Variable Changes

```
...
 BEGIN
  -- Debit Account
  UPDATE accounts
   SET act_bal = act_bal - p_item_value
   WHERE act_id = p_act_id;
    acct_mgmt.g_debit := 'Y';
 -- Place Order
   INSERT INTO orders(order_id,
                      order_item_id,
                      order_act_id)
            VALUES( order_seq.NEXTVAL,
                     p_item_id,
                    p_act_id);
   COMMIT;
 EXCEPTION
 WHEN OTHERS THEN
   .....
   ROLLBACK;
   DBMS_OUTPUT.PUT_LINE(acct_mgmt.g_debit);
  RAISE;
END process_order;
```

Y

Savepoint

```
SAVEPOINT 'savepoint_name';
```

User Defined Marker

Logical Break

Multiple

Partial Rollback

Savepoint

```
BEGIN
 -- Debit Account
 UPDATE accounts   SET act_bal = act_bal - 500  WHERE act_id = 1;
 -- Place Order
  INSERT INTO orders(order_id, order_item_id, order_act_id)
                 VALUES( order_seq.NEXTVAL, 2, 1);

  SAVEPOINT savepoint_after_first_order;

 -- Debit Account
 UPDATE accounts   SET act_bal = act_bal - 600  WHERE act_id = 2;
 -- Place Order
  INSERT INTO orders(order_id, order_item_id, order_act_id)
                 VALUES( order_seq.NEXTVAL, 1, 2);

 COMMIT;
 EXCEPTION
 WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
   DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
   ROLLBACK to savepoint_after_first_order;
   COMMIT;
   RAISE;
END ;
```

Multiple Savepoints

```
BEGIN
   INSERT INTO items ….
    …..
   SAVEPOINT first;

   UPDATE accounts …
    …..
    SAVEPOINT second;

   INSERT INTO orders…
    ……
    SAVEPOINT third;

   IF condition=TRUE THEN
      ROLLBACK TO second;
   END IF;
….
```

Multiple Savepoints

```
BEGIN
   INSERT INTO items ….
    …..
   SAVEPOINT first;

   UPDATE accounts …
    …..
    SAVEPOINT second;

   INSERT INTO orders…
    ……
    SAVEPOINT third;

   IF condition=TRUE THEN
      ROLLBACK ;
   END IF;
….
```

Savepoint Overriding

```
…
BEGIN
   FOR order_var IN cur_get_order_queue LOOP

    SAVEPOINT savepoint_before_order;

    BEGIN
      -- Debit Account
      UPDATE accounts   SET act_bal = act_bal - order_var.item_value  WHERE act_id = order_var.act_id;

      -- Place Order
      INSERT INTO orders(order_id, order_item_id, order_act_id)
                        VALUES( order_seq.NEXTVAL, order_var.item_id, order_var.act_id);

      COMMIT;
    EXCEPTION
       WHEN OTHERS THEN
           ROLLBACK TO savepoint_before_order;
          log_error(order_var.act_id,order_var.item_id, SQLERRM);
          COMMIT;
    END;

 END LOOP;
END ;
```

Explicit Locks

Cursor FOR UPDATE

Lock Table

Cursor FOR UPDATE

```
DECLARE
 CURSOR cur_upd_acts (p_bal accounts.act_bal%TYPE) IS
   SELECT act_id,
                act_bal
     FROM accounts
  WHERE act_bal < p_bal
  FOR UPDATE OF act_bal;
BEGIN
  FOR cur_upd_acts_var IN cur_upd_acts(500) LOOP
          ....
          .....
          UPDATE accounts
             SET act_bal = act_bal - 10
           WHERE CURRENT OF cur_upd_acts;
 END LOOP;
 COMMIT;
END;
```

Lock Table

Shared Lock

Exclusive Lock

```
LOCK TABLE accounts IN ROW SHARE MODE NOWAIT;

LOCK TABLE accounts IN ROW EXCLUSIVE MODE;
```

Summary

Transaction Management Overview

Commit

Rollback

Savepoint

Native Dynamic SQL

Pankaj Jain

@twit_pankajj

pluralsight
hardcore dev and IT training

What is Dynamic SQL?

SQL Statement Known at Runtime

Static SQL

```
CREATE OR REPLACE FUNCTION
get_count(p_act_id accounts.act_id%TYPE)
RETURN NUMBER IS
   l_count NUMBER;
 BEGIN
   SELECT COUNT(*)
      INTO l_count
   FROM   orders
  WHERE   order_act_id =  p_act_id;
  RETURN l_count;
END get_count;
```

Dynamic SQL

```
CREATE OR REPLACE FUNCTION get_count(
                p_where VARCHAR2)  RETURN NUMBER
IS
   l_count NUMBER;
   l_select VARCHAR2(100) := 'SELECT COUNT(*) ';
   l_from   VARCHAR2(60) := 'FROM orders  ';
   l_query VARCHAR2(200);
 BEGIN
   l_query := l_select ||
                  l_from ||
                  p_where;
  EXECUTE IMMEDIATE l_query INTO l_count;
  RETURN l_count;
END get_count;
```

Static vs Dynamic SQL

Static SQL	Dynamic SQL
SQL Known at Compile Time	SQL Known at Runtime
Compiler Verifies Object References	Compiler Cannot Verify Object References
Compiler Can Verify Privileges	Compiler Cannot Verify Privileges
Less Flexible	More Flexible
Faster	Slower

Common Uses

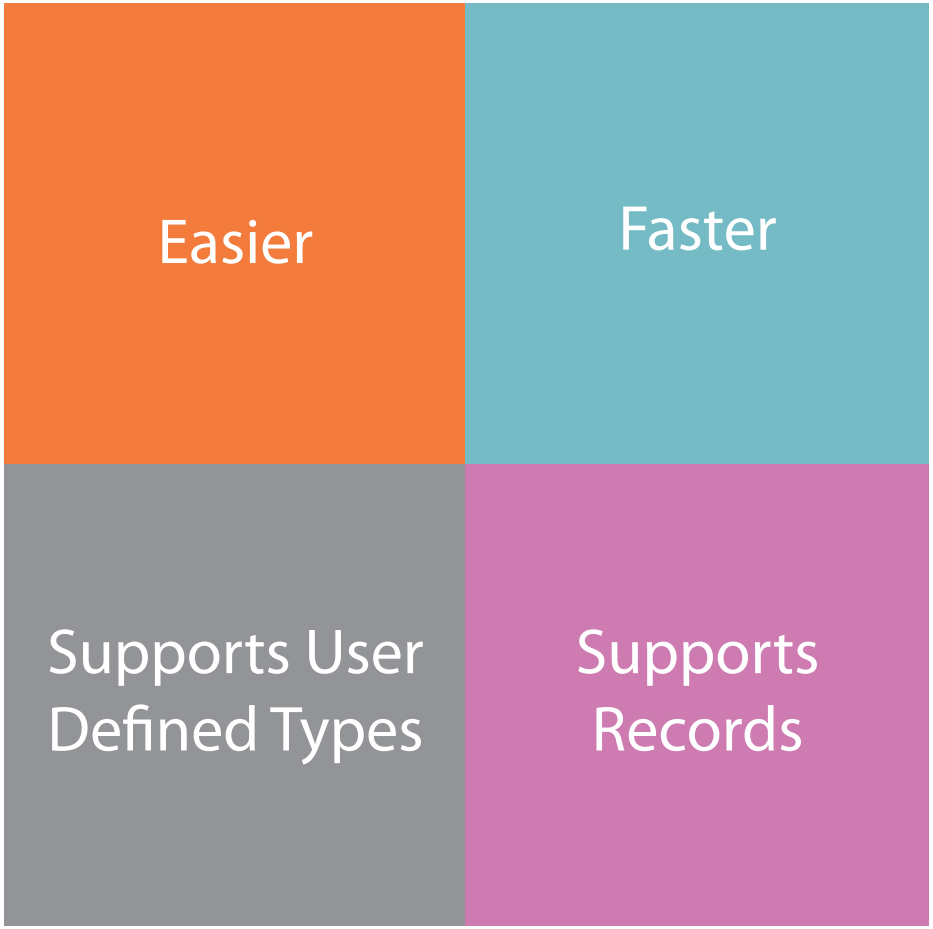
- Dynamic Queries
- Dynamic Sorts
- Dynamic Subprogram Invocation
- Dynamic Optimizations
- DDL
- Frameworks
- Varying Table Definitions

Invoking Dynamic SQL

Native Dynamic SQL

DBMS_SQL

Why Use Native Dynamic SQL?



Native Dynamic SQL

```
EXECUTE IMMEDIATE <dynamic_sql_string>

[INTO {select_var1[, select_var2].. | record }

[USING [ IN | OUT | IN OUT ] bind_var1
[ ,        [ IN | OUT | IN OUT ] bind_var2]…]

[{RETURNING | RETURN} INTO bind_var1
[,                                  bind_var2]...];
```

- Ref Cursors

DDL Operations

- Create Objects

```
CREATE OR REPLACE   PROCEDURE  create_table (p_table_name      VARCHAR2,
                                                p_table_columns  VARCHAR2)  IS
BEGIN
    EXECUTE IMMEDIATE  'CREATE TABLE '|| p_table_name || p_table_columns;
END create_table;
```

```
EXEC create_table(
                  'ORDERS_QUEUE_CA',
                  '(queue_id NUMBER,queue_act_id NUMBER,queue_item_id NUMBER)'
                  );
```

Object Privileges in Native Dynamic SQL

- **Direct Grants**

User demo

```
CREATE OR REPLACE   PROCEDURE  create_table_procedure(p_table_name      VARCHAR2,
                                                        p_table_columns  VARCHAR2)  IS
BEGIN
    EXECUTE IMMEDIATE  'CREATE TABLE '|| p_table_name || p_table_columns;
END create_table_procedure;
```

```
CREATE ROLE create_table_role;
GRANT CREATE TABLE TO create_table_role;
GRANT create_table_role TO demo;
```

```
EXEC
create_table_procedure('ORDERS_QUEUE_WA',
'(queue_id NUMBER,queue_act_id
NUMBER,queue_item_id NUMBER)');
```

ORA-01031: insufficient privileges

```
GRANT CREATE TABLE TO demo;
```

- 12c

```
GRANT create  table  role to create  table  procedure;
```

DDL Operations

- Drop Objects

```
CREATE OR REPLACE   PROCEDURE  drop_table (p_table_name   VARCHAR2)  IS
BEGIN
    EXECUTE IMMEDIATE  'DROP TABLE '|| p_table_name;
END drop_table;
```

```
EXEC drop_table('ORDERS_QUEUE_CA');
```

Single Row Selects

```
CREATE OR REPLACE FUNCTION get_count(p_table VARCHAR2)
RETURN NUMBER IS
   l_count NUMBER;
   l_query VARCHAR2(200);
 BEGIN
   l_query := 'SELECT COUNT(*) FROM ' ||
                    p_table ;

  EXECUTE IMMEDIATE l_query INTO l_count;

  RETURN l_count;
END get_count;
/
```

```
DECLARE
  l_cnt NUMBER;
BEGIN
  l_cnt := get_count('ORDERS');
  DBMS_OUTPUT.PUT_LINE('Count is '||l_cnt);
END;
/
```

```
DECLARE
  l_cnt NUMBER;
BEGIN
  l_cnt := get_count('ITEMS');
  DBMS_OUTPUT.PUT_LINE('Count is '||l_cnt);
END;
/
```

Single Row Selects

```
CREATE OR REPLACE FUNCTION get_order_count(p_column VARCHAR2,
                                           p_value    NUMBER )
  RETURN NUMBER IS
   l_count NUMBER;
   l_query VARCHAR2(200);
 BEGIN
  l_query := 'SELECT COUNT(*) FROM orders WHERE ' ||
                 p_column ||' = :col_value ';

  EXECUTE IMMEDIATE l_query INTO l_count USING p_value;
  RETURN l_count;
END get_order_count;
/
```

```
SELECT COUNT(*) FROM orders WHERE
 order_act_id = 1;
```

```
SELECT COUNT(*) FROM orders WHERE
 order_item_id = 2;
```

```
DECLARE
  l_cnt NUMBER;
BEGIN
  l_cnt := get_order_count('order_act_id',1);
  DBMS_OUTPUT.PUT_LINE('Count is '||l_cnt);
END;
/
```

```
DECLARE
  l_cnt NUMBER;
BEGIN
  l_cnt := get_order_count('order_item_id',2);
  DBMS_OUTPUT.PUT_LINE('Count is '||l_cnt);
END;
/
```

Passing Schema Object Names

- Not as Bind Variables

```
CREATE OR REPLACE FUNCTION get_count(p_table VARCHAR2)
RETURN NUMBER IS
   l_count NUMBER;
   l_query VARCHAR2(200);
 BEGIN
   l_query := 'SELECT COUNT(*) FROM ' || p_table ;
  EXECUTE IMMEDIATE l_query INTO l_count;


   l_query := 'SELECT COUNT(*) FROM :1';
  EXECUTE IMMEDIATE l_query INTO l_count USING p_table;

  RETURN l_count;
END get_count;
/
```

Performance Consideration

- **Bind Variables**

```
CREATE OR REPLACE FUNCTION get_order_count(p_column VARCHAR2,
                                           p_value    NUMBER )
  RETURN NUMBER IS
   l_count NUMBER;
   l_query VARCHAR2(200);
 BEGIN
  l_query := 'SELECT COUNT(*) FROM orders WHERE ' ||
             p_column ||' = :col_value ';

  EXECUTE IMMEDIATE l_query INTO l_count USING p_value;
  RETURN l_count;
END get_order_count;
```

```
CREATE OR REPLACE FUNCTION get_order_count(p_column VARCHAR2,
                                           p_value    NUMBER )
  RETURN NUMBER IS
   l_count NUMBER;
   l_query VARCHAR2(200);
 BEGIN
  l_query := 'SELECT COUNT(*) FROM orders WHERE ' ||
             p_column || ' = ' ||p_value ;

  EXECUTE IMMEDIATE l_query INTO l_count;
  RETURN l_count;
END get_order_count;
```

Multi-Row Selects

Execute Immediate

Ref Cursors

Ref Cursors

- Not in Nested Blocks

```
CREATE OR REPLACE PROCEDURE apply_fees(p_column VARCHAR2,
                                       p_value    NUMBER) IS
  TYPE cur_ref IS REF CURSOR;
  cur_account cur_ref;
  l_query VARCHAR2(400);
  l_act_id accounts.act_id%TYPE;
BEGIN
  l_query := 'SELECT act_id FROM accounts';

  IF p_column IS NOT NULL THEN
    l_query := l_query||' WHERE '||p_column||' = :pvalue';
    OPEN cur_account FOR l_query USING p_value;
  ELSE
     OPEN cur_account FOR l_query;
  END IF;

  LOOP
   FETCH cur_account INTO l_act_id;
   EXIT WHEN cur_account%NOTFOUND;
   UPDATE accounts SET act_bal = act_bal - 10 WHERE act_id = l_act_id;
  COMMIT;
 END LOOP;
END apply_fees;
```

Fetching in Records

```
CREATE OR REPLACE PROCEDURE initiate_order(p_where VARCHAR2)  IS

   TYPE cur_ref IS REF CURSOR;
   cur_order cur_ref;
   TYPE order_rec IS RECORD( act_id      orders_queue.queue_act_id%TYPE,
                             item_id     orders_queue.queue_item_id%TYPE);
   l_order_rec  order_rec;
   l_item_rec   items%ROWTYPE;
   l_query VARCHAR2(400);
 BEGIN
   l_query := 'SELECT queue_act_id,queue_item_id  FROM  orders_queue ' ||  p_where ;
   OPEN    cur_order FOR l_query;
   LOOP
      FETCH cur_order INTO l_order_rec;
      EXIT WHEN cur_order%NOTFOUND;

      EXECUTE IMMEDIATE  'SELECT * FROM items WHERE item_id = :item_id '
         INTO l_item_rec  USING l_order_rec.item_id ;

      process_order(l_order_rec.act_id,  l_order_rec.item_id, l_item_rec.item_value   );
  END LOOP;

END initiate_order;
```

DML Statements

- Insert Statement

```
CREATE OR REPLACE PROCEDURE insert_record (p_table_name     VARCHAR2,
                                           p_col1_name      VARCHAR2,
                                           p_col1_value     NUMBER,
                                           p_col2_name      VARCHAR2,
                                           p_col2_value     NUMBER )
  BEGIN
    EXECUTE IMMEDIATE  'INSERT INTO  '||p_table_name || '('||
                                          p_col1_name||' , '||
                                          p_col2_name||
                                        ' )   '||
                                  ' VALUES( :col1_value,:col2_value)'
                                    USING p_col1_value, p_col2_value;
    COMMIT;
  END insert_record;
```

Number of Bind Values

- Equal to Bind Variables

```
…..
   BEGIN
      EXECUTE IMMEDIATE  'INSERT INTO  '||p_table_name || '('||
                                                                  p_col1_name||' , '||
                                                                  p_col2_name||
                                                              ' )   '||
                                                     ' VALUES( :col1_value,:col1_value)'
                                                       USING p_col1_value, p_col1_value;
      COMMIT;
……
```

Type of Bind Variables

- Supports All SQL Datatypes
- Oracle 12c: Supports Pl/SQL Only Datatypes Like
  - Boolean
  - Associative Arrays With PLS_INTEGER indexes
  - Composite Types Declared in Package Specification Like Records, Collections

Passing Nulls in Dynamic SQL

```
DECLARE
   ....

  BEGIN
     EXECUTE IMMEDIATE  'INSERT INTO  '||p_table_name || '('||
                                                    p_col1_name||' , '||
                                                    p_col2_name||
                                                 ' )   '||
                                        ' VALUES( :col1_value,:col2_value)'
                                          USING p_col1_value, NULL;
     ....
```

```
DECLARE
   ....
   l_null VARCHAR2(1);
  BEGIN
     EXECUTE IMMEDIATE  'INSERT INTO  '||p_table_name || '('||
                                                    p_col1_name||' , '||
                                                    p_col2_name||
                                                 ' )   '||
                                        ' VALUES( :col1_value,:col2_value)'
                                          USING p_col1_value, l_null;
     ....
```

Update Statements

```
CREATE OR REPLACE PROCEDURE update_record (p_table_name   VARCHAR2,
                                           p_col1_name    VARCHAR2,
                                           p_col1_value   NUMBER,
                                           p_where_col    VARCHAR2,
                                           p_where_value  NUMBER )
  BEGIN
    EXECUTE IMMEDIATE  'UPDATE  '||p_table_name || ' SET '||
                                                p_col1_name||' = :p_col1_value '||
                                        ' WHERE   '|| p_where_col ||' = :p_where_value '
                                USING p_col1_value, p_where_value;

    COMMIT;
  END update_record;
```

Returning Into Clause

```
DECLARE
   l_item_value   items.item_value%TYPE   := 100;
   l_act_id       accounts.act_id%TYPE    := 1;
   l_cust_id      customers.cust_id%TYPE;
   l_act_bal      accounts.act_bal%TYPE;

BEGIN

   EXECUTE IMMEDIATE  'UPDATE   accounts SET act_bal = act_bal  - :p_item_val '||
       ' WHERE act_id = :p_act_id RETURNING act_cust_id,act_bal INTO :l_id, :l_bal '
       USING l_item_value, l_act_id RETURNING INTO l_cust_id,l_act_bal;
   COMMIT;
END;
```

Returning Into Clause

```
DECLARE
   l_item_value   items.item_value%TYPE   := 100;
   l_act_id       accounts.act_id%TYPE    := 1;
   l_cust_id      customers.cust_id%TYPE;
   l_act_bal      accounts.act_bal%TYPE;

BEGIN

   EXECUTE IMMEDIATE  'UPDATE   accounts SET act_bal = act_bal  - :p_item_val '||
       ' WHERE act_id = :p_act_id RETURNING act_cust_id,act_bal INTO :l_id, :l_bal '
       USING l_item_value, l_act_id RETURNING INTO l_cust_id,l_act_bal;
   COMMIT;
END;
```

```
DECLARE
   l_item_value   items.item_value%TYPE   := 100;
   l_act_id       accounts.act_id%TYPE    := 1;
   l_cust_id      customers.cust_id%TYPE;
   l_act_bal      accounts.act_bal%TYPE;

BEGIN

   EXECUTE IMMEDIATE  'UPDATE   accounts SET act_bal = act_bal  - :p_item_val '||
       ' WHERE act_id = :p_act_id RETURNING act_cust_id,act_bal INTO :l_id, :l_bal '
       USING l_item_value, l_act_id, OUT l_cust_id, OUT l_act_bal;
   COMMIT;
END;
```

Delete Statements

```
CREATE OR REPLACE PROCEDURE delete_table  (p_table_name  VARCHAR2)
   BEGIN
      EXECUTE IMMEDIATE  ' DELETE FROM  ' ||p_table_name;
     COMMIT;
   END delete_table;
```

Execute Anonymous Blocks

```
DECLARE
 l_sql     VARCHAR2(500);
 l_inout NUMBER := 1;
 l_out     NUMBER := 2 ;
 l_num   NUMBER := 1;
BEGIN
 l_sql := ' BEGIN  :l_inout := :l_inout + :l_num *2 ;   ' ||
           '              :l_out     := :l_num / 2 ;   END; ' ;
 EXECUTE IMMEDIATE l_sql USING  IN OUT l_inout, l_num, OUT l_out;
END;
```

- Semi-Colon After End
- Duplicate Placeholders

Length of SQL String

- Maximum Parse Length Pre 11g : 64K

```
DECLARE
  l_sql1  VARCHAR2(32767):= '…… ';
  l_sql 2 VARCHAR2(32767):= '…… ‘;
BEGIN
  EXECUTE IMMEDIATE l_sql1|| l_sql2;
END;
```

- CLOB Support

```
DECLARE
  l_sql     CLOB;
  l_inout NUMBER := 1;
  l_out     NUMBER := 2 ;
  l_num   NUMBER;
BEGIN
  l_sql := ' BEGIN  :l_inout := :l_inout + :l_num *2 ;   ' ||
            '              :l_out     := :l_num / 2 ;   END; ' ;
  EXECUTE IMMEDIATE l_sql USING  IN OUT l_inout, l_num, OUT l_out;
END;
```

Executing Procedures

```
CREATE OR REPLACE PROCEDURE
   calculate_tier  (p_act_id   IN        accounts.act_id%TYPE,
                    p_act_bal  IN OUT accounts.act_bal%TYPE,
                    p_tier            OUT NUMBER) IS
    ....
    ....
   END calculate_tier;
```

```
DECLARE
 l_act_id accounts.act_id%TYPE := 1;
 l_act_bal accounts.act_bal%TYPE;
 l_tier NUMBER;
BEGIN
 EXECUTE IMMEDIATE ' CALL calculate_tier(:act_id,:act_bal,:tier) '
 USING l_act_id, IN OUT l_act_bal, OUT l_tier;
END;
```

```
DECLARE
 l_act_id    accounts.act_id%TYPE := 1;
 l_act_bal  accounts.act_bal%TYPE;
 l_tier NUMBER;
BEGIN
 EXECUTE IMMEDIATE ' BEGIN calculate_tier(:act_id,:act_bal,:tier);  END; '
 USING l_act_id, IN OUT l_act_bal, OUT l_tier;
END;
```

Execute Functions

```
CREATE OR REPLACE FUNCTION
   get_tier  (p_act_id    IN        accounts.act_id%TYPE,
              p_act_bal  IN OUT accounts.act_bal%TYPE,
              p_tier            OUT NUMBER)  RETURN NUMBER IS
    ....
    ....
   END get_tier;
```

```
DECLARE
 l_act_id    accounts.act_id%TYPE := 1;
 l_act_bal  accounts.act_bal%TYPE;
 l_tier NUMBER;
 l_out NUMBER;
BEGIN
 EXECUTE IMMEDIATE ' BEGIN :l_out := get_tier(:act_id,:act_bal,:tier);  END; '
 USING OUT l_out, l_act_id, IN OUT l_act_bal, OUT l_tier;
END;
```

Specifying Hints

```
CREATE OR REPLACE PROCEDURE apply_fees(p_column VARCHAR2,
                                       p_value  NUMBER,
                                       p_hint   VARCHAR2) IS
  TYPE cur_ref IS REF CURSOR;
  cur_account cur_ref;
  l_query VARCHAR2(400);
  l_act_id accounts.act_id%TYPE;
BEGIN
  l_query := 'SELECT  '|| p_hint ||' act_id FROM accounts';

  IF p_column IS NOT NULL THEN
    l_query := l_query||' WHERE '||p_column||' = :pvalue';
    OPEN cur_account FOR l_query USING p_value;
  ELSE
     OPEN cur_account FOR l_query;
  END IF;

  LOOP
   FETCH cur_account INTO l_act_id;
   EXIT WHEN cur_account%NOTFOUND;
   UPDATE accounts SET act_bal = act_bal - 10 WHERE act_id = l_act_id;
  COMMIT;
 END LOOP;
END apply_fees;
```

```
EXEC apply_fees('act_id ' , 1,  '/*+ PARALLEL(accounts, 3)  */' );
```

Specifying Session Control Statements

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-RRRR';
```

```
CREATE OR REPLACE PROCEDURE set_date_format(p_format VARCHAR2) IS
BEGIN
  EXECUTE IMMEDIATE 'ALTER SESSION SET NLS_DATE_FORMAT = '||p_format;
END;
```

```
EXEC  set_date_format('''DD-MON-RRRR''');
```

Invokers Right & Dynamic SQL

```
CREATE OR REPLACE   PROCEDURE  create_table (p_table_name      VARCHAR2,
                                              p_table_columns  VARCHAR2)
AUTHID CURRENT_USER IS
BEGIN
    EXECUTE IMMEDIATE  'CREATE TABLE '|| p_table_name || p_table_columns;
END create_table;
```

```
CREATE OR REPLACE PROCEDURE delete_table  (p_table_name  VARCHAR2)
AUTHID CURRENT_USER IS
   BEGIN
      EXECUTE IMMEDIATE  'DELETE FROM  '||p_table_name;
      COMMIT;
END delete_table;
```

Database Links With Dynamic SQL

db2 instance

```
create public database link  db1link connect to demo identified by  demo
using 'db1';
```

db2 instance

```
CREATE OR REPLACE PROCEDURE delete_table  (p_table_name  VARCHAR2,
                                            p_dblink      VARCHAR2)  IS
 BEGIN
      EXECUTE IMMEDIATE  'DELETE FROM  '||p_table_name||'@'||p_dblink;
     COMMIT;
END update_record;
```

```
EXEC delete_table  ('accounts', 'db1link') ;
```

SQL Injection

- Statement Modification

```
CREATE OR REPLACE PROCEDURE delete_order(p_column VARCHAR2,
                                         p_value    VARCHAR2 )  IS
l_query VARCHAR2(200);
 BEGIN
   l_query := 'DELETE FROM orders WHERE ' ||
                  p_column ||' = '||p_value ;
   DBMS_OUTPUT.PUT_LINE(l_query);

   EXECUTE IMMEDIATE l_query;
END delete_order;
/
```

```
DELETE
FROM orders
WHERE
order_act_id = 1;
```

```
DELETE
FROM orders
WHERE
order_act_id = 1
 OR 1=1;
```

Normal Execution

```
EXEC delete_order('order_act_id',1);
```

Statement Modification

```
EXEC delete_order(' order_act_id',  ' 1 OR 1=1 ');
```

SQL Injection

- Use Bind Variables

```
CREATE OR REPLACE PROCEDURE delete_order(p_column VARCHAR2,
                                         p_value    VARCHAR2 )  IS
l_query VARCHAR2(200);
 BEGIN
   l_query := 'DELETE FROM orders WHERE ' ||
                  p_column ||' = :l_value';
   DBMS_OUTPUT.PUT_LINE(l_query);

   EXECUTE IMMEDIATE l_query USING p_value;
END delete_order;
/
```

```
DELETE
FROM orders
WHERE
order_act_id = 1;
```

```
ORA-01722:
invalid number
```

Normal Execution

```
EXEC delete_order('order_act_id',1);
```

Statement Modification

```
EXEC delete_order('order_act_id',  ' 1 OR 1=1 ');
```

SQL Injection

- Statement Injection

```
CREATE OR REPLACE PROCEDURE calc(p_condition VARCHAR2) IS
 l_block VARCHAR2(1000);
BEGIN
 l_block :=
        ' BEGIN  IF  '||p_condition||' = "A" THEN proc1; END IF; END; ';
EXECUTE IMMEDIATE l_block;
...
END calc;
```

```
BEGIN
 IF 'A' = 'A' THEN
  proc1;
 END IF;
END;
```

Normal Execution

```
EXEC   calc('"A"');
```

```
BEGIN
 IF 1=1 THEN
  DELETE FROM ORDERS;
 END IF;
 IF 'A' = 'A'  THEN proc1;
 END IF;
END;
```

Statement Injection

```
EXEC   calc('1=1 THEN DELETE FROM orders; END IF; IF "A"');
```

SQL Injection

- Validations

```
CREATE OR REPLACE PROCEDURE calc(p_condition VARCHAR2) IS
 l_block VARCHAR2(1000);
 malicious_attack EXCEPTION;
BEGIN
 IF INSTR(p_condition, ';') > 0 THEN RAISE malicious_attack; END IF;
 IF INSTR(p_condition, 'END IF;') > 0 THEN RAISE malicious_attack; END IF;
 l_block :=
        ' BEGIN  IF  '||p_condition||' = "A" THEN proc1; END IF; END; ';
 EXECUTE IMMEDIATE l_block;
…
EXCEPTION
 WHEN malicious_attack THEN
  DBMS_OUTPUT.PUT_LINE('Suspicious Input '||p_condition);
  RAISE;
END calc;
```

Statement Injection

```
EXEC   calc('1=1 THEN DELETE FROM orders; END IF; IF "A"');
```

Summary

What is Dynamic SQL?

Usage

Execute Immediate

SQL Injection

Native Dynamic SQL

Pankaj Jain

@twit_pankajj

pluralsight
hardcore dev and IT training

What is Dynamic SQL?

SQL Statement Known at Runtime

Static SQL

```
CREATE OR REPLACE FUNCTION
get_count(p_act_id accounts.act_id%TYPE)
RETURN NUMBER IS
   l_count NUMBER;
 BEGIN
   SELECT COUNT(*)
      INTO l_count
   FROM   orders
  WHERE   order_act_id =  p_act_id;
  RETURN l_count;
END get_count;
```

Dynamic SQL

```
CREATE OR REPLACE FUNCTION get_count(
                p_where VARCHAR2)  RETURN NUMBER
IS
   l_count NUMBER;
   l_select VARCHAR2(100) := 'SELECT COUNT(*) ';
   l_from   VARCHAR2(60) := 'FROM orders  ';
   l_query VARCHAR2(200);
 BEGIN
   l_query := l_select ||
                  l_from ||
                  p_where;
  EXECUTE IMMEDIATE l_query INTO l_count;
  RETURN l_count;
END get_count;
```

Static vs Dynamic SQL

Static SQL	Dynamic SQL
SQL Known at Compile Time	SQL Known at Runtime
Compiler Verifies Object References	Compiler Cannot Verify Object References
Compiler Can Verify Privileges	Compiler Cannot Verify Privileges
Less Flexible	More Flexible
Faster	Slower

Common Uses

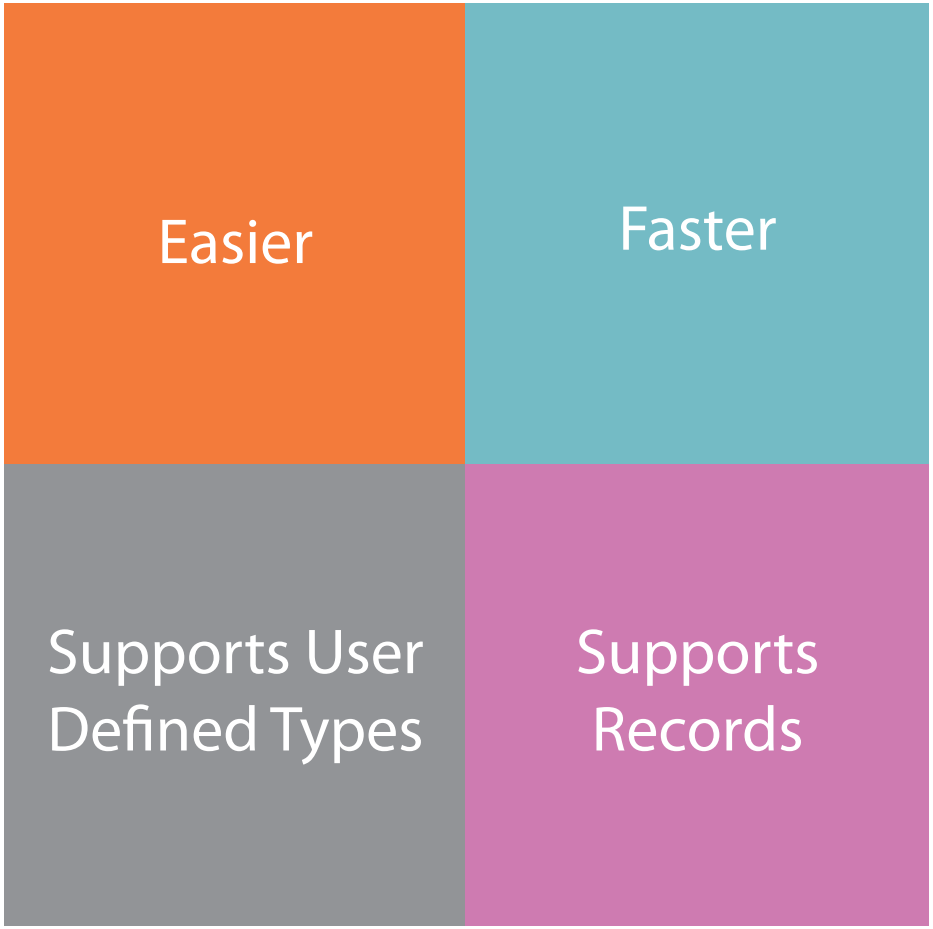
- Dynamic Queries
- Dynamic Sorts
- Dynamic Subprogram Invocation
- Dynamic Optimizations
- DDL
- Frameworks
- Varying Table Definitions

Invoking Dynamic SQL

Native Dynamic SQL

DBMS_SQL

Why Use Native Dynamic SQL?



Native Dynamic SQL

```
EXECUTE IMMEDIATE <dynamic_sql_string>

[INTO {select_var1[, select_var2].. | record }

[USING [ IN | OUT | IN OUT ] bind_var1
[ ,        [ IN | OUT | IN OUT ] bind_var2]…]

[{RETURNING | RETURN} INTO bind_var1
[,                                  bind_var2]...];
```

- Ref Cursors

DDL Operations

- Create Objects

```
CREATE OR REPLACE   PROCEDURE  create_table (p_table_name      VARCHAR2,
                                                p_table_columns  VARCHAR2)  IS
BEGIN
    EXECUTE IMMEDIATE  'CREATE TABLE '|| p_table_name || p_table_columns;
END create_table;
```

```
EXEC create_table(
                  'ORDERS_QUEUE_CA',
                  '(queue_id NUMBER,queue_act_id NUMBER,queue_item_id NUMBER)'
                  );
```

Object Privileges in Native Dynamic SQL

- **Direct Grants**

User demo

```
CREATE OR REPLACE   PROCEDURE  create_table_procedure(p_table_name      VARCHAR2,
                                                        p_table_columns  VARCHAR2)  IS
BEGIN
    EXECUTE IMMEDIATE  'CREATE TABLE '|| p_table_name || p_table_columns;
END create_table_procedure;
```

```
CREATE ROLE create_table_role;
GRANT CREATE TABLE TO create_table_role;
GRANT create_table_role TO demo;
```

```
EXEC
create_table_procedure('ORDERS_QUEUE_WA',
'(queue_id NUMBER,queue_act_id
NUMBER,queue_item_id NUMBER)');
```

ORA-01031: insufficient privileges

```
GRANT CREATE TABLE TO demo;
```

- 12c

```
GRANT create  table  role to create  table  procedure;
```

DDL Operations

- Drop Objects

```
CREATE OR REPLACE   PROCEDURE  drop_table (p_table_name   VARCHAR2)  IS
BEGIN
    EXECUTE IMMEDIATE  'DROP TABLE '|| p_table_name;
END drop_table;
```

```
EXEC drop_table('ORDERS_QUEUE_CA');
```

Single Row Selects

```
CREATE OR REPLACE FUNCTION get_count(p_table VARCHAR2)
RETURN NUMBER IS
   l_count NUMBER;
   l_query VARCHAR2(200);
 BEGIN
   l_query := 'SELECT COUNT(*) FROM ' ||
                    p_table ;

  EXECUTE IMMEDIATE l_query INTO l_count;

  RETURN l_count;
END get_count;
/
```

```
DECLARE
  l_cnt NUMBER;
BEGIN
  l_cnt := get_count('ORDERS');
  DBMS_OUTPUT.PUT_LINE('Count is '||l_cnt);
END;
/
```

```
DECLARE
  l_cnt NUMBER;
BEGIN
  l_cnt := get_count('ITEMS');
  DBMS_OUTPUT.PUT_LINE('Count is '||l_cnt);
END;
/
```

Single Row Selects

```
CREATE OR REPLACE FUNCTION get_order_count(p_column VARCHAR2,
                                           p_value    NUMBER )
  RETURN NUMBER IS
   l_count NUMBER;
   l_query VARCHAR2(200);
 BEGIN
  l_query := 'SELECT COUNT(*) FROM orders WHERE ' ||
                 p_column ||' = :col_value ';

  EXECUTE IMMEDIATE l_query INTO l_count USING p_value;
  RETURN l_count;
END get_order_count;
/
```

```
SELECT COUNT(*) FROM orders WHERE
 order_act_id = 1;
```

```
SELECT COUNT(*) FROM orders WHERE
 order_item_id = 2;
```

```
DECLARE
  l_cnt NUMBER;
BEGIN
  l_cnt := get_order_count('order_act_id',1);
  DBMS_OUTPUT.PUT_LINE('Count is '||l_cnt);
END;
/
```

```
DECLARE
  l_cnt NUMBER;
BEGIN
  l_cnt := get_order_count('order_item_id',2);
  DBMS_OUTPUT.PUT_LINE('Count is '||l_cnt);
END;
/
```

Passing Schema Object Names

- Not as Bind Variables

```
CREATE OR REPLACE FUNCTION get_count(p_table VARCHAR2)
RETURN NUMBER IS
   l_count NUMBER;
   l_query VARCHAR2(200);
 BEGIN
   l_query := 'SELECT COUNT(*) FROM ' || p_table ;
  EXECUTE IMMEDIATE l_query INTO l_count;


   l_query := 'SELECT COUNT(*) FROM :1';
  EXECUTE IMMEDIATE l_query INTO l_count USING p_table;

  RETURN l_count;
END get_count;
/
```

Performance Consideration

- **Bind Variables**

```
CREATE OR REPLACE FUNCTION get_order_count(p_column VARCHAR2,
                                           p_value    NUMBER )
  RETURN NUMBER IS
   l_count NUMBER;
   l_query VARCHAR2(200);
 BEGIN
  l_query := 'SELECT COUNT(*) FROM orders WHERE ' ||
             p_column ||' = :col_value ';

  EXECUTE IMMEDIATE l_query INTO l_count USING p_value;
  RETURN l_count;
END get_order_count;
```

```
CREATE OR REPLACE FUNCTION get_order_count(p_column VARCHAR2,
                                           p_value    NUMBER )
  RETURN NUMBER IS
   l_count NUMBER;
   l_query VARCHAR2(200);
 BEGIN
  l_query := 'SELECT COUNT(*) FROM orders WHERE ' ||
             p_column || ' = ' ||p_value ;

  EXECUTE IMMEDIATE l_query INTO l_count;
  RETURN l_count;
END get_order_count;
```

Multi-Row Selects

Execute Immediate

Ref Cursors

Ref Cursors

- Not in Nested Blocks

```
CREATE OR REPLACE PROCEDURE apply_fees(p_column VARCHAR2,
                                       p_value    NUMBER) IS
  TYPE cur_ref IS REF CURSOR;
  cur_account cur_ref;
  l_query VARCHAR2(400);
  l_act_id accounts.act_id%TYPE;
BEGIN
  l_query := 'SELECT act_id FROM accounts';

  IF p_column IS NOT NULL THEN
    l_query := l_query||' WHERE '||p_column||' = :pvalue';
    OPEN cur_account FOR l_query USING p_value;
  ELSE
     OPEN cur_account FOR l_query;
  END IF;

  LOOP
   FETCH cur_account INTO l_act_id;
   EXIT WHEN cur_account%NOTFOUND;
   UPDATE accounts SET act_bal = act_bal - 10 WHERE act_id = l_act_id;
  COMMIT;
 END LOOP;
END apply_fees;
```

Fetching in Records

```
CREATE OR REPLACE PROCEDURE initiate_order(p_where VARCHAR2)  IS

   TYPE cur_ref IS REF CURSOR;
   cur_order cur_ref;
   TYPE order_rec IS RECORD( act_id      orders_queue.queue_act_id%TYPE,
                             item_id     orders_queue.queue_item_id%TYPE);
   l_order_rec  order_rec;
   l_item_rec   items%ROWTYPE;
   l_query VARCHAR2(400);
 BEGIN
   l_query := 'SELECT queue_act_id,queue_item_id  FROM  orders_queue ' ||  p_where ;
   OPEN    cur_order FOR l_query;
   LOOP
      FETCH cur_order INTO l_order_rec;
      EXIT WHEN cur_order%NOTFOUND;

      EXECUTE IMMEDIATE  'SELECT * FROM items WHERE item_id = :item_id '
         INTO l_item_rec  USING l_order_rec.item_id ;

      process_order(l_order_rec.act_id,  l_order_rec.item_id, l_item_rec.item_value   );
  END LOOP;

END initiate_order;
```

DML Statements

- Insert Statement

```
CREATE OR REPLACE PROCEDURE insert_record (p_table_name     VARCHAR2,
                                           p_col1_name      VARCHAR2,
                                           p_col1_value     NUMBER,
                                           p_col2_name      VARCHAR2,
                                           p_col2_value     NUMBER )
  BEGIN
    EXECUTE IMMEDIATE  'INSERT INTO  '||p_table_name || '('||
                                          p_col1_name||' , '||
                                          p_col2_name||
                                        ' )   '||
                                  ' VALUES( :col1_value,:col2_value)'
                                    USING p_col1_value, p_col2_value;
    COMMIT;
  END insert_record;
```

Number of Bind Values

- Equal to Bind Variables

```
…..
   BEGIN
      EXECUTE IMMEDIATE  'INSERT INTO  '||p_table_name || '('||
                                                                  p_col1_name||' , '||
                                                                  p_col2_name||
                                                              ' )   '||
                                                     ' VALUES( :col1_value,:col1_value)'
                                                       USING p_col1_value, p_col1_value;
      COMMIT;
……
```

Type of Bind Variables

- Supports All SQL Datatypes
- Oracle 12c: Supports Pl/SQL Only Datatypes Like
  - Boolean
  - Associative Arrays With PLS_INTEGER indexes
  - Composite Types Declared in Package Specification Like Records, Collections

Passing Nulls in Dynamic SQL

```
DECLARE
   ....

  BEGIN
     EXECUTE IMMEDIATE  'INSERT INTO  '||p_table_name || '('||
                                                    p_col1_name||' , '||
                                                    p_col2_name||
                                                 ' )   '||
                                        ' VALUES( :col1_value,:col2_value)'
                                          USING p_col1_value, NULL;
     ....
```

```
DECLARE
   ....
   l_null VARCHAR2(1);
  BEGIN
     EXECUTE IMMEDIATE  'INSERT INTO  '||p_table_name || '('||
                                                    p_col1_name||' , '||
                                                    p_col2_name||
                                                 ' )   '||
                                        ' VALUES( :col1_value,:col2_value)'
                                          USING p_col1_value, l_null;
     ....
```

Update Statements

```
CREATE OR REPLACE PROCEDURE update_record (p_table_name   VARCHAR2,
                                           p_col1_name    VARCHAR2,
                                           p_col1_value   NUMBER,
                                           p_where_col    VARCHAR2,
                                           p_where_value  NUMBER )
  BEGIN
    EXECUTE IMMEDIATE  'UPDATE  '||p_table_name || ' SET '||
                                                p_col1_name||' = :p_col1_value '||
                                        ' WHERE   '|| p_where_col ||' = :p_where_value '
                                USING p_col1_value, p_where_value;

    COMMIT;
  END update_record;
```

Returning Into Clause

```
DECLARE
   l_item_value   items.item_value%TYPE   := 100;
   l_act_id       accounts.act_id%TYPE    := 1;
   l_cust_id      customers.cust_id%TYPE;
   l_act_bal      accounts.act_bal%TYPE;

BEGIN

   EXECUTE IMMEDIATE  'UPDATE   accounts SET act_bal = act_bal  - :p_item_val '||
       ' WHERE act_id = :p_act_id RETURNING act_cust_id,act_bal INTO :l_id, :l_bal '
       USING l_item_value, l_act_id RETURNING INTO l_cust_id,l_act_bal;
   COMMIT;
END;
```

Returning Into Clause

```
DECLARE
   l_item_value   items.item_value%TYPE   := 100;
   l_act_id       accounts.act_id%TYPE    := 1;
   l_cust_id      customers.cust_id%TYPE;
   l_act_bal      accounts.act_bal%TYPE;

BEGIN

   EXECUTE IMMEDIATE  'UPDATE   accounts SET act_bal = act_bal  - :p_item_val '||
       ' WHERE act_id = :p_act_id RETURNING act_cust_id,act_bal INTO :l_id, :l_bal '
       USING l_item_value, l_act_id RETURNING INTO l_cust_id,l_act_bal;
   COMMIT;
END;
```

```
DECLARE
   l_item_value   items.item_value%TYPE   := 100;
   l_act_id       accounts.act_id%TYPE    := 1;
   l_cust_id      customers.cust_id%TYPE;
   l_act_bal      accounts.act_bal%TYPE;

BEGIN

   EXECUTE IMMEDIATE  'UPDATE   accounts SET act_bal = act_bal  - :p_item_val '||
       ' WHERE act_id = :p_act_id RETURNING act_cust_id,act_bal INTO :l_id, :l_bal '
       USING l_item_value, l_act_id, OUT l_cust_id, OUT l_act_bal;
   COMMIT;
END;
```

Delete Statements

```
CREATE OR REPLACE PROCEDURE delete_table  (p_table_name  VARCHAR2)
   BEGIN
      EXECUTE IMMEDIATE  ' DELETE FROM  ' ||p_table_name;
     COMMIT;
   END delete_table;
```

Execute Anonymous Blocks

```
DECLARE
 l_sql     VARCHAR2(500);
 l_inout NUMBER := 1;
 l_out     NUMBER := 2 ;
 l_num   NUMBER := 1;
BEGIN
 l_sql := ' BEGIN  :l_inout := :l_inout + :l_num *2 ;   ' ||
           '              :l_out     := :l_num / 2 ;   END; ' ;
 EXECUTE IMMEDIATE l_sql USING  IN OUT l_inout, l_num, OUT l_out;
END;
```

- Semi-Colon After End
- Duplicate Placeholders

Length of SQL String

- Maximum Parse Length Pre 11g : 64K

```
DECLARE
  l_sql1  VARCHAR2(32767):= '…… ';
  l_sql 2 VARCHAR2(32767):= '…… ‘;
BEGIN
  EXECUTE IMMEDIATE l_sql1|| l_sql2;
END;
```

- CLOB Support

```
DECLARE
  l_sql     CLOB;
  l_inout NUMBER := 1;
  l_out     NUMBER := 2 ;
  l_num   NUMBER;
BEGIN
  l_sql := ' BEGIN  :l_inout := :l_inout + :l_num *2 ;   ' ||
            '              :l_out     := :l_num / 2 ;   END; ' ;
  EXECUTE IMMEDIATE l_sql USING  IN OUT l_inout, l_num, OUT l_out;
END;
```

Executing Procedures

```
CREATE OR REPLACE PROCEDURE
   calculate_tier  (p_act_id   IN        accounts.act_id%TYPE,
                    p_act_bal  IN OUT accounts.act_bal%TYPE,
                    p_tier            OUT NUMBER) IS
    ....
    ....
   END calculate_tier;
```

```
DECLARE
 l_act_id accounts.act_id%TYPE := 1;
 l_act_bal accounts.act_bal%TYPE;
 l_tier NUMBER;
BEGIN
 EXECUTE IMMEDIATE ' CALL calculate_tier(:act_id,:act_bal,:tier) '
 USING l_act_id, IN OUT l_act_bal, OUT l_tier;
END;
```

```
DECLARE
 l_act_id    accounts.act_id%TYPE := 1;
 l_act_bal  accounts.act_bal%TYPE;
 l_tier NUMBER;
BEGIN
 EXECUTE IMMEDIATE ' BEGIN calculate_tier(:act_id,:act_bal,:tier);  END; '
 USING l_act_id, IN OUT l_act_bal, OUT l_tier;
END;
```

Execute Functions

```
CREATE OR REPLACE FUNCTION
   get_tier  (p_act_id    IN        accounts.act_id%TYPE,
              p_act_bal  IN OUT accounts.act_bal%TYPE,
              p_tier            OUT NUMBER)  RETURN NUMBER IS
    ....
    ....
   END get_tier;
```

```
DECLARE
 l_act_id    accounts.act_id%TYPE := 1;
 l_act_bal  accounts.act_bal%TYPE;
 l_tier NUMBER;
 l_out NUMBER;
BEGIN
 EXECUTE IMMEDIATE ' BEGIN :l_out := get_tier(:act_id,:act_bal,:tier);  END; '
 USING OUT l_out, l_act_id, IN OUT l_act_bal, OUT l_tier;
END;
```

Specifying Hints

```
CREATE OR REPLACE PROCEDURE apply_fees(p_column VARCHAR2,
                                       p_value  NUMBER,
                                       p_hint   VARCHAR2) IS
  TYPE cur_ref IS REF CURSOR;
  cur_account cur_ref;
  l_query VARCHAR2(400);
  l_act_id accounts.act_id%TYPE;
BEGIN
  l_query := 'SELECT  '|| p_hint ||' act_id FROM accounts';

  IF p_column IS NOT NULL THEN
    l_query := l_query||' WHERE '||p_column||' = :pvalue';
    OPEN cur_account FOR l_query USING p_value;
  ELSE
     OPEN cur_account FOR l_query;
  END IF;

  LOOP
   FETCH cur_account INTO l_act_id;
   EXIT WHEN cur_account%NOTFOUND;
   UPDATE accounts SET act_bal = act_bal - 10 WHERE act_id = l_act_id;
  COMMIT;
 END LOOP;
END apply_fees;
```

```
EXEC apply_fees('act_id ' , 1,  '/*+ PARALLEL(accounts, 3)  */' );
```

Specifying Session Control Statements

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-RRRR';
```

```
CREATE OR REPLACE PROCEDURE set_date_format(p_format VARCHAR2) IS
BEGIN
  EXECUTE IMMEDIATE 'ALTER SESSION SET NLS_DATE_FORMAT = '||p_format;
END;
```

```
EXEC  set_date_format('''DD-MON-RRRR''');
```

Invokers Right & Dynamic SQL

```
CREATE OR REPLACE   PROCEDURE  create_table (p_table_name      VARCHAR2,
                                              p_table_columns  VARCHAR2)
AUTHID CURRENT_USER IS
BEGIN
    EXECUTE IMMEDIATE  'CREATE TABLE '|| p_table_name || p_table_columns;
END create_table;
```

```
CREATE OR REPLACE PROCEDURE delete_table  (p_table_name  VARCHAR2)
AUTHID CURRENT_USER IS
   BEGIN
      EXECUTE IMMEDIATE  'DELETE FROM  '||p_table_name;
      COMMIT;
END delete_table;
```

Database Links With Dynamic SQL

db2 instance

```
create public database link  db1link connect to demo identified by  demo
using 'db1';
```

db2 instance

```
CREATE OR REPLACE PROCEDURE delete_table  (p_table_name  VARCHAR2,
                                            p_dblink      VARCHAR2)  IS
 BEGIN
      EXECUTE IMMEDIATE  'DELETE FROM  '||p_table_name||'@'||p_dblink;
     COMMIT;
END update_record;
```

```
EXEC delete_table  ('accounts', 'db1link') ;
```

SQL Injection

- Statement Modification

```
CREATE OR REPLACE PROCEDURE delete_order(p_column VARCHAR2,
                                         p_value    VARCHAR2 )  IS
l_query VARCHAR2(200);
 BEGIN
   l_query := 'DELETE FROM orders WHERE ' ||
                  p_column ||' = '||p_value ;
   DBMS_OUTPUT.PUT_LINE(l_query);

   EXECUTE IMMEDIATE l_query;
END delete_order;
/
```

```
DELETE
FROM orders
WHERE
order_act_id = 1;
```

```
DELETE
FROM orders
WHERE
order_act_id = 1
 OR 1=1;
```

Normal Execution

```
EXEC delete_order('order_act_id',1);
```

Statement Modification

```
EXEC delete_order(' order_act_id',  ' 1 OR 1=1 ');
```

SQL Injection

- Use Bind Variables

```
CREATE OR REPLACE PROCEDURE delete_order(p_column VARCHAR2,
                                         p_value    VARCHAR2 )  IS
l_query VARCHAR2(200);
 BEGIN
   l_query := 'DELETE FROM orders WHERE ' ||
                  p_column ||' = :l_value';
   DBMS_OUTPUT.PUT_LINE(l_query);

   EXECUTE IMMEDIATE l_query USING p_value;
END delete_order;
/
```

```
DELETE
FROM orders
WHERE
order_act_id = 1;
```

```
ORA-01722:
invalid number
```

Normal Execution

```
EXEC delete_order('order_act_id',1);
```

Statement Modification

```
EXEC delete_order('order_act_id',  ' 1 OR 1=1 ');
```

SQL Injection

- Statement Injection

```
CREATE OR REPLACE PROCEDURE calc(p_condition VARCHAR2) IS
 l_block VARCHAR2(1000);
BEGIN
 l_block :=
        ' BEGIN  IF  '||p_condition||' = "A" THEN proc1; END IF; END; ';
EXECUTE IMMEDIATE l_block;
...
END calc;
```

```
BEGIN
 IF 'A' = 'A' THEN
  proc1;
 END IF;
END;
```

Normal Execution

```
EXEC   calc('"A"');
```

```
BEGIN
 IF 1=1 THEN
  DELETE FROM ORDERS;
 END IF;
 IF 'A' = 'A'  THEN proc1;
 END IF;
END;
```

Statement Injection

```
EXEC   calc('1=1 THEN DELETE FROM orders; END IF; IF "A"');
```

SQL Injection

- Validations

```
CREATE OR REPLACE PROCEDURE calc(p_condition VARCHAR2) IS
 l_block VARCHAR2(1000);
 malicious_attack EXCEPTION;
BEGIN
 IF INSTR(p_condition, ';') > 0 THEN RAISE malicious_attack; END IF;
 IF INSTR(p_condition, 'END IF;') > 0 THEN RAISE malicious_attack; END IF;
 l_block :=
        ' BEGIN  IF  '||p_condition||' = "A" THEN proc1; END IF; END; ';
 EXECUTE IMMEDIATE l_block;
…
EXCEPTION
 WHEN malicious_attack THEN
  DBMS_OUTPUT.PUT_LINE('Suspicious Input '||p_condition);
  RAISE;
END calc;
```

Statement Injection

```
EXEC   calc('1=1 THEN DELETE FROM orders; END IF; IF "A"');
```

Summary

What is Dynamic SQL?

Usage

Execute Immediate

SQL Injection

Dynamic SQL Using DBMS_SQL

Pankaj Jain

@twit_pankajj

pluralsight
hardcore dev and IT training

Why Use DBMS_SQL?

Return Results to Client

Unknown Number of Select Columns

Unknown Number of PlaceHolders

Type of Statements

DML, DDL, Alter Session Statements

Queries

Procedures & Functions

Anonymous Blocks

DBMS_SQL Workflow

Open Cursor

Parse

Bind Variable

Define Column

Execute

Fetch Rows

Variable Value

Column Value

Close Cursor

DBMS_SQL

Owned By SYS

Invoker's Right

Executing DDL & Session Control Statements

Open Cursor

Parse

Bind Variable

Define Column

Execute

Fetch Rows

Variable Value

Column Value

Close Cursor

Executing DDL & Session Control Statements

Open Cursor

Parse

Execute

Close Cursor

Executing DDL Statements

- Open Cursor

```
FUNCTION OPEN_CURSOR RETURN INTEGER;
```

```
FUNCTION OPEN_CURSOR(security_level IN INTEGER) RETURN INTEGER;
```

- 0 No security check
- 1 Userid / Role Parsing Be the Same as Binding / Executing
- 2 Most Secure

```
CREATE OR REPLACE PROCEDURE drop_table (p_table_name VARCHAR2)  IS
   l_sql  VARCHAR2(100);
   l_cursor_id INTEGER;
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
..
..
END drop_table;
```

Executing DDL Statements

- Parse

```
PROCEDURE PARSE( c              IN INTEGER,
                 statement      IN  VARCHAR2,
                 language_flag  IN  INTEGER);
```

- Language Flag
  - V6 , V7, NATIVE, FOREIGN_SYNTAX

```
CREATE OR REPLACE PROCEDURE drop_table (p_table_name VARCHAR2)  IS
   l_sql  VARCHAR2(100);
   l_cursor_id INTEGER;
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
..
..
END drop_table;
```

Executing DDL Statements

- Execute

```
FUNCTION EXECUTE (c IN INTEGER) RETURN INTEGER;
```

- Optional For DDL

```
CREATE OR REPLACE PROCEDURE drop_table (p_table_name VARCHAR2)  IS
   l_sql  VARCHAR2(100);
   l_cursor_id INTEGER;
   l_return  INTEGER;
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
   l_return := DBMS_SQL.EXECUTE(l_cursor_id);
..
END drop_table;
```

Executing DDL Statements

- Close Cursor

```
PROCEDURE CLOSE_CURSOR( c IN OUT INTEGER);
```

```
CREATE OR REPLACE PROCEDURE drop_table (p_table_name VARCHAR2)  IS
   l_sql  VARCHAR2(100);
   l_cursor_id INTEGER;
   l_return  INTEGER;
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
   l_return := DBMS_SQL.EXECUTE(l_cursor_id);
   DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END drop_table;
```

Executing Session Control Statements

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-RRRR';
```

```
CREATE OR REPLACE PROCEDURE alter_format (p_format VARCHAR2)  IS

        l_sql  VARCHAR2(100);
        l_cursor_id INTEGER;
        l_return  INTEGER;
 BEGIN
        l_sql := 'ALTER SESSION SET NLS_DATE_FORMAT = '||p_format;
        l_cursor_id := DBMS_SQL.OPEN_CURSOR;
        DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
        l_return := DBMS_SQL.EXECUTE(l_cursor_id);
        DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END alter_format;
```

```
EXEC alter_format('''DD-MON-RRRR''');
```

Executing DML Statements, Subprograms & Anonymous Blocks

Open Cursor

Parse

Bind Variable

Define Column

Execute

Fetch Rows

Variable Value

Column Value

Close Cursor

Executing DML Statements, Subprograms & Anonymous Blocks

Open Cursor

Parse

Bind Variable

Execute

Variable Value

Close Cursor

Bind Variable

```
DBMS_SQL.BIND_VARIABLE (
  c          IN INTEGER,
  name       IN VARCHAR2,
  value      IN VARCHAR2 CHARACTER SET ANY_CS [,out_value_size IN
                                                          INTEGER]);
```

```
dbms_sql.bind_variable(
  c        IN INTEGER,
  name     IN VARCHAR2,
  value    IN NUMBER);
```

```
dbms_sql.bind_variable(
c        IN INTEGER,
name     IN VARCHAR2,
value    IN DATE);
```

Executing DML Statements

- Insert Statement

```
CREATE OR REPLACE PROCEDURE insert_record (p_table_name    VARCHAR2,
                                           p_col1_name     VARCHAR2,
                                           p_col1_value    NUMBER,
                                           p_col2_name     VARCHAR2,
                                           p_col2_value    NUMBER ) IS
        l_sql  VARCHAR2(100);
        l_cursor_id INTEGER;
        l_return  INTEGER;
 BEGIN
        l_sql := 'INSERT INTO  '||p_table_name || '('||
                 p_col1_name||' , '||
                 p_col2_name||
                 ' )  '||
                 'VALUES( :col1_value,:col2_value)';
        l_cursor_id := DBMS_SQL.OPEN_CURSOR;
        DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
        DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':col1_value', p_col1_value);
        DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':col2_value', p_col2_value);
        l_return := DBMS_SQL.EXECUTE(l_cursor_id);
        DBMS_OUTPUT.PUT_LINE('Rows Processed '||l_return);
        DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
        COMMIT;
END insert_record;
```

Variable Value

```
dbms_sql.variable_value(
c    IN  INTEGER,
name  IN  VARCHAR2,
value OUT VARCHAR2 CHARACTER SET ANY_CS);
```

```
dbms_sql.variable_value(
c    IN  INTEGER,
name  IN  VARCHAR2,
value OUT NUMBER);
```

```
dbms_sql.variable_value(
c    IN  INTEGER,
name  IN  VARCHAR2,
value OUT DATE);
```

Executing DML Statements

- Returning Into Clause

```
DECLARE
      l_item_value   items.item_value%TYPE   := 100;
      l_item_id      items.item_id%TYPE      := 1;
      l_item_name    items.item_name%TYPE;

      l_sql  VARCHAR2(200);
      l_cursor_id INTEGER;
      l_return  INTEGER;
  BEGIN
      l_sql :=  'UPDATE   items SET item_value =  :p_item_val '||
          ' WHERE item_id = :p_item_id RETURNING item_name INTO  :l_name '  ;
      l_cursor_id :=  DBMS_SQL.OPEN_CURSOR;
      DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_val', l_item_value);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_id', l_item_id);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':l_name', l_item_name,60);
      l_return := DBMS_SQL.EXECUTE(l_cursor_id);
      DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':l_name', l_item_name);
      DBMS_OUTPUT.PUT_LINE('Rows Processed '||l_return||'Item Name '||l_item_name);
      DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
      COMMIT;
   END;
```

Executing DML Statements

- Returning Into Clause

```
DECLARE
      l_item_value   items.item_value%TYPE   := 100;
      l_item_id      items.item_id%TYPE      := 1;
      l_item_name    items.item_name%TYPE;

      l_sql  VARCHAR2(200);
      l_cursor_id INTEGER;
      l_return  INTEGER;
  BEGIN
      l_sql :=  'UPDATE   items SET item_value =  :p_item_val '||
          ' WHERE item_id = :p_item_id RETURNING item_name INTO  :l_name '  ;
      l_cursor_id :=  DBMS_SQL.OPEN_CURSOR;
      DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_val', l_item_value);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_id', l_item_id);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':l_name', l_item_name);
      l_return := DBMS_SQL.EXECUTE(l_cursor_id);
      DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':l_name', l_item_name);
      DBMS_OUTPUT.PUT_LINE('Rows Processed '||l_return||'Item Name '||l_item_name);
      DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
      COMMIT;
   END;
```

ORA-6502: PL/SQL: numeric or value error

Executing DML Statements

- Returning Into Clause

```
DECLARE
      l_item_value   items.item_value%TYPE   := 100;
      l_item_id      items.item_id%TYPE      := 1;
      l_item_name    items.item_name%TYPE := 'Maximum Item Length Name ';

      l_sql  VARCHAR2(200);
      l_cursor_id INTEGER;
      l_return  INTEGER;
  BEGIN
      l_sql :=  'UPDATE   items SET item_value =  :p_item_val '||
          ' WHERE item_id = :p_item_id RETURNING item_name INTO  :l_name '  ;
      l_cursor_id :=  DBMS_SQL.OPEN_CURSOR;
      DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_val', l_item_value);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_id', l_item_id);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':l_name', l_item_name);
      l_return := DBMS_SQL.EXECUTE(l_cursor_id);
      DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':l_name', l_item_name);
      DBMS_OUTPUT.PUT_LINE('Rows Processed '||l_return||'Item Name '||l_item_name);
      DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
      COMMIT;
   END;
```

Executing Procedures

```
CREATE OR REPLACE PROCEDURE calculate_tier
(p_act_id    IN        accounts.act_id%TYPE,
 p_act_bal  IN OUT accounts.act_bal%TYPE,
 p_tier             OUT NUMBER) IS
    ....
    ....
 END calculate_tier;
```

```
DECLARE
  l_act_id accounts.act_id%TYPE := 1;
  l_act_bal accounts.act_bal%TYPE;
  l_tier NUMBER;
  l_sql  VARCHAR2(200);
  l_cursor_id INTEGER;
  l_return  INTEGER;

BEGIN
  l_sql :=  ' CALL calculate_tier(:act_id,:act_bal,:tier) ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':act_id', l_act_id);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':act_bal', l_act_bal);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':tier', l_tier);
  l_return := DBMS_SQL.EXECUTE(l_cursor_id);
  DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':act_bal', l_act_bal);
  DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':tier', l_tier);
  DBMS_OUTPUT.PUT_LINE('Act Bal'||l_act_bal||'Tier: '||l_tier);
  DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

Executing Function With Anonymous Block

```
CREATE OR REPLACE FUNCTION get_tier
(p_act_id    IN        accounts.act_id%TYPE,
 p_act_bal  IN OUT accounts.act_bal%TYPE,
 p_tier             OUT NUMBER)
   RETURN NUMBER IS
     ....

 END get_tier;
```

```
DECLARE
  l_act_id    accounts.act_id%TYPE := 1;
  l_act_bal  accounts.act_bal%TYPE;
  l_tier NUMBER;
  l_out NUMBER;

  l_sql  VARCHAR2(200);
  l_cursor_id INTEGER;
  l_return  INTEGER;

BEGIN
  l_sql:=  ' BEGIN :l_out := get_tier(:act_id,:act_bal,:tier);  END; ';
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
   DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':act_id', l_act_id);
   DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':act_bal', l_act_bal);
   DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':tier', l_tier);
   DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':l_out', l_out);
   l_return := DBMS_SQL.EXECUTE(l_cursor_id);
   DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':act_bal', l_act_bal);
   DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':tier', l_tier);
   DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':l_out', l_out);
   DBMS_OUTPUT.PUT_LINE('Act Bal'||l_act_bal||'Tier:  '||l_tier||
                                          ' l_out '||l_out);
   DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

Executing Select Statements

Open Cursor

Parse

Bind Variable

Define Column

Execute

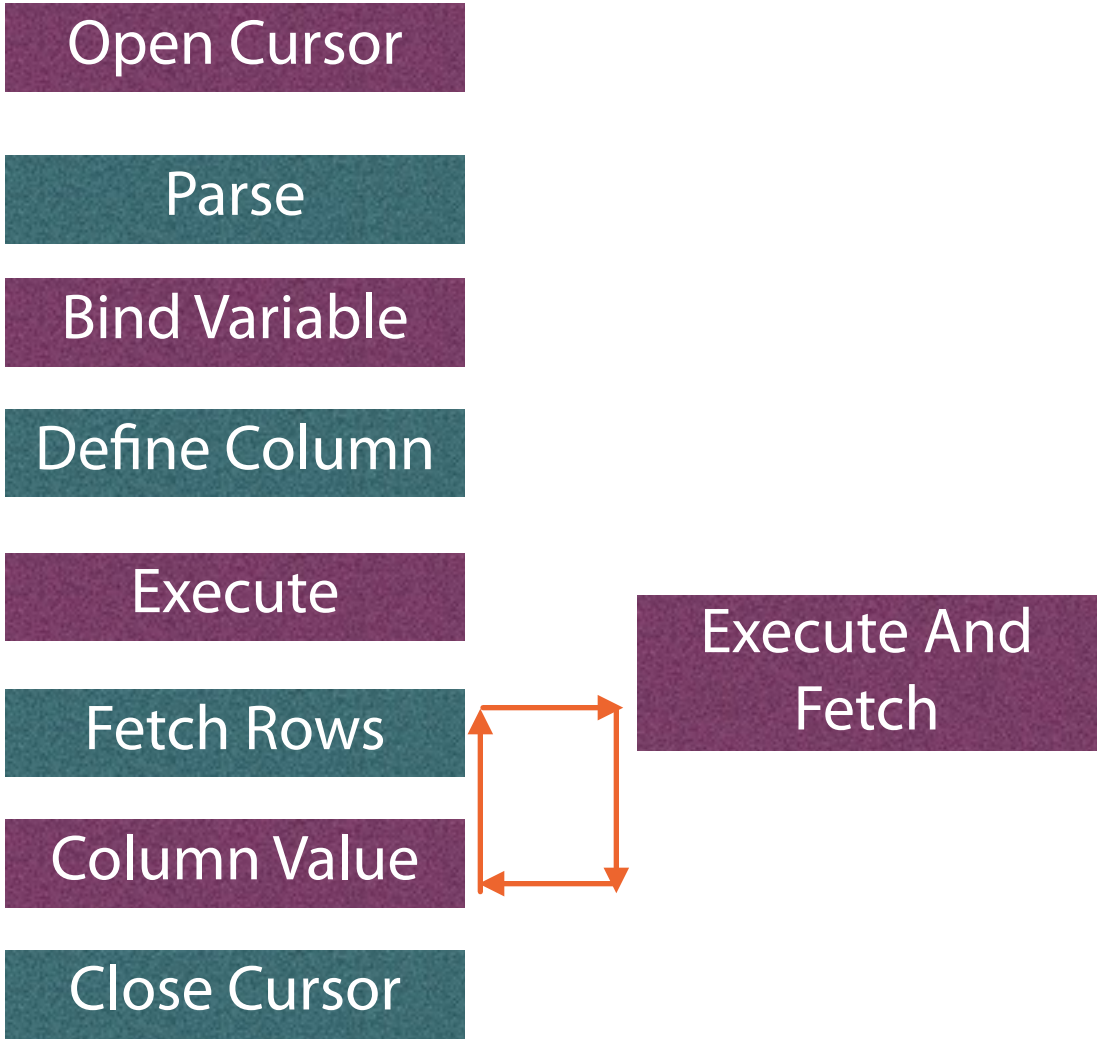
Fetch Rows

Variable Value

Column Value

Close Cursor

Executing Select Statements



Define Column

```
PROCEDURE  define_column(
                    c                IN  INTEGER,
                    position         IN  INTEGER,
                    column           IN VARCHAR2,
                    column_size  IN INTEGER);
```

```
PROCEDURE  define_column(
                    c                IN  INTEGER,
                    position         IN  INTEGER,
                    column           IN NUMBER);
```

```
PROCEDURE  define_column(
                    c                IN  INTEGER,
                    position         IN  INTEGER,
                    column           IN DATE);
```

Column Value

```
PROCEDURE   column_value(
                c                 IN  INTEGER,
                position          IN  INTEGER,
                value             OUT VARCHAR2);
```

```
PROCEDURE   column_value(
                c                 IN  INTEGER,
                position          IN  INTEGER,
                value             OUT VARCHAR2,
                column_error      OUT NUMBER,
                actual_length     OUT NUMBER);
```

Fetch Rows

```
FUNCTION fetch_rows( c  IN  INTEGER) RETURN INTEGER;
```

```
FUNCTION execute_and_fetch( c        IN  INTEGER,
                            exact IN  BOOLEAN DEFAULT FALSE)
        RETURN INTEGER;
```

Multi Row Select

```
DECLARE
 l_item_id          items.item_id%TYPE;
 l_item_name        items.item_name%TYPE;
 l_value            items.item_value%TYPE:= 50;
 l_sql              VARCHAR2(200);
 l_cursor_id        INTEGER;
 l_return           INTEGER;

BEGIN
  l_sql:=  ' SELECT item_id, item_name FROM items WHERE item_value > :p_value ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_value', l_value);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 1, l_item_id);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 2, l_item_name, 100);
  l_return := DBMS_SQL.EXECUTE(l_cursor_id);
  LOOP
    IF DBMS_SQL.FETCH_ROWS(l_cursor_id) = 0 THEN
       exit;
     END IF;
     DBMS_SQL.COLUMN_VALUE(l_cursor_id, 1, l_item_id);
     DBMS_SQL.COLUMN_VALUE(l_cursor_id, 2, l_item_name);
     DBMS_OUTPUT.PUT_LINE('Item Id: ' ||l_item_id|| ' Item Name:  ' ||l_item_name);
   END LOOP;
  DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

Multi Row Select

```
DECLARE
  l_item_id          items.item_id%TYPE;
  l_item_name        items.item_name%TYPE;
  l_value            items.item_value%TYPE:= 50;
  l_sql              VARCHAR2(200);
  l_cursor_id        INTEGER;
  l_return           INTEGER;
BEGIN
  l_sql:=  ' SELECT item_id, item_name FROM items WHERE item_value > :p_value ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_value', l_value);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 1, l_item_id);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 2, l_item_name, 100);
  l_return := DBMS_SQL.EXECUTE(l_cursor_id);
  LOOP
    IF DBMS_SQL.FETCH_ROWS(l_cursor_id) = 0 THEN
       exit;
    END IF;
    DBMS_SQL.COLUMN_VALUE(l_cursor_id, 1, l_item_name);
    DBMS_SQL.COLUMN_VALUE(l_cursor_id, 2, l_item_name);
    DBMS_OUTPUT.PUT_LINE('Item Id: ' ||l_item_id|| ' Item Name:  ' ||l_item_name);
  END LOOP;
  DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

ORA-6562 type of out argument must match type of column or bind variable

LAST_ERROR_POSITION

```
FUNCTION  LAST_ERROR_POSITION RETURN INTEGER;
```

```
DECLARE
  l_errpos INTEGER;
 …
BEGIN
  l_sql:=  ' SELECT item_id, item_name ,  FROM items WHERE item_value > :p_value ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  …
 DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
EXCEPTION
 WHEN OTHERS THEN
    l_errpos := DBMS_SQL.LAST_ERROR_POSITION;
    DBMS_OUTPUT.PUT_LINE (SQLERRM || ' at pos ' || l_errpos);
    DBMS_SQL.CLOSE_CURSOR (l_cursor_id);
END;
```

```
ora-00936 missing expression at pos 28
```

Array Processing

```
PROCEDURE   bind_array(
                c                 IN  INTEGER,
                name              IN  VARCHAR2,
                table_variable    IN table_datatype);
```

```
PROCEDURE   define_array(
                c                 IN  INTEGER,
                position          IN  INTEGER,
                table_variable    IN table_datatype,
                cnt               IN INTEGER,
                index             IN  INTEGER);
```

DESCRIBE_COLUMNS

```
DBMS_SQL.DESCRIBE_COLUMNS (
                    c          IN     INTEGER,
                    col_cnt    OUT    INTEGER,
                    desc_t     OUT    DESC_TAB);
```

```
DBMS_SQL.DESCRIBE_COLUMNS2 (
                    c          IN     INTEGER,
                    col_cnt    OUT    INTEGER,
                    desc_t     OUT    DESC_TAB2);
```

```
DBMS_SQL.DESCRIBE_COLUMNS3 (
                    c          IN     INTEGER,
                    col_cnt    OUT    INTEGER,
                    desc_t     OUT    DESC_TAB3);
```

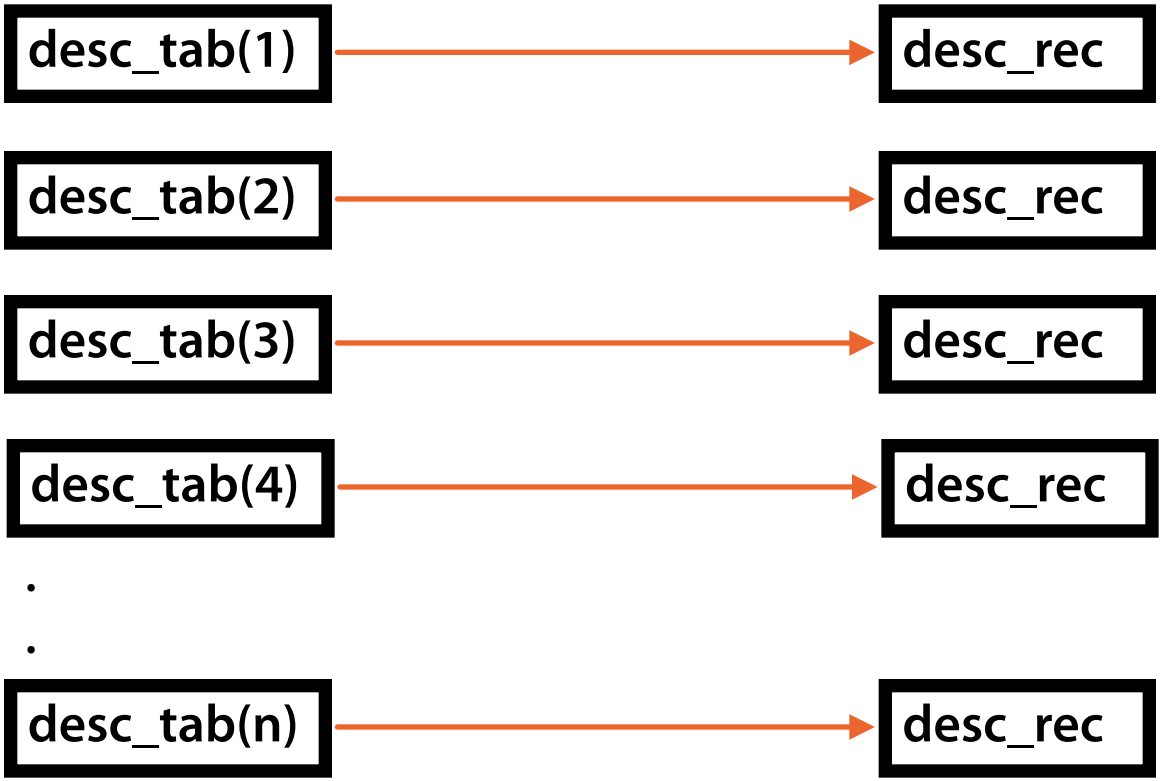
DESC_TAB

```
TYPE desc_tab IS TABLE OF desc_rec INDEX BY BINARY_INTEGER;
```

```
TYPE desc_tab2 IS TABLE OF desc_rec2 INDEX BY BINARY_INTEGER;
```

```
TYPE desc_tab3 IS TABLE OF desc_rec3 INDEX BY BINARY_INTEGER;
```

DESC_TAB


desc_tab(1)
desc_rec
desc_tab(2)
desc_rec
desc_tab(3)
desc_rec
desc_tab(4)
desc_rec
desc_tab(n)
desc_rec

DESC_REC

```
TYPE desc_rec IS RECORD (
  col_type                binary_integer := 0,
  col_max_len             binary_integer := 0,
  col_name                varchar2(32) := '',
  col_name_len            binary_integer := 0,
  col_schema_name         varchar2(32)   := '',
  col_schema_name_len binary_integer := 0,
  col_precision           binary_integer := 0,
  col_scale               binary_integer := 0,
  col_charsetid           binary_integer := 0,
  col_charsetform         binary_integer := 0,
  col_null_ok             boolean        := TRUE);
```

DESC_REC2

```
col_name  varchar2(32767) := ''
```

```
  col_type_name           varchar2(32767)   := '',
  col_type_name_len       binary_integer := 0);
```

DESC_REC3

Column Types

Column Type	Value
VARCHAR2	1
NVARCHAR	1
NUMBER	2
INTEGER	2
LONG	8
ROWID	11
DATE	12
RAW	23
LONG RAW	24
CHAR	96
NCHAR	96
MLSLABEL	106
CLOB (Oracle8)	112
NCLOB (Oracle8)	112
BLOB (Oracle8)	113
BFILE (Oracle8)	114
Object Type (Oracle8)	121
Nested Table Type (Oracle8)	122
Variable Array (Oracle8)	123

Unknown No of Select Columns

```
CREATE OR REPLACE PROCEDURE desc_columns (p_query VARCHAR2) AUTHID DEFINER IS
 l_cursor_id          INTEGER;
 l_no_of_columns  INTEGER;
 l_desc_tab2          DBMS_SQL.DESC_TAB2;
 l_desc_rec2          DBMS_SQL.DESC_REC2;
BEGIN
 l_cursor_id := DBMS_SQL.OPEN_CURSOR;
 dbms_sql.parse(l_cursor_id, p_query, DBMS_SQL.NATIVE);
 DBMS_SQL.DESCRIBE_COLUMNS2(l_cursor_id, l_no_of_columns, l_desc_tab2);
  FOR i IN 1 .. l_no_of_columns LOOP
    l_desc_rec2 := l_desc_tab2(i);
    DBMS_OUTPUT.PUT_LINE('Column Name '||l_desc_rec2.col_name);
    DBMS_OUTPUT.PUT_LINE('Column Type '||l_desc_rec2.col_type);
 END LOOP;
 DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END desc_columns;
```

```
EXEC desc_columns('SELECT order_act_id, item_name FROM orders, items WHERE order_item_id = item_id');
```

```
Column Name ORDER_ACT_ID
Column Type 2
Column Name ITEM_NAME
Column Type 1
```

Unknown No of Select Columns

```
CREATE OR REPLACE PROCEDURE desc_columns (p_query VARCHAR2, p_key VARCHAR2, p_value VARCHAR2)
AUTHID DEFINER IS
 l_cursor_id          INTEGER;
 l_return             INTEGER;
 l_no_of_columns      INTEGER;
 l_desc_tab2          DBMS_SQL.DESC_TAB2;
 l_desc_rec2          DBMS_SQL.DESC_REC2;
 l_number             NUMBER;
 l_date               DATE;
 l_varchar2           VARCHAR2(100);
BEGIN
 l_cursor_id := DBMS_SQL.OPEN_CURSOR;
 DBMS_SQL.parse(l_cursor_id, p_query, DBMS_SQL.NATIVE);
 DBMS_SQL.DESCRIBE_COLUMNS2(l_cursor_id, l_no_of_columns, l_desc_tab2);
 -- Define columns
 FOR i IN 1 .. l_no_of_columns LOOP
     l_desc_rec2 := l_desc_tab2(i);
     IF l_desc_rec2.col_type = 2 THEN
       DBMS_SQL.DEFINE_COLUMN(l_cursor_id, i, l_number);
     ELSIF l_desc_rec2.col_type = 12 THEN
       DBMS_SQL.DEFINE_COLUMN(l_cursor_id, i, l_date);
    ELSE
      DBMS_SQL.DEFINE_COLUMN(l_cursor_id, i, l_varchar2, 100);
    END IF;
 END LOOP;
 DBMS_SQL.BIND_VARIABLE(l_cursor_id, p_key, p_value);
  l_return := DBMS_SQL.EXECUTE(l_cursor_id);
……
END desc_columns;
```

DBMS_SQL Security Aspects

- **Invalid Cursor Check**

Demo User Session

```
DECLARE
 …
BEGIN
  l_sql:= ' SELECT item_id, item_name FROM items WHERE ' ||
            ' item_value > :p_value ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  dbms_output.put_line('Cursor id is '||l_cursor_id);
  …
  LOOP
   IF DBMS_SQL.FETCH_ROWS(l_cursor_id) = 0 THEN
     exit;
   END IF;
   DBMS_SQL.COLUMN_VALUE(l_cursor_id, 1, l_item_name);
   …
   END LOOP;
   CLOSE_CURSOR;
 EXCEPTION
  WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Inside Exception Section '||SQLERRM);
  RAISE;
END;
```

1655307019

Hacker

```
DECLARE
  l_sql VARCHAR2(200);
  l_return INTEGER;
BEGIN
   l_sql:= ' DELETE FROM items ';
   DBMS_SQL.PARSE(1655307019,
                  l_sql, DBMS_SQL.NATIVE);
   l_return :=
DBMS_SQL.EXECUTE(1655307019);
END;
```

ORA-29471: DBMS_SQL access denied

DBMS_SQL Security Aspects

- **Random Cursor Number Generation**

```
FUNCTION OPEN_CURSOR(security_level IN INTEGER) RETURN INTEGER;
```

- **Open Cursor**
  - 0 No security check
  - 1 Userid / Role Parsing Be the Same as Binding / Executing
  - 2 Most Secure
- **Checks**
  - Current Calling User Same As the Recent Parse User
  - Enabled Roles on Current Call Same As Enabled Roles on Recent Parse
  - Container on Current Call Same As Container on Recent Parse

ORA-29470: Effective userid or roles are not the same as when cursor was parsed

DBMS_SQL Security Aspects

Demo User Session

```
CREATE OR REPLACE FUNCTION get_count_cursor RETURN
NUMBER AUTHID DEFINER IS
  l_sql VARCHAR2(200);
  l_cursor_id INTEGER;
 BEGIN
  l_sql:= ' SELECT count(*) FROM orders ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR(2);
  DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
  RETURN l_cursor_id;
 END get_count_cursor;
```

Test

```
DECLARE
  l_cursor_id INTEGER;
  l_count     INTEGER;
  l_return INTEGER;
 BEGIN
  l_cursor_id := demo.get_count_cursor;
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 1, l_count);
  …
 WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE('Inside Exception Section '||SQLERRM);
 RAISE;
END;
```

ORA-29470: Effective userid or roles are not the same as when cursor was parsed

DBMS_SQL vs Native Dynamic SQL

DBMS_SQL

```
CREATE OR REPLACE PROCEDURE drop_table (p_table_name VARCHAR2)  IS
   l_sql  VARCHAR2(100);
   l_cursor_id INTEGER;
   l_return  INTEGER;
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
   l_return := DBMS_SQL.EXECUTE(l_cursor_id);
   DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END drop_table;
```

Native Dynamic SQL

```
CREATE OR REPLACE PROCEDURE drop_table  (p_table_name  VARCHAR2) IS
   l_sql  VARCHAR2(100);
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   EXECUTE IMMEDIATE  l_sql ;
END drop_table;
```

DBMS_SQL vs Native Dynamic SQL

DBMS_SQL

```
DECLARE
  l_item_id          items.item_id%TYPE;
  l_item_name        items.item_name%TYPE;
  l_value            items.item_value%TYPE:= 50;
  l_sql              VARCHAR2(200);
  l_cursor_id        INTEGER;
  l_return           INTEGER;

BEGIN
  l_sql:=  ' SELECT item_id, item_name FROM items WHERE item_value > :p_value ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_value', l_value);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 1, l_item_id);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 2, l_item_name, 100);
  l_return := DBMS_SQL.EXECUTE(l_cursor_id);
  LOOP
    IF DBMS_SQL.FETCH_ROWS(l_cursor_id) = 0 THEN
       exit;
    END IF;
    DBMS_SQL.COLUMN_VALUE(l_cursor_id, 1, l_item_id);
    DBMS_SQL.COLUMN_VALUE(l_cursor_id, 2, l_item_name);
    DBMS_OUTPUT.PUT_LINE('Item Id: ' ||l_item_id|| ' Item Name:  ' ||l_item_name);
  END LOOP;
  DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

DBMS_SQL vs Native Dynamic SQL

Native Dynamic SQL

```
DECLARE
 l_item_id          items.item_id%TYPE;
 l_item_name        items.item_name%TYPE;
 l_value            items.item_value%TYPE:= 50;
 l_sql              VARCHAR2(200);
 l_ref_cursor       SYS_REFCURSOR;

BEGIN
  l_sql:=  ' SELECT item_id, item_name FROM items WHERE item_value > :p_value ';
  OPEN l_ref_cursor FOR l_sql USING l_value;
   LOOP
      FETCH l_ref_cursor INTO l_item_id, l_item_name;
      DBMS_OUTPUT.PUT_LINE('Item Id: ' ||l_item_id|| ' Item Name:  ' ||l_item_name);
      EXIT WHEN l_ref_cursor%NOTFOUND;
END;
```

When Use DBMS_SQL?

Unknown Number of Select Columns

Unknown Number of PlaceHolders

Interoperability

```
DBMS_SQL.TO_REFCURSOR(cursor_number IN OUT INTEGER)
 RETURN SYS_REFCURSOR;
```

```
DECLARE
  l_cursor_id  INTEGER;
  l_sql            VARCHAR2(200);
  l_ref_cursor  SYS_REFCURSOR;
  l_items_rec    items%ROWTYPE;
BEGIN
  l_sql:=  ' SELECT * FROM items WHERE item_value = :p_item_value';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_value' ,100);
  DBMS_SQL.EXECUTE(l_cursor_id);
  l_ref_cursor := DBMS_SQL.TO_REFCURSOR(l_cursor_id);
   LOOP
     FETCH l_ref_cursor INTO l_items_rec;
     EXIT WHEN l_ref_cursor%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE('Item Name is  '||l_items_rec.item_name);
  END LOOP;
  CLOSE l_ref_cursor;
END;
```

Interoperability

```
DBMS_SQL.TO_CURSOR_NUMBER(rc IN OUT SYS_REFCURSOR)
 RETURN INTEGER;
```

```
CREATE OR REPLACE PROCEDURE getinfo(p_query VARCHAR2) IS
  l_cursor_id      INTEGER;
  l_ref_cursor   SYS_REFCURSOR;
  l_col_count      INTEGER;
  l_desc_tab2      DBMS_SQL.DESC_TAB2;
  l_desc_rec2      DBMS_SQL.DESC_REC2;
  l_return         INTEGER;
BEGIN
  OPEN l_ref_cursor FOR p_query;
  l_cursor_id := dbms_sql.to_cursor_number(l_ref_cursor);
  DBMS_SQL.DESCRIBE_COLUMNS2(l_cursor_id, l_col_count, l_desc_tab2);
          FOR i IN 1 .. l_col_count LOOP
          l_desc_rec2 := l_desc_tab2(i);
          DBMS_OUTPUT.PUT_LINE('Column Name '||l_desc_rec2.col_name);
          DBMS_OUTPUT.PUT_LINE('Column Type '||l_desc_rec2.col_type);
  END LOOP;
  DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

```
EXEC getinfo('SELECT order_item_id, order_act_id FROM orders');
```

Summary

What Is DBMS_SQL?

Usage

Security Implications

Debugging PL/SQL Code

Pankaj Jain

@twit_pankajj

pluralsight
hardcore dev and IT training

Debugging Methods

DBMS_OUTPUT

DBMS_UTILITY

SQL Developer Debugger

DBMS_OUTPUT.PUT_LINE

PUT_LINE (msg IN VARCHAR2);

- **Length of Message**
  - 255 Bytes for Older Versions
  - Version 10.2 and Above: 32767 Bytes
- **Puts End of Line**

```
CREATE OR REPLACE PROCEDURE set_location(p_location VARCHAR2) IS
  l_num NUMBER(1);
BEGIN
  DBMS_OUTPUT.PUT_LINE('Input location is '||p_location);
…
EXCEPTION
  WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Error: '||SQLERRM);
  …
END;
```

```
Input location is WA
Error: ORA-01426: numeric overflow
```

Usage

- **Output Visible Only After Block Ends**
- **SQLPLUS**
  - **set serveroutput on**

Enables the Output

Calls Get Lines Automatically

```
set serveroutput on size 200000
set serveroutput on size unlimited
```

- **SQLDeveloper**

Enables the Output

Calls Get Lines Automatically

Usage

- **Unit Tests**

```
CREATE OR REPLACE PROCEDURE set_location(p_location VARCHAR2, p_debug VARCHAR2 DEFAULT 'N') IS
  l_num NUMBER(1);
BEGIN
 IF p_debug = 'Y'THEN
   DBMS_OUTPUT.PUT_LINE('Input location is '||p_location);
 END IF;
…
EXCEPTION
 WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE('Error: '||SQLERRM);
  …
END;
```

DBMS_UTILITY

- **FORMAT_ERROR_STACK**

Formats Error Stack | 2000 Bytes

- **FORMAT_ERROR_BACKTRACE**

Line Number of Error

```
CREATE OR REPLACE PROCEDURE set_lcn(p_location VARCHAR2) IS

  l_dept NUMBER;
…
 BEGIN
  l_dept := get_dept;
…
EXCEPTION
 WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);


 DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
 …
END set_lcn;
```

```
CREATE OR REPLACE FUNCTION
get_dept  RETURN NUMBER IS
l_id NUMBER;
..
BEGIN
l_id := 'abc';
…
END get_dept;
```

```
ORA-6502: PL/SQL : numeric or
value error : character to
number conversion error
```

```
ORA-6512:  at  "DEMO.GET_DEPT", line 6
ORA-6512:  at  "DEMO.SET_LCN", line 4
```

SQL Developer Debugger

- **Compiling in Debug Mode**

```
ALTER PACKAGE hr_mgmt COMPILE  DEBUG ;
```

```
ALTER PROCEDURE test COMPILE  DEBUG ;
```

```
ALTER FUNCTION get_item_value COMPILE  DEBUG ;
```

- **Privileges**
  - To Be Granted by a DBA

```
GRANT DEBUG CONNECT SESSION TO demo ;
```

```
GRANT DEBUG ANY PROCEDURE TO demo ;
```

Summary

DBMS_OUTPUT

DBMS_UTILITY

SQL Developer Debugger