# 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

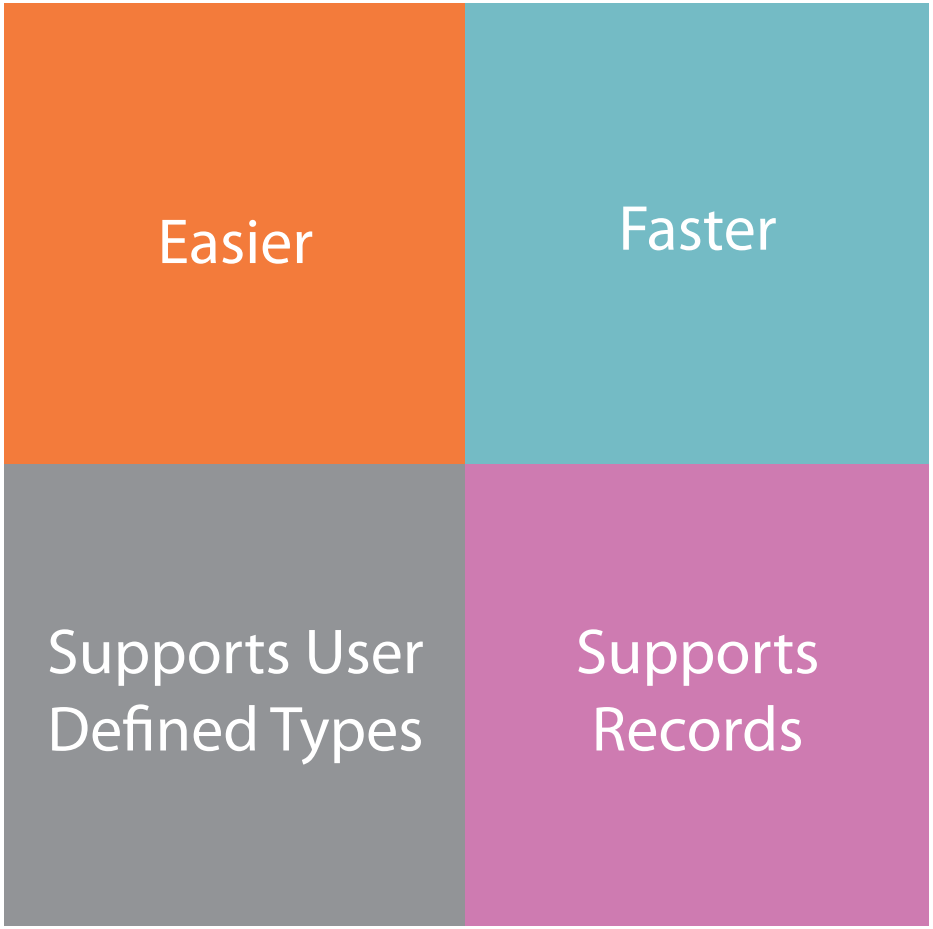
- Dynamic Queries
- Dynamic Sorts
- Dynamic Subprogram Invocation
- Dynamic Optimizations
- DDL
- Frameworks
- Varying Table Definitions

# 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