Oracle PL/SQL Fundamentals - Part 2

Introduction

Pankaj Jain

@twit_pankajj

pluralsight

hardcore dev and IT training

Calling Functions From SQL

Roles & Privs With Subprograms

Local Subprograms

Package Specification

Package Body

Procedures

Functions

Parameters

Named Program Units

Create Reusable Units of Work

Abstract Complex Logic

Performance

Reduce Errors

Pre-requisites

Oracle PL/SQL Fundamentals - Part 1

Equivalent Basic Programming Knowledge

Audience

Oracle Database Programmers

Web Developers

Other Programmers

Tools

Oracle Express Edition

SQL Developer

SQLPLUS

Toad

SQL Navigator

Procedures

Pankaj Jain

@twit_pankajj

What is a Procedure?

- Named Program Unit
- Performs Unit of Work
- Does Not Return Anything

Setup

```
CREATE TABLE departments
(dept_id NUMBER NOT NULL PRIMARY KEY,
 dept_name VARCHAR2(60));
```

```
CREATE TABLE employee
(emp_id NUMBER NOT NULL PRIMARY KEY,
 emp_name VARCHAR2(60),
 emp_dept_id  NUMBER ,
 emp_loc VARCHAR2(2),
 emp_sal  NUMBER,
 emp_status VARCHAR2(1),
 CONSTRAINT emp_dept_fk FOREIGN KEY(emp_dept_id)
 REFERENCES departments(dept_id));
```

Privileges

- CREATE PROCEDURE
- CREATE ANY PROCEDURE
- ALTER ANY PROCEDURE
- EXECUTE

```
GRANT CREATE PROCEDURE TO demo;
GRANT CREATE ANY PROCEDURE TO demo;
GRANT ALTER ANY PROCEDURE TO demo;

GRANT EXECUTE ON <schema_name>.<procedure_name> TO demo;
```

Defining Procedures

```
 CREATE  [OR REPLACE]  PROCEDURE
[schema_name.]<procedure_name> IS | AS

  <declaration section>

BEGIN

    statements;

[EXCEPTION]

END [<procedure_name>];
```

Simple Procedure

```
CREATE OR REPLACE PROCEDURE update_dept  AS
   l_emp_id employee.emp_id%TYPE := 10;
 BEGIN
    UPDATE employee
       SET      emp_dept_id = 2
    WHERE   emp_id = l_emp_id;
    COMMIT;
  EXCEPTION
     WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE(SQLERRM);
       DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
       ROLLBACK;
       RAISE;
  END update_dept;
```

Compiling Procedure

update_dept.sql

```
CREATE OR REPLACE PROCEDURE update_dept  AS
   l_emp_id employee.emp_id%TYPE := 10;
 BEGIN
   UPDATE employee
      SET emp_dept_id = 2
   WHERE emp_id = l_emp_id;
   COMMIT;
 EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(SQLERRM);
     DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
     ROLLBACK;
     RAISE;
 END update_dept;
/
```

```
@C:\Demo\update_dept.sql
```

```
ALTER PROCEDURE update_dept COMPILE;
```

Native Compilation

- **PLSQL_CODE_TYPE**
  - INTERPRETED
  - NATIVE

```
ALTER SESSION SET PLSQL_CODE_TYPE=NATIVE;

ALTER PROCEDURE update_dept COMPILE PLSQL_CODE_TYPE=NATIVE;
```

PL/SQL Optimization Level

- PLSQL_OPTIMIZE_LEVEL
  - 0 Pre 10g Optimization
  - 1   Removed Unnecessary Computations
  - 2   Code Refactoring
  - 3   Code Inlining

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL=2;
```

```
SELECT   PLSQL_OPTIMIZE_LEVEL,
         PLSQL_CODE_TYPE
  FROM  ALL_PLSQL_OBJECT_SETTINGS
WHERE  NAME= 'UPDATE_DEPT';
```

Compile for Debug

- INTERPRETED
- Non-Production Environment
- PLSQL_DEBUG

```
ALTER PROCEDURE update_dept COMPILE DEBUG;
```

```
ALTER SESSION SET PLSQL_DEBUG = FALSE;
```

Errors

Invalid Object Name

Syntax Errors

```
Warning: Procedure created with compilation errors.
```

```
Errors: check compiler log
```

```
SHOW ERRORS;
4/12     PL/SQL: ORA-00942: table or view does not exist
4/5      PL/SQL: SQL Statement ignored
```

Warnings

Severe	Enable
Performance	Disable
Informational	Error

PLW-06002: Unreachable code

```
CREATE OR REPLACE PROCEDURE update_dept  AS
   l_emp_id employee.emp_id%TYPE := 10;
 BEGIN
   UPDATE employee
      SET emp_dept_id = 2
   WHERE emp_id = l_emp_id;
   COMMIT;
 EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(SQLERRM);
     DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
     RAISE;
     ROLLBACK;
 END update_dept;
/
```

Setting Warning Levels

```
ALTER SESSION SET PLSQL_WARNINGS='ENABLE:ALL';
ALTER SESSION SET PLSQL_WARNINGS='DISABLE:ALL';
ALTER SESSION SET PLSQL_WARNINGS=‘ENABLE:PERFORMANCE', ‘ENABLE:SEVERE', 'DISABLE:INFORMATIONAL' ;

ALTER PROCEDURE update_dept
  COMPILE PLSQL_WARNINGS='ENABLE:PERFORMANCE', 'ERROR:SEVERE', 'ERROR:06002'
  REUSE SETTINGS;

SHOW ERRORS;
```

```
Procedure created with compilation warnings.
```

DBMS_WARNING

```
add_warning_setting_cat(warning_category  IN VARCHAR2,
                        warning_value     IN VARCHAR2,
                        scope             IN VARCHAR2);
```

```
get_warning_setting_string;
```

```
call dbms_warning.add_warning_setting_cat('INFORMATIONAL', 'DISABLE', 'SESSION');
call dbms_warning.add_warning_setting_cat('SEVERE', 'ENABLE', 'SESSION');
call dbms_warning.add_warning_setting_cat('PERFORMANCE', 'ENABLE', 'SESSION');
call dbms_warning.add_warning_setting_cat('ALL', 'ENABLE', 'SESSION');

call dbms_warning.add_warning_setting_cat('ALL', 'DISABLE', 'SYSTEM');

SELECT dbms_warning.get_warning_setting_string  FROM dual;
```

Executing Procedure

```
CALL / EXEC[UTE] <procedure_name>;
```

```
call      update_dept();

exec      update_dept;

execute   update_dept;
```

```
BEGIN

   update_dept;

END;
```

Dropping Procedure

```
DROP PROCEDURE <procedure_name>;
```

```
DROP PROCEDURE update_dept;
```

Procedure Termination

- Normal Completion

```
CREATE OR REPLACE PROCEDURE update_dept  AS
   l_emp_id employee.emp_id%TYPE := 10;
 BEGIN
     UPDATE employee
      SET emp_dept_id = 2
   WHERE emp_id = l_emp_id;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Finished Successfully');
 EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
      ROLLBACK;
      RAISE;
 END update_dept;
```

Procedure Termination

- Exception

```
CREATE OR REPLACE PROCEDURE update_dept  AS
   l_emp_id employee.emp_id%TYPE := 10;
 BEGIN
     UPDATE employee
       SET emp_dept_id = 20
    WHERE emp_id = l_emp_id;
     COMMIT;
     DBMS_OUTPUT.PUT_LINE('Finished Successfully');
  EXCEPTION
      WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE(SQLERRM);
       DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
       ROLLBACK;
       RAISE;
  END update_dept;
```

Procedure Termination

- Explicitly

```
CREATE OR REPLACE PROCEDURE update_dept  AS
   l_emp_id employee.emp_id%TYPE := 10;
 BEGIN
     UPDATE   employee
      SET        emp_dept_id = 20
     WHERE    emp_id = l_emp_id;
    COMMIT;
    RETURN;
    DBMS_OUTPUT.PUT_LINE('Finished Successfully');
 EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
      ROLLBACK;
      RAISE;
 END update_dept;
```

Summary

Need for Procedures

Errors & Warnings

Procedure Operations

Functions

Pankaj Jain

@twit_pankajj

What is a Function?

- Stored Subprogram
- Returns Information
- Used in Expressions

Oracle Provided Functions

- Numeric Functions
  - ROUND
  - CEIL
- Character Functions
  - LPAD
  - LTRIM
- DateTime Functions
  - SYSDATE
  - SYSTIMESTAMP

```
SELECT ABS(-123)  FROM DUAL;
```

```
DECLARE
 l_num NUMBER;
BEGIN
 l_num := ABS(-123);
 DBMS_OUTPUT.PUT_LINE(l_num);
END;
```

```
123
```

```
SELECT UPPER('Test')  FROM DUAL;
```

```
DECLARE
 l_char VARCHAR2(4);
BEGIN
 l_char := UPPER('Test');
 DBMS_OUTPUT.PUT_LINE(l_char);
END;
```

```
TEST
```

```
DECLARE
 l_date DATE;
BEGIN
l_date := ADD_MONTHS( TO_DATE('10-FEB-2014','DD-MON-RRRR') ,1) ;
 DBMS_OUTPUT.PUT_LINE(l_date);
END;
```

```
10-MAR-2014
```

Privileges

- CREATE PROCEDURE
- CREATE ANY PROCEDURE
- ALTER ANY PROCEDURE
- EXECUTE

```
GRANT CREATE PROCEDURE TO demo;
GRANT CREATE ANY PROCEDURE TO demo;
GRANT ALTER ANY PROCEDURE TO demo;

GRANT EXECUTE ON <schema>.<procedure_name> TO demo;
```

Defining Functions

```
 CREATE  [OR REPLACE] FUNCTION [schema.]<function_name>

     RETURN <datatype> IS | AS

  <declaration section>

BEGIN

   statements;

RETURN <datatype>;

[EXCEPTION]

END [<function_name>];
```

Simple Function

```
CREATE OR REPLACE FUNCTION get_emp_count  RETURN NUMBER AS
   CURSOR    cur_get_dept_id IS
     SELECT   dept_id
        FROM  departments
      WHERE  dept_name = 'IT';
   l_dept_id  departments.dept_id%TYPE;
   l_count  NUMBER := 0;
 BEGIN
   OPEN cur_get_dept_id;
   FETCH cur_get_dept_id INTO l_dept_id;
   CLOSE cur_get_dept_id;
    SELECT count(*)
        INTO  l_count
        FROM employee
     WHERE  emp_dept_id = l_dept_id;
     RETURN l_count;
 EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
      RAISE;
  END get_emp_count;
```

Compiling Function

get_emp_count.sql

```
CREATE OR REPLACE FUNCTION get_emp_count  RETURN NUMBER AS
   CURSOR    cur_get_dept_id IS
     SELECT   dept_id
        FROM  departments
      WHERE  dept_name = 'IT';
   l_dept_id  departments.dept_id%TYPE;
   l_count  NUMBER := 0;
 BEGIN
   OPEN cur_get_dept_id;
   FETCH cur_get_dept_id INTO l_dept_id;
   CLOSE cur_get_dept_id;
    SELECT count(*)
       INTO  l_count
       FROM employee
     WHERE  emp_dept_id = l_dept_id;
    RETURN l_count;
 EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(SQLERRM);
     DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
     RAISE;
  END get_emp_count;
/
```

```
@C:\Demo\ get_emp_count.sql
```

```
ALTER FUNCTION get_emp_count COMPILE;
```

Compiling Function

- **PLSQL_CODE_TYPE**
  - NATIVE
  - INTERPRETED

```
ALTER SESSION SET PLSQL_CODE_TYPE=NATIVE;

ALTER FUNCTION get_emp_count COMPILE PLSQL_CODE_TYPE=NATIVE;
```

- **PLSQL_OPTIMIZE_LEVEL**
  - 0-3

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL=2;
```

- **Debug Mode**

```
ALTER FUNCTION get_emp_count COMPILE DEBUG;
```

Errors	Syntax Errors
	Invalid Object Name

	Severe	Enable
Warnings	Performance	Disable
	Informational	Error

```
ALTER SESSION SET PLSQL_WARNINGS='ENABLE:ALL';
ALTER FUNCTION get_emp_count
  COMPILE PLSQL_WARNINGS='ENABLE:PERFORMANCE', 'ERROR:SEVERE', 'ERROR:06002'   REUSE SETTINGS;
SHOW ERRORS;
```

```
call dbms_warning.add_warning_setting_cat('ALL', 'ENABLE', 'SESSION');
```

Executing Function

- PL/SQL Block or Subprogram
- SQL Statement

```
DECLARE
   l_return NUMBER;

BEGIN

     l_return := get_emp_count;

END;
```

```
select get_emp_count from dual;
```

```
1
```

Executing Function

```
EXEC[UTE]  :bind_variable := <function_name>;
```

```
VARIABLE    l_return NUMBER;

EXEC        :l_return := get_emp_count;
EXECUTE  :l_return := get_emp_count;

PRINT       :l_return
```

```
1
```

```
VARIABLE    l_return NUMBER;
 BEGIN
   :l_return := get_emp_count;
END;
/
PRINT       :l_return
```

```
1
```

Dropping Function

```
DROP FUNCTION <function_name>;
```

```
DROP FUNCTION get_emp_count;
```

Function Termination

- Normal Completion

```
CREATE OR REPLACE FUNCTION get_emp_count  RETURN NUMBER AS
   CURSOR    cur_get_dept_id IS
     SELECT   dept_id
        FROM  departments
       WHERE  dept_name = 'IT';
   l_dept_id  departments.dept_id%TYPE;
   l_count  NUMBER := 0;
 BEGIN
   OPEN cur_get_dept_id;
   FETCH cur_get_dept_id INTO l_dept_id;
   CLOSE cur_get_dept_id;
    SELECT count(*)
       INTO  l_count
       FROM employee
     WHERE  emp_dept_id = l_dept_id;
    DBMS_OUTPUT.PUT_LINE('Finished Successfully');
   RETURN l_count;
 EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(SQLERRM);
     DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
     RAISE;
  END get_emp_count;
```

Exception Causing Function Termination

```
CREATE OR REPLACE FUNCTION get_emp_count  RETURN NUMBER AS
   CURSOR    cur_get_dept_name IS
     SELECT   dept_name
        FROM  departments
      WHERE  dept_name = 'IT';
   l_dept_id  departments.dept_id%TYPE;
   l_count  NUMBER := 0;
 BEGIN
   OPEN cur_get_dept_name;
   FETCH cur_get_dept_name INTO l_dept_id;
   CLOSE cur_get_dept_name;
    SELECT count(*)
        INTO  l_count
        FROM employee
     WHERE  emp_dept_id = l_dept_id;
    DBMS_OUTPUT.PUT_LINE('Finished Successfully');
   RETURN l_count;
 EXCEPTION
    WHEN OTHERS THEN
     IF cur_get_dept_name%ISOPEN THEN
        CLOSE cur_get_dept_name;
     END IF;
     DBMS_OUTPUT.PUT_LINE(SQLERRM);
     DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
     RAISE;
 END get_emp_count;
```

ORA-06503: PL/SQL:
Function returned without value

Function Termination

- Explicitly

```
CREATE OR REPLACE FUNCTION get_tier  RETURN
NUMBER AS
   l_salary NUMBER := 50000;
 BEGIN
     IF l_salary < 40000 THEN
       RETURN 1;
     ELSIF l_salary < 60000 THEN
       RETURN 2;
     ELSE
       RETURN 3;
      END IF;
    DBMS_OUTPUT.PUT_LINE('Finished Successfully');
  EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      RAISE;
  END get_tier;
```

```
CREATE OR REPLACE FUNCTION get_tier  RETURN
NUMBER  AS
   l_salary NUMBER := 50000;
   l_return NUMBER;
 BEGIN
     IF l_salary < 40000 THEN
       l_return := 1;
     ELSIF l_salary < 60000 THEN
       l_return := 2;
     ELSE
       l_return := 3;
      END IF;
    DBMS_OUTPUT.PUT_LINE('Finished Successfully');
    RETURN l_return;
  EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      RAISE;
  END get_tier;
```

Summary

Need for Functions

Errors & Warnings

Function Operations

Parameters in Procedures & Functions

Pankaj Jain

@twit_pankajj

pluralsight
hardcore dev and IT training

How to Pass Parameters?

```
[schema.]<function_or_procedure_name> (parameter1,….parameterN)
```

```
<param_name> <param_mode><param_datatype> {:= | DEFAULT} def_val
```

```
CREATE OR REPLACE FUNCTION get_tier(p_salary IN NUMBER)  RETURN NUMBER  IS
```

Parameter Modes

IN	OUT	IN OUT

- Multiple
- Overloading

Formal vs Actual Parameters

- **Formal**
  - Declared in Subprogram Specification
  - No Constraints Specified

```
CREATE OR REPLACE PROCEDURE update_emp(p_dept_name IN VARCHAR2)  RETURN NUMBER  IS
```

```
CREATE OR REPLACE PROCEDURE update_emp(p_dept_name IN employee.emp_dept_id%TYPE)
RETURN NUMBER  IS
```

- **Actual**
  - Variable or Expression Passed from Calling Client

```
DECLARE
   l_dept_name VARCHAR2(60):= 'IT';
BEGIN
  update_emp(l_dept_name);
END;
```

IN Mode

- Default Mode
- Read Only

```
DECLARE
  l_emp_id       NUMBER(10) := 50;
  l_dept_name VARCHAR2(60):= 'IT';
  l_status         NUMBER;
BEGIN
 l_status := update_emp(l_emp_id,
                            l_dept_name);
END;
```

```
DECLARE
  l_status          NUMBER;
BEGIN
  l_status := update_emp(50,
                           'IT' );
END;
```

```
CREATE OR REPLACE
   FUNCTION update_emp( p_emp_id IN  NUMBER,
                                p_dept_name VARCHAR2)
   RETURN NUMBER AS
  CURSOR   cur_get_dept_id IS
    SELECT   dept_id
      FROM  departments
     WHERE  dept_name = p_dept_name;
  l_dept_id  departments.dept_id%TYPE;
BEGIN
   p_emp_id := 20;
  OPEN cur_get_dept_id;
  FETCH cur_get_dept_id INTO l_dept_id;
  CLOSE cur_get_dept_id;
  UPDATE   employee
        SET   emp_dept_id = l_dept_id
   WHERE   emp_id = p_emp_id;
   COMMIT;
   RETURN 1;
 EXCEPTION
   WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
    ROLLBACK;
    RETURN 0;
 END update_emp;
```

OUT Mode

Actual Parameter Value Ignored

Read & Write

Cannot Pass Literals or Constants

OUT Mode

```
DECLARE
   l_emp_id       NUMBER(10)    := 50;
   l_dept_id      NUMBER        := 1;
   l_location     VARCHAR2(10) :='CA';
   l_status       NUMBER;
BEGIN
  l_status := update_emp(l_emp_id,
                         l_dept_id,
                   'CA'  l_location );
DBMS_OUTPUT.PUT_LINE('Location '||l_location);
DBMS_OUTPUT.PUT_LINE('Status  '||l_status);
END;
```

```
Location Initially
Location  WA
 Status    1
```

```
CREATE OR REPLACE
    FUNCTION update_emp( p_emp_id  IN    NUMBER,
                         p_dept_id       NUMBER,
                         p_location OUT VARCHAR2)
    RETURN NUMBER AS
BEGIN
   DBMS_OUTPUT.PUT_LINE('Location Initially '||p_location);
   UPDATE        employee
         SET        emp_dept_id = p_dept_id
    WHERE         emp_id          = p_emp_id
     RETURNING  emp_loc   INTO p_location;
    COMMIT;
    RETURN 1;
  EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      DBMS_OUTPUT.PUT_LINE(
                  DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
      ROLLBACK;
      RETURN 0;
  END update_emp;
```

IN OUT Mode

Actual Parameter Value Passed	Read & Write	Cannot be a Literal or a Constant

IN OUT Mode

```
DECLARE
   l_emp_id       NUMBER(10)    := 50;
   l_dept_id      NUMBER        := 1;
   l_location     VARCHAR2(10) :='CA';
   l_status       NUMBER        := -1;
BEGIN
  update_emp(l_emp_id,
             l_dept_id,
             l_location,
             l_status);
DBMS_OUTPUT.PUT_LINE(l_location);
DBMS_OUTPUT.PUT_LINE(l_status);
END;
```

```
p_status Initially -1
Location  WA
 Status     1
```

```
CREATE OR REPLACE
    PROCEDURE update_emp( p_emp_id          IN  NUMBER,
                          p_dept_id             NUMBER,
                          p_location        OUT VARCHAR2,
                          p_status       IN OUT NUMBER)    AS
BEGIN
   DBMS_OUTPUT.PUT_LINE('p_status Initially '|| p_status);
   UPDATE   employee
          SET   emp_dept_id     = p_dept_id
    WHERE   emp_id              = p_emp_id
     RETURNING emp_loc INTO p_location;
    p_status := 1;
   COMMIT;
EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(SQLERRM);
     DBMS_OUTPUT.PUT_LINE(
                  DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
     ROLLBACK;
     p_status := 0;
    END update_emp;
```

Exception Inside Stored Subprogram

- Passed by Reference
  - IN
- Passed by Value
  - OUT & IN OUT

```
DECLARE
  l_emp_id       NUMBER(10) := 50;
  l_dept_id      NUMBER          := 1;
  l_location     VARCHAR2(10) :='CA';
  l_status        NUMBER := -1;
BEGIN
  update_emp(l_emp_id,
                       l_dept_id,
                       l_location,
                       l_status);
DBMS_OUTPUT.PUT_LINE(l_location);
DBMS_OUTPUT.PUT_LINE(l_status);
END;
```

```
WA
 0
```

```
CREATE OR REPLACE
   PROCEDURE update_emp( p_emp_id         IN  NUMBER,
                                         p_dept_id             NUMBER,
                                         p_location      OUT VARCHAR2,
                                         p_status      IN OUT NUMBER) AS
    l_number  NUMBER;
 BEGIN
    UPDATE         employee
          SET          emp_dept_id = p_dept_id
     WHERE          emp_id = p_emp_id
    RETURNING    emp_loc INTO p_location;
    p_status := 1;
    l_number := 'CHAR';
    COMMIT;
 EXCEPTION
   WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
    DBMS_OUTPUT.PUT_LINE(
                  DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
    ROLLBACK;
    p_status := 0;
    p_location := null;
  END update_emp;
```

Exception Inside Stored Subprogram

```
DECLARE
   l_emp_id       NUMBER(10) := 50;
   l_dept_id      NUMBER          := 1;
   l_location     VARCHAR2(10) :='CA';
   l_status        NUMBER := -1;
BEGIN
  update_emp(l_emp_id,
                   l_dept_id,
                   l_location,
                   l_status);
EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(SQLERRM);
     DBMS_OUTPUT.PUT_LINE(l_location);
     DBMS_OUTPUT.PUT_LINE(l_status);
END;
```

```
CA
 -1
```

```
CREATE OR REPLACE
   PROCEDURE update_emp( p_emp_id         IN  NUMBER,
                         p_dept_id             NUMBER,
                         p_location       OUT VARCHAR2,
                         p_status      IN OUT NUMBER) AS
    l_number  NUMBER;
 BEGIN
    UPDATE        employee
          SET         emp_dept_id = p_dept_id
     WHERE          emp_id = p_emp_id
    RETURNING   emp_loc INTO p_location;
    p_status := 1;
    l_number := 'CHAR';
    COMMIT;
EXCEPTION
   WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
    DBMS_OUTPUT.PUT_LINE(
              DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
    ROLLBACK;
    RAISE;
 END update_emp;
```

NOCOPY Hint

```
<parameter_name> <parameter_mode> NOCOPY <parameter_datatype>
```

```
CREATE OR REPLACE
   PROCEDURE update_emp( p_emp_id        IN  NUMBER,
                         p_dept_id           NUMBER,
                         p_location      OUT NOCOPY VARCHAR2,
                         p_status     IN OUT NOCOPY NUMBER)
AS
    l_number  NUMBER;
  BEGIN
    UPDATE       employee
          SET        emp_dept_id = p_dept_id
     WHERE       emp_id = p_emp_id
    RETURNING   emp_loc INTO p_location;
    p_status := 1;
    l_number := 'CHAR';
    COMMIT;
 EXCEPTION
   WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
    DBMS_OUTPUT.PUT_LINE(
              DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
    ROLLBACK;
    RAISE;
 END update_emp;
```

```
DECLARE
  l_emp_id       NUMBER(10) := 50;
  l_dept_id      NUMBER        := 1;
  l_location     VARCHAR2(10) :='CA';
  l_status        NUMBER := -1;
BEGIN
  update_emp(l_emp_id,
                   l_dept_id,
                   l_location,
                   l_status);
 EXCEPTION
   WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
    DBMS_OUTPUT.PUT_LINE(l_location);
    DBMS_OUTPUT.PUT_LINE(l_status);
END;
```

WA
1

NoCopy Restrictions

Implicit **Conversions**	**Scalar** Datatype with **NOT NULL** Constraint	Scalar **Numeric** with Constraints
Records with Different Field **Constraints**	**Elements** of a **Collection**	Remote **Procedure** Calls

Positional Notation

- Parameters Passed by Position
- Compact
- Not Affected by Renaming of Formal Parameters
- Affected by Repositioning of Formal Parameters

```
CREATE OR REPLACE PROCEDURE
     update_emp( p_emp_id          IN   NUMBER,
                 p_dept_id              NUMBER,
                 p_location       OUT  VARCHAR2,
                 p_status      IN OUT  NUMBER)
  …….
  ……..
END update_emp;
```

```
DECLARE
   l_emp_id       NUMBER(10) := 50;
   l_dept_id      NUMBER        := 1;
   l_location     VARCHAR2(10) :='CA';
   l_status       NUMBER := -1;
BEGIN
  update_emp(l_emp_id,
             l_dept_id,
             l_location,
             l_status);
END;
```

Named Notation

- Parameters Passed by Names
- Verbose
- Not Affected by Repositioning of Formal Parameters
- Flexibility for Passing Default Values
- Affected by Renaming of Formal Parameters

```
CREATE OR REPLACE PROCEDURE
     update_emp( p_emp_id        IN   NUMBER,
                 p_dept_id            NUMBER,
                 p_location     OUT  VARCHAR2,
                 p_status    IN OUT  NUMBER)
  …….
  ……..
 END update_emp;
```

```
DECLARE
   l_emp_id       NUMBER(10) := 50;
   l_bonus
   l_dept_id      NUMBER          := 1;
   l_location     VARCHAR2(10) :='CA';
   l_status       NUMBER := -1;
BEGIN
  update_emp(p_location => l_location,
             p_emp_id => l_emp_id,
             p_status    => l_status,
             p_dept_id => l_dept_id);
 END;
```

Mixed Notation

- Positional Parameters First

```
CREATE OR REPLACE PROCEDURE
     update_emp(  p_emp_id        IN   NUMBER,
                  p_dept_id            NUMBER,
                  p_location      OUT  VARCHAR2,
                  p_status     IN OUT  NUMBER)
  …….
  ……..
 END update_emp;
```

```
DECLARE
  l_emp_id       NUMBER(10) := 50;
  l_dept_id      NUMBER        := 1;
  l_location     VARCHAR2(10) :='CA';
  l_status        NUMBER := -1;
BEGIN
  update_emp(l_emp_id,
             l_dept_id,
             p_status     => l_status,
             p_location => l_location);
 END;
```

```
DECLARE
  l_emp_id       NUMBER(10) := 50;
  l_dept_id      NUMBER        := 1;
  l_location     VARCHAR2(10) :='CA';
  l_status        NUMBER := -1;
BEGIN
  update_emp(p_emp_id => l_emp_id,
             p_dept_id => l_dept_id,
             l_status,
             l_location);
 END;
```

✗

```
PLS-00312:  a positional parameter association may not follow a named association
```

Default Values

```
<param_name> <param_mode><param_datatype> {:= | DEFAULT} def_val
```

- IN Mode

```
CREATE OR REPLACE PROCEDURE
     update_info ( p_emp_id     IN     NUMBER  DEFAULT  50,
                   p_dept_id    IN     NUMBER  DEFAULT   1,
                   p_bonus      IN     NUMBER  DEFAULT  10)
  …….
  ……..
 END update_info;
```

```
BEGIN
  update_info;
END;
```

```
DECLARE
   l_emp_id      NUMBER(10) := 30;
   l_dept_id     NUMBER     := 2;
   l_bonus       NUMBER     := 5;
BEGIN
  update_info(l_emp_id,
              l_dept_id,
              l_bonus);
 END;
```

```
DECLARE
   l_emp_id      NUMBER(10) := 30;
BEGIN
  update_info(l_emp_id);
 END;
```

Default Values

```
CREATE OR REPLACE PROCEDURE
     update_emp( p_emp_id    IN     NUMBER  ,
                 p_dept_id   IN     NUMBER  DEFAULT   1,
                 p_bonus     IN     NUMBER  DEFAULT  10)
  …….
  ……..
 END update_emp;
```

```
DECLARE
  l_emp_id       NUMBER(10) := 30;
BEGIN
  update_emp(l_emp_id);
 END;
```

```
DECLARE
  l_emp_id       NUMBER(10) := 30;
  l_dept_id      NUMBER       := 2;
  l_bonus        NUMBER       := 5;
BEGIN
  update_emp(l_emp_id,
             l_bonus);
 END;
```

```
DECLARE
  l_emp_id       NUMBER(10) := 30;
  l_dept_id      NUMBER       := 2;
  l_bonus        NUMBER       := 5;
BEGIN
  update_emp(p_emp_id  => l_emp_id,
             p_bonus   => l_bonus);
 END;
```

Constraints on Formal Parameters

- Acquires from Actual Parameter

```
DECLARE
   l_emp_id       NUMBER(10)    := 50;
   l_location     VARCHAR2(6) NOT NULL :='INIT';
BEGIN
  get_emp_loc(l_emp_id,
                     l_location);
 END;
```

```
CREATE OR REPLACE
   PROCEDURE get_emp_loc   ( p_emp_id      IN  NUMBER  ,
                             p_location    OUT VARCHAR2) AS
   CURSOR get_loc IS
      SELECT emp_loc
        FROM  employee
      WHERE  emp_id = p_emp_id;
   BEGIN
    p_location  := 'NONE';
    OPEN   get_loc;
    FETCH get_loc INTO p_location;
    CLOSE get_loc;
     p_location   := NULL;
END get_emp_loc;
```

Constraints on Formal Parameters

- Acquires from %TYPE

```
DECLARE
   l_emp_id       NUMBER(10)    := 50;
   l_location     VARCHAR2(6) ;
BEGIN
  get_emp_loc(l_emp_id,
                      l_location);
 END;
```

```
CREATE TABLE employee
(emp_id NUMBER NOT NULL PRIMARY KEY,
 emp_name VARCHAR2(60),
 emp_dept_id  NUMBER ,
 emp_loc VARCHAR2(2),
 emp_sal  NUMBER,
 CONSTRAINT emp_dept_fk FOREIGN KEY(emp_dept_id)
 REFERENCES departments(dept_id));
```

```
CREATE OR REPLACE
    PROCEDURE get_emp_loc   (  p_emp_id      IN  employee.emp_id%TYPE  ,
                               p_location    OUT employee.emp_loc%TYPE) AS
    CURSOR get_loc IS
       SELECT emp_loc
        FROM  employee
       WHERE  emp_id = p_emp_id;
    BEGIN
     p_location  := 'NONE';
     OPEN   get_loc;
     FETCH get_loc INTO p_location;
     CLOSE get_loc;
  END get_emp_loc;
```

Constraints on Formal Parameters

- Numeric Subtypes
  - NOT NULL Constraint Inherited
  - Only Range Inherited for Numeric Base Type

```
DECLARE
  SUBTYPE numsubtype IS NUMBER(2) NOT NULL;

    PROCEDURE  testsubtype   ( p_num  IN  numsubtype ) AS
    BEGIN
      DBMS_OUTPUT.PUT_LINE(p_num);
   END testsubtype;

BEGIN
  testsubtype(1234);
  testsubtype(NULL);
END;
```

```
PLS-00567: cannot pass NULL to a NOT NULL constrained formal
parameter
```

Constraints on Formal Parameters

- Character Subtypes
  - NOT NULL Constraint Inherited
  - Size Not Inherited

```
DECLARE
  SUBTYPE charsubtype IS VARCHAR2(2) NOT NULL;

    PROCEDURE  testsubtype    ( p_char  IN  charsubtype ) AS
    BEGIN
      DBMS_OUTPUT.PUT_LINE(p_char);
    END testsubtype;

BEGIN
  testsubtype('TEST');
  testsubtype(NULL);  ✗
END;
```

```
TEST
```

```
PLS-00567: cannot pass NULL to a NOT NULL constrained formal
parameter
```

Summary

Need for Parameters

Parameter Modes

Passing by Reference and Value

Positional & Named Notation

Default Values & Parameter Constraints

Local Subprograms

Pankaj Jain

@twit_pankajj

Local Subprograms

Declaration Section	**Scope** From Point of Declaration to **End** of Block	**End** of Declaration
Parent Block Variables **Visible**	**Eliminate** Repetition	**Multiple**

Local Procedure in a Stored Procedure

```
CREATE OR REPLACE PROCEDURE update_dept(p_emp_id  employee.emp_id%TYPE)
AS
   l_dept_id departments.dept_id%TYPE := 2;

   PROCEDURE display_message(p_location IN VARCHAR2, p_msg VARCHAR2) IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE('***'||p_location||'***');
      DBMS_OUTPUT.PUT_LINE(p_msg);
    END display_message;

 BEGIN
   display_message('Before Updating',  'Input Employee ID:'||p_emp_id);
   UPDATE employee
      SET emp_dept_id = l_dept_id
   WHERE emp_id = p_emp_id;
   display_message('After Updating',  'Rows Updated:' ||SQL%ROWCOUNT);
 EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(SQLERRM);
     DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
     RAISE;
 END update_dept;
```

```
***Before Updating***
 Input Employee ID:10
```

```
***After Updating***
    Rows Updated:1
```

Local Procedure in Anonymous Block

```
DECLARE
   l_emp_id  := 10;
   l_dept_id departments.dept_id%TYPE := 2;

   PROCEDURE display_message(p_location IN VARCHAR2, p_msg VARCHAR2) IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE('***'||p_location||'***');
      DBMS_OUTPUT.PUT_LINE(p_msg);
    END display_message;

 BEGIN
   display_message('Before Updating', 'Input Employee ID:'|| l_emp_id);
   UPDATE employee
      SET emp_dept_id = l_dept_id
   WHERE emp_id = p_emp_id;
   display_message('After Updating', 'Rows Updated:' ||SQL%ROWCOUNT);
 EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
      RAISE;
 END ;
```

```
***Before Updating***
 Input Employee ID:10
```

```
***After Updating***
    Rows Updated:1
```

Local Function in a Stored Procedure

```
CREATE OR REPLACE PROCEDURE determine_tiers  AS
   l_salary NUMBER := 50000;
   l_tier      NUMBER;
   FUNCTION get_tier  RETURN NUMBER IS
     l_return NUMBER;
   BEGIN
       IF l_salary < 40000 THEN
         l_return := 1;
      ELSIF l_salary < 60000 THEN
         l_return := 2;
      ELSE
         l_return := 3;
       END IF;
       RETURN l_return;
   EXCEPTION
      WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE(SQLERRM|| '  ' || DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
       RAISE;
   END get_tier;
 BEGIN
    l_tier := get_tier;          → 2
    l_salary := 120000;
    l_tier := get_tier;          → 3
 EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM|| '  '||DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
      RAISE;
  END determine_tiers;
```

Exception in a Local Subprogram

```
CREATE OR REPLACE PROCEDURE determine_tiers  AS
   l_salary NUMBER := 50000;
   l_tier      NUMBER;
   FUNCTION get_tier  RETURN NUMBER IS
     l_return NUMBER;
   BEGIN
       IF l_salary < 40000 THEN
         l_return := 1;
      ELSIF l_salary < 60000 THEN
         l_return := 'B';
      ELSE
         l_return := 3;
       END IF;
       RETURN l_return;
   EXCEPTION
      WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE(SQLERRM|| '  ' || DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
       RAISE;
   END get_tier;
 BEGIN
    l_tier := get_tier;
    l_salary := 120000;
    l_tier := get_tier;
 EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM|| '  '||DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
      RAISE;
  END determine  tiers;
```

Visibility of Variables

```
CREATE OR REPLACE PROCEDURE determine_tiers  AS
   l_salary NUMBER := 50000;
   l_tier      NUMBER;
   FUNCTION get_tier  RETURN NUMBER IS
     l_salary NUMBER := 30000;
     l_return NUMBER;
   BEGIN
       IF l_salary < 40000 THEN
         l_return := 1;
      ELSIF l_salary < 60000 THEN
         l_return := 2;
      ELSE
         l_return := 3;
       END IF;
       RETURN l_return;
   EXCEPTION
      WHEN OTHERS THEN
       RAISE;
   END get_tier;
 BEGIN
    l_tier := get_tier;          ────────► 1
    l_salary := 120000;
    l_tier := get_tier;          ────────► 1
 EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM|| '  '||DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
      RAISE;
  END determine_tiers;
```

Scope of Variables

```
CREATE OR REPLACE PROCEDURE determine_tiers  AS
   l_salary NUMBER := 50000;
   l_tier      NUMBER;
   PROCEDURE get_tier  IS
   BEGIN
       IF l_salary < 40000 THEN
         l_tier := 1;
      ELSIF l_salary < 60000 THEN
         l_tier := 2;
      ELSE
         l_tier := 3;
       END IF;
  EXCEPTION
      WHEN OTHERS THEN
       RAISE;
   END get_tier;
 BEGIN
    get_tier;
    DBMS_OUTPUT.PUT_LINE(l_tier);      → 2
    l_salary := 120000;
    get_tier;
    DBMS_OUTPUT.PUT_LINE(l_tier);      → 3
 EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM|| '  '||DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
      RAISE;
  END determine_tiers;
```

Summary

Need for Local Subprograms

Exceptions in Local Subprograms

Scope & Visibility of Variables

Package Specification

Pankaj Jain

@twit_pankajj

pluralsight
hardcore dev and IT training

Why Should You Use Packages

Logical Grouping	Global Variables	Session Data
Selective Exposure	Better Application Design	Performance

When Should You Not Use Packages

- Constantly Changing Specifications

Package Structure

Package Specification

Public APIs, Variables & Objects

No Implementations

Can Exist Independently

Package Body

Implementations

Private APIs, Variables & Objects

Cannot Exist Independently

Contents of Package

- Procedures
- Functions
- Cursors
- Types, Variables & Constants
- Records & Collections
- Others

Package Specification

- CREATE PROCEDURE
- CREATE ANY PROCEDURE

```
 CREATE  [OR REPLACE]  PACKAGE
[schema_name.]<package_name> IS | AS

  declarations;

END [<package_name>];
```

Package Specification

```
CREATE OR REPLACE PACKAGE hr_mgmt AS

  g_active_status  CONSTANT  VARCHAR2(1)  := 'A';
  g_inactive_status CONSTANT VARCHAR2(1)  := 'I';

  g_bonus_pct NUMBER;

  dept_not_found_ex EXCEPTION;

  TYPE g_rec IS RECORD(p_profit NUMBER, p_dept_name departments.dept_name%TYPE);

  CURSOR gcur_get_deptid(p_dept_name VARCHAR2) IS
    SELECT dept_id
      FROM departments
     WHERE dept_name = p_dept_name;

  FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER;
  PROCEDURE update_emp(p_emp_id NUMBER, p_dept_name VARCHAR2);

END hr_mgmt;
```

Order of Declaration

- Referenced Items Declared Before Referring Items

```
CREATE OR REPLACE PACKAGE hr_mgmt AS

  g_active_status  CONSTANT  VARCHAR2(1)  := 'A';
  g_inactive_status CONSTANT VARCHAR2(1)  := 'I';
  g_bonus_pct NUMBER;

  dept_not_found_ex EXCEPTION;

  TYPE g_rec IS RECORD(p_profit NUMBER, p_dept_name departments.dept_name%TYPE);

  CURSOR gcur_get_sal(p_emp_id NUMBER) IS
      SELECT dept_id
        FROM departments
      WHERE dept_name = g_active_status;

  FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER;
  PROCEDURE update_emp(p_emp_id NUMBER, p_dept_name VARCHAR2);

END hr_mgmt;
/
```

Compiling Package Specification

hr_mgmt.spc

```
CREATE OR REPLACE PACKAGE hr_mgmt AS

  g_active_status  CONSTANT  VARCHAR2(1)  := 'A';
  g_inactive_status CONSTANT VARCHAR2(1)  := 'I';
  g_bonus_pct NUMBER;
....
....
.....
END hr_mgmt;
/
```

```
@C:\Demo\hr_mgmt.spc
```

```
ALTER PACKAGE hr_mgmt COMPILE SPECIFICATION;
```

Compiling Package Specification

- Errors & Warnings
- Native Compilation
- PLSQL_OPTIMIZE_LEVEL
- Compile for Debug

```
ALTER PACKAGE hr_mgmt COMPILE SPECIFICATION PLSQL_CODE_TYPE=NATIVE;
```

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL=2;
```

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

```
SELECT   PLSQL_WARNINGS,
         PLSQL_OPTIMIZE_LEVEL,
         PLSQL_CODE_TYPE
  FROM
ALL_PLSQL_OBJECT_SETTINGS
WHERE  NAME= 'HR_MGMT';
```

Executing Package Specification

- **Procedures & Functions**
  - Cannot Execute Without Body

```
ORA-04067: not executed, package body “DEMO.HR_MGMT” does not exist
```

```
DECLARE
   l_emp_id       NUMBER(10) := 50;
   l_dept_name VARCHAR2(60):= 'IT';
BEGIN
  demo.hr_mgmt.update_emp(l_emp_id,l_dept_name);
END;
```

```
DECLARE
   l_profit       NUMBER(10) := 100000;
   l_dept_id   NUMBER        :=  1;
   l_bonus           NUMBER;
BEGIN
 l_bonus:=  demo.hr_mgmt.calc_bonus(l_profit, l_dept_id);
END;
```

Executing Package Specification

- Types, Variables, Constants,Cursors etc.

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(demo.hr_mgmt. g_bonus_pct);
END;
```

```
DECLARE
  l_dept_id   NUMBER;
BEGIN
  OPEN   hr_mgmt.gcur_get_deptid(l_dept_name);
  FETCH hr_mgmt.gcur_get_deptid INTO l_dept_id;
  CLOSE hr_mgmt.gcur_get_deptid;
  END;
```

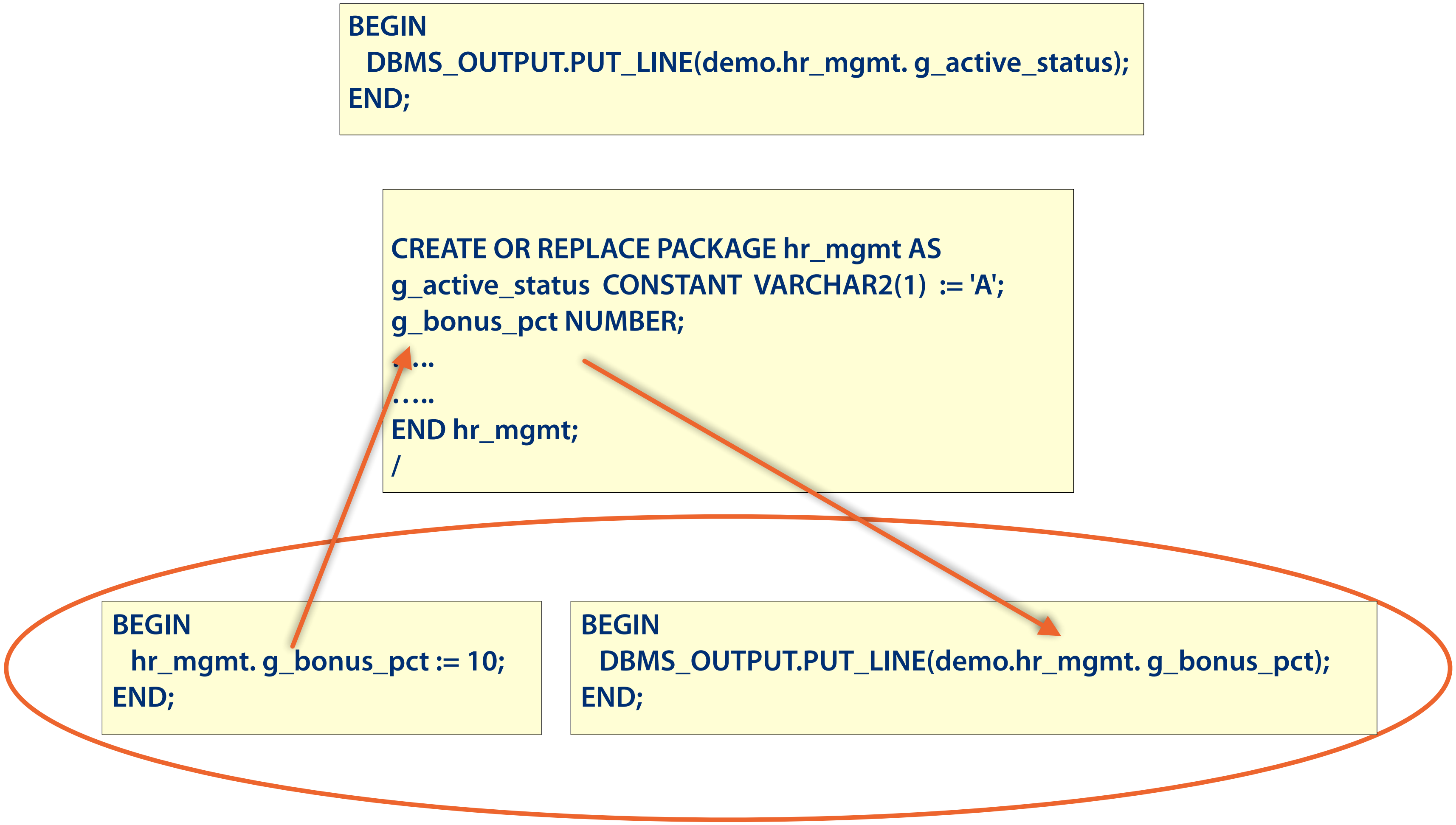
Dropping Package Specification

▪

```
DROP PACKAGE [schema_name.]<package_name> ;
```

```
DROP PACKAGE demo.hr_mgmt;
```

Global Variables & Session State



```
BEGIN
  DBMS_OUTPUT.PUT_LINE(demo.hr_mgmt. g_active_status);
END;
```

```
CREATE OR REPLACE PACKAGE hr_mgmt AS
g_active_status  CONSTANT  VARCHAR2(1)  := 'A';
g_bonus_pct NUMBER;
…..
…..
END hr_mgmt;
/
```

```
BEGIN
  hr_mgmt. g_bonus_pct := 10;
END;
```

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(demo.hr_mgmt. g_bonus_pct);
END;
```

Summary

Need for **Packages**

Package Specification Contents

Order of **Declaration**

Global Variables & Session State

Package Body

Pankaj Jain

@twit_pankajj

Package Body

- Declarations Have Package Body Scope

```
 CREATE  [OR REPLACE]  PACKAGE  BODY
[schema_name.]<package_name> IS | AS

  declarations;

  implementations;

[BEGIN

EXCEPTION]

END [<package_name>];
```

Package Specification

```
CREATE OR REPLACE PACKAGE hr_mgmt AS

  g_active_status  CONSTANT  VARCHAR2(1)  := 'A';
  g_inactive_status CONSTANT VARCHAR2(1)  := 'I';
  g_bonus_pct NUMBER;

  dept_not_found_ex EXCEPTION;

  TYPE g_rec IS RECORD(p_profit NUMBER, p_dept_name departments.dept_name%TYPE);

  CURSOR gcur_get_deptid(p_dept_name VARCHAR2) IS
    SELECT dept_id
      FROM departments
     WHERE dept_name = p_dept_name;

  FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER;
  PROCEDURE update_emp(p_emp_id NUMBER, p_dept_name VARCHAR2);

END hr_mgmt;
/
```

Package Body

```
CREATE OR REPLACE PACKAGE BODY hr_mgmt AS
   CURSOR cur_get_sal(p_dept_id NUMBER ) IS
    SELECT SUM(emp_sal)
      FROM employee
     WHERE emp_dept_id = p_dept_id
      AND  emp_status = g_active_status;
  PROCEDURE  set_bonus(p_profit NUMBER) IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE('Inside set_bonus ');
      IF p_profit < 100000 THEN
        g_bonus_pct := 1;
      ELSE
        g_bonus_pct := 2;
      END IF;
  END set_bonus;
  FUNCTION   get_bonus(p_dept_id NUMBER) RETURN NUMBER IS
    l_sal NUMBER;
    BEGIN
      DBMS_OUTPUT.PUT_LINE('Inside get_bonus ');
      OPEN cur_get_sal(p_dept_id);
      FETCH cur_get_sal INTO l_sal;
      CLOSE cur_get_sal;
      RETURN l_sal * g_bonus_pct;
  END get_bonus;
….
END hr_mgmt;
```

Package Body

```
FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER IS
  BEGIN
    set_bonus(p_profit);
    return get_bonus(p_dept_id);
  EXCEPTION
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(SQLERRM ||' '||DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
     RAISE;
  END calc_bonus;
  PROCEDURE update_emp(p_emp_id NUMBER, p_dept_name VARCHAR2) IS
    l_dept_id departments.dept_id%TYPE;
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Inside update_emp ');
    OPEN gcur_get_deptid(p_dept_name);
    FETCH gcur_get_deptid INTO l_dept_id;
    CLOSE gcur_get_deptid;
    IF l_dept_id IS NULL THEN
      RAISE dept_not_found_ex;
    END IF;
    UPDATE employee
      SET emp_dept_id = l_dept_id  WHERE emp_id = p_emp_id;
    COMMIT;
  EXCEPTION
   WHEN dept_not_found_ex THEN
     DBMS_OUTPUT.PUT_LINE('Invalid dept name '||p_dept_name);
     RAISE;
  END update_emp;
END hr_mgmt;
```

Package Initialization

- First Time Package Called in the Session
- Optional
- At the End

```
CREATE OR REPLACE PACKAGE BODY hr_mgmt AS
…..
….
    END update_emp;
BEGIN
   g_bonus_pct:= 0;
EXCEPTION
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
     DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
      RAISE;
 END hr_mgmt;
/
```

Package Initialization

- Exception Handler for the Execution Section Only

```
CREATE OR REPLACE PACKAGE BODY hr_mgmt AS
  g_number NUMBER(1) := 12;
…..
….
   END update_emp;
BEGIN
  g_bonus_pct:= 0;
EXCEPTION
  WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
   DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
    RAISE;
 END hr_mgmt;
```

First Execution:

```
ORA-06502: PL/SQL : numeric or value error : number precision too large
```

Second Execution:

```
anonymous block completed
```

Package Initialization

```
CREATE OR REPLACE PACKAGE BODY hr_mgmt AS
     g_number NUMBER(1) := 12;✗
  FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER IS
    BEGIN
    set_bonus(p_profit);
    return get_bonus(p_dept_id);
    …
   END calc_bonus;
   ….
   …
END hr_mgmt;
```

```
CREATE OR REPLACE PACKAGE BODY hr_mgmt AS
   g_number NUMBER(1) ;
   PROCEDURE initialize IS
   BEGIN
     g_number := 12;
  END initialize;
  FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER IS
  BEGIN
    initialize;
   set_bonus(p_profit);
   return get_bonus(p_dept_id);
…
  END calc_bonus;
….
END hr_mgmt;
```

Compiling Package Body

hr_mgmt.pkg

```
CREATE OR REPLACE PACKAGE BODY hr_mgmt AS

  CURSOR cur_get_sal(p_dept_id NUMBER ) IS
   SELECT SUM(emp_sal)
    FROM employee
   WHERE emp_dept_id = p_dept_id
    AND  emp_status = g_active_status;


….
….
…..
END hr_mgmt;
/
```

```
@C:\Demo\hr_mgmt.pkg
```

```
ALTER PACKAGE hr_mgmt COMPILE BODY;
```

Compiling Package Body

- Errors & Warnings
- Native Compilation
- PLSQL_OPTIMIZE_LEVEL
- Compile for Debug

```
ALTER PACKAGE hr_mgmt COMPILE BODY PLSQL_CODE_TYPE=NATIVE;
```

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL=2;
```

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

Compiling Entire Package

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

```
ALTER PACKAGE hr_mgmt COMPILE PACKAGE PLSQL_CODE_TYPE=NATIVE;
```

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

Executing Package Body

- Procedures & Functions in Specification

```
DECLARE
   l_emp_id       NUMBER(10) := 50;
   l_dept_name VARCHAR2(60):= 'IT';
BEGIN
  demo.hr_mgmt.update_emp(l_emp_id,l_dept_name);
END;
```

```
DECLARE
   l_profit      NUMBER(10) := 100000;
   l_dept_id   NUMBER        :=  1;
   l_bonus          NUMBER;
BEGIN
 l_bonus:=  demo.hr_mgmt.calc_bonus(l_profit, l_dept_id);
END;
```

- Local Procedures & Functions Cannot be Called Directly

Executing Package Body

- **Local Types, Variables, Constants, Cursors etc.**
  - Cannot be Called by Outside Clients
  - Only be Referred by Internal Clients

Order of Subprograms in Package Body

- Referenced Items Declared Before Referring Items
- Referenced Local Subprograms Declared Before Referring Subprograms

```
CREATE OR REPLACE PACKAGE BODY hr_mgmt AS
  PROCEDURE  set_bonus(p_profit NUMBER) IS
    …
  END set_bonus;
  FUNCTION   get_bonus(p_dept_id NUMBER) RETURN NUMBER IS
     …
  END get_bonus;
FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER IS
  BEGIN
    set_bonus(p_profit);
    return get_bonus(p_dept_id);
    …
  END calc_bonus;
  PROCEDURE update_emp(p_emp_id NUMBER, p_dept_name VARCHAR2) IS
  …
  END update_emp;
…
END hr_mgmt;
/
```

Forward Declaration

```
CREATE OR REPLACE PACKAGE BODY hr_mgmt AS

   PROCEDURE  set_bonus(p_profit NUMBER);
   FUNCTION     get_bonus(p_dept_id NUMBER) RETURN NUMBER;
   FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER IS
    BEGIN
      set_bonus(p_profit);
      return get_bonus(p_dept_id);
    …
  END calc_bonus;
  PROCEDURE  set_bonus(p_profit NUMBER) IS
    …
  END set_bonus;
  FUNCTION   get_bonus(p_dept_id NUMBER) RETURN NUMBER IS
      …
  END get_bonus;
  PROCEDURE update_emp(p_emp_id NUMBER, p_dept_name VARCHAR2) IS
   …
  END update_emp;
…
END hr_mgmt;
/
```

Stateful & Stateless Packages

- Session State Due to Variables, Constants & Cursors

```
CREATE OR REPLACE PACKAGE hr_mgmt AS

  g_active_status  CONSTANT  VARCHAR2(1)  := 'A';
  g_inactive_status CONSTANT VARCHAR2(1)  := 'I';
  g_bonus_pct NUMBER;

  dept_not_found_ex EXCEPTION;

  TYPE g_rec IS RECORD(p_profit NUMBER, p_dept_name departments.dept_name%TYPE);

  CURSOR gcur_get_deptid(p_dept_name VARCHAR2) IS
   SELECT dept_id
     FROM departments
    WHERE dept_name = p_dept_name;

  FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER;
  PROCEDURE update_emp(p_emp_id NUMBER, p_dept_name VARCHAR2);

END hr_mgmt;
/
```

```
ORA-04068: existing state of packages has been discarded
```

Stateful & Stateless Packages

```
CREATE OR REPLACE PACKAGE hr_mgmt AS
   dept_not_found_ex EXCEPTION;
   TYPE g_rec IS RECORD(p_profit NUMBER, p_dept_name departments.dept_name%TYPE);

  FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER;
  PROCEDURE update_emp(p_emp_id NUMBER, p_dept_name VARCHAR2);
END hr_mgmt;
/
```

```
CREATE OR REPLACE PACKAGE global_state AS

  g_active_status  CONSTANT  VARCHAR2(1)  := 'A';
  g_inactive_status CONSTANT VARCHAR2(1)  := 'I';
  g_bonus_pct NUMBER;

  CURSOR gcur_get_deptid(p_dept_name VARCHAR2) IS
    SELECT dept_id
      FROM departments
     WHERE dept_name = p_dept_name;
END hr_mgmt;
/
```

Overloading

- **Multiple Subprograms with Same Name**
  - Order
  - Number
  - Name
  - Datatype Family

- **Only Packaged Subprograms or Local Subprograms**

Overloading Local Subprograms

```
CREATE OR REPLACE PROCEDURE update_dept(p_emp_id  employee.emp_id%TYPE) AS
   l_dept_id departments.dept_id%TYPE := 2;

   PROCEDURE display_message(p_location IN VARCHAR2, p_msg VARCHAR2) IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE('***'||p_location||'***');
      DBMS_OUTPUT.PUT_LINE(p_msg);
    END display_message;

   PROCEDURE display_message( p_msg VARCHAR2) IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE(p_msg);
    END display_message;

 BEGIN
   display_message('Before Updating', 'Input Employee ID:'||p_emp_id);
   UPDATE employee
      SET emp_dept_id = l_dept_id
   WHERE emp_id = p_emp_id;
   display_message('Finished Successfully');
 EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(SQLERRM);
     DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
     RAISE;
 END update_dept;
```

Overloading Packaged Subprograms

```
CREATE OR REPLACE PACKAGE hr_mgmt AS
..
  TYPE g_rec IS RECORD(p_profit NUMBER, p_dept_name departments.dept_name%TYPE);

  FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER;
  FUNCTION  calc_bonus(p_profit NUMBER, p_dept_name VARCHAR2) RETURN NUMBER;
  FUNCTION  calc_bonus(p_rec g_rec) RETURN NUMBER;
  PROCEDURE update_emp(p_emp_id NUMBER, p_dept_name VARCHAR2);
END hr_mgmt;
```

Overloading Packaged Subprograms

```
…
FUNCTION  calc_bonus(p_profit NUMBER, p_dept_id NUMBER) RETURN NUMBER IS
  BEGIN
    set_bonus(p_profit);
    return get_bonus(p_dept_id);
    …
  END calc_bonus;
FUNCTION  calc_bonus(p_profit NUMBER, p_dept_name VARCHAR2) RETURN NUMBER IS
    l_dept_id departments.dept_id%TYPE;
 BEGIN
    OPEN gcur_get_deptid(p_dept_name ) ;
    FETCH gcur_get_deptid INTO l_dept_id;
    CLOSE gcur_get_deptid;
    RETURN calc_bonus(p_profit,l_dept_id);
    …
  END calc_bonus;

FUNCTION  calc_bonus(p_rec g_rec) RETURN NUMBER IS
  BEGIN
    return calc_bonus(p_rec.p_profit,  p_rec.p_dept_name);
    …
  END calc_bonus;

END hr_mgmt;
/
```

Overloading Rules

- Different Datatype Family

```
CREATE OR REPLACE PACKAGE hr_mgmt AS
..
 PROCEDURE update_emp(p_emp_id NUMBER,              p_dept_name VARCHAR2);

 PROCEDURE update_emp(p_emp_id BINARY_INTEGER, p_dept_name VARCHAR2);    ✓

 PROCEDURE update_emp(p_emp_id BINARY_INTEGER, p_dept_id NUMBER);        ✓

 PROCEDURE update_emp(p_emp_id BINARY_INTEGER, p_dept_name VARCHAR2);    ✗

 PROCEDURE update_emp(p_emp_id BINARY_INTEGER, p_dept_name CHAR);        ✗

 PROCEDURE update_emp(p_emp_id INTEGER,             p_dept_name VARCHAR2);   ✗

...
END hr_mgmt;
/
```

```
PLS-00307: too many declarations of UPDATE_EMP match this call
```

Overloading Rules

- **Different Order**

```
CREATE OR REPLACE PACKAGE hr_mgmt AS
..
  PROCEDURE update_emp(p_emp_id NUMBER,              p_dept_name VARCHAR2);

  PROCEDURE update_emp(p_dept_name VARCHAR2 ,    p_emp_id NUMBER);
…
END hr_mgmt;
/
```

✓

Overloading Rules

- Different Subprogram Type

```
CREATE OR REPLACE PACKAGE hr_mgmt AS
..
  PROCEDURE update_emp(p_emp_id NUMBER,   p_dept_name VARCHAR2);

  FUNCTION   update_emp(p_emp_id NUMBER,    p_dept_name VARCHAR2) RETURN NUMBER;
…
END hr_mgmt;
/
```

✓

Overloading Rules

- Different Parameter Name

```
CREATE OR REPLACE PACKAGE hr_mgmt AS
..
 PROCEDURE update_emp(p_emp_id NUMBER,   p_dept_name VARCHAR2);

  PROCEDURE update_emp(p_empid  NUMBER,   p_deptname VARCHAR2);
...
END hr_mgmt;
/
```
✓

- Use Named Notation

```
BEGIN

  hr_mgmt .update_emp(p_emp_id => 1,   p_dept_name =>'IT');   ✓

  hr_mgmt .update_emp( 1,  'IT');   ✗

END hr_mgmt;
```

Overloading Rules

- Mode

```
CREATE OR REPLACE PACKAGE hr_mgmt AS

  PROCEDURE update_emp(p_emp_id   IN   NUMBER );

  PROCEDURE update_emp(p_emp_id  OUT NUMBER );  ✗

END hr_mgmt;
/
```

```
PLS-00307: too many declarations of UPDATE_EMP match this call
```

Overloading Rules

- Default Values

```
CREATE OR REPLACE PACKAGE hr_mgmt AS

  PROCEDURE update_emp(p_emp_id NUMBER,   p_dept_name VARCHAR2 DEFAULT 'IT' );

  PROCEDURE update_emp(p_empid  NUMBER );   ✓

END hr_mgmt;
/
```

- Use Named Notation

```
BEGIN

  hr_mgmt.update_emp(p_emp_id => 1); ✓

  hr_mgmt.update_emp( 1);  ✗

END hr_mgmt;
```

```
PLS-00307: too many declarations of UPDATE_EMP match this call
```

Overloading Considerations

- Explicit Datatypes
- Numeric Datatypes
  - BINARY_DOUBLE
  - BINARY_FLOAT
  - NUMBER
  - BINARY_INTEGER / PLS_INTEGER

```
CREATE OR REPLACE PACKAGE hr_mgmt AS
   PROCEDURE overload(p_emp_id     BINARY_INTEGER );
   PROCEDURE overload(p_emp_id     NUMBER );
  PROCEDURE  overload(p_emp_id     BINARY_FLOAT );
  PROCEDURE  overload(p_emp_id      BINARY_DOUBLE );
END hr_mgmt;
/
```

```
BEGIN
  hr_mgmt.overload(1);
END;
```

```
BEGIN
  hr_mgmt.overload(1.1);
END;
```

```
DECLARE
 l_numeric  BINARY_DOUBLE := 1;
BEGIN
  hr_mgmt.overload(l_numeric);
END;
```

```
BEGIN
  hr_mgmt.overload(TO_BINARY_DOUBLE(1));
END;
```

Overloading Considerations

- **Numeric Datatypes**
  - BINARY_DOUBLE
  - BINARY_FLOAT
  - NUMBER
  - BINARY_INTEGER / PLS_INTEGER

```
CREATE OR REPLACE PACKAGE hr_mgmt AS
   PROCEDURE overload(p_first   NUMBER,   p_second   BINARY_DOUBLE );
   PROCEDURE overload(p_first   NUMBER,   p_second   BINARY_FLOAT );
END hr_mgmt;
/
```

```
DECLARE
 l_first      NUMBER := 1;
 l_second  NUMBER := 1;
BEGIN
  hr_mgmt.overload(l_first, l_second);
END;
```

Overloading Considerations

- Character Literal
  - BINARY_DOUBLE
  - BINARY_FLOAT
  - NUMBER

```
CREATE OR REPLACE PACKAGE hr_mgmt AS
   PROCEDURE overload(p_emp_id     BINARY_INTEGER );
   PROCEDURE overload(p_emp_id     NUMBER );
   PROCEDURE  overload(p_emp_id     BINARY_FLOAT );
   PROCEDURE  overload(p_emp_id      BINARY_DOUBLE );
END hr_mgmt;
/
```

```
BEGIN
  hr_mgmt.overload('1');
END;
```

Summary

Package Body Structure

Order of Declaration

Stateful & Stateless Packages

Overloading

Calling Functions From SQL

Pankaj Jain

@twit_pankajj

Why Call Functions From SQL?

Increases Efficiency of SQL

Extends SQL

Parallel Query Execution

Where Can They Appear?

- Select List of a Query

```
SELECT UPPER('test') FROM dual;

SELECT emp_id,
        get_dept_name(emp_dept_id)
  FROM employee;
```

- Conditions
  - WHERE
  - HAVING

```
SELECT count(*)
  FROM employee
WHERE  get_dept_name(emp_dept_id) = 'IT';
```

```
UPDATE employee
      SET emp_loc = ‘WA’
WHERE  get_dept_name(emp_dept_id) = 'IT';
```

Where Can They Appear?

- INSERT Statement
  - VALUES

```
INSERT INTO employee (emp_id,
                      emp_name,
                      emp_dept_id  )
  VALUES              (  5,
                        ‘John’,
                        get_dept_id('IT'));
```

- Update Statement
  - SET Clause

```
UPDATE   employee
     SET    emp_dept_id =  get_dept_id('IT')
WHERE    emp_id = 10;
```

Where Can They Appear?

- **Others**
  - ORDER BY
  - GROUP BY
  - CONNECT BY / START WITH

```
  SELECT   count(*),
           get_dept_name(emp_dept_id),
    FROM   employee
GROUP BY   get_dept_name(emp_dept_id);
```

Where Can They Not Appear?

DEFAULT Value of a Column

CHECK Constraint Clause

Restrictions

Schema or **Package** Level Function

Formal Parameter in **IN Mode**

Formal Parameter & Return Value **Built-in Datatype**

Restrictions

Select Statement

Cannot Alter System or Session

Cannot Run a DML Statement

No Transaction Statements

```
SELECT emp_id,
        get_dept_name(emp_dept_id)
 FROM employee;
```

```
CREATE OR REPLACE
FUNCTION get_dept_name(p_dept_id in NUMBER) RETURN VARCHAR2   AS
   CURSOR cur_get_dept_name IS
       SELECT dept_name
         FROM  departments
       WHERE  dept_id = p_dept_id;
       l_dept_name departments.dept_name%TYPE;
    BEGIN
      OPEN   cur_get_dept_name;
      FETCH cur_get_dept_name INTO l_dept_name;
      CLOSE cur_get_dept_name;
      UPDATE employee SET emp_dept_id = p_dept_id WHERE emp_id = 10;
      COMMIT;
  RETURN l_dept_name;
END get_dept_name;
```

```
ORA-14551: cannot perform a DML operation inside a query
```

```
ORA-14552: cannot perform a DDL, commit or rollback inside a query or DML
```

Restrictions

DML Statement

Cannot Alter System or Session

No DML for Parallelized DML Statement

DML on the Same Table

No Transaction Statements

```
UPDATE   employee
     SET    emp_dept_id =  get_dept_id('IT')
WHERE    emp_id = 10;
```

```
CREATE OR REPLACE
FUNCTION get_dept_id(p_dept_name in VARCHAR2) RETURN NUMBER  AS
   CURSOR cur_get_dept_id IS
       SELECT dept_id
         FROM  departments
       WHERE  dept_name = p_dept_name;
       l_dept_id departments.dept_id%TYPE;
    BEGIN
     OPEN   cur_get_dept_id;
     FETCH cur_get_dept_id INTO l_dept_id;
     CLOSE cur_get_dept_id;
     UPDATE employee SET emp_dept_id = p_dept_id WHERE emp_id = 10;
     COMMIT;
  RETURN l_dept_id;
END get_dept_id;
```

```
ORA-04091: table EMPLOYEE is mutating, trigger / function may not see it
```

Deterministic Functions

Optimization Technique

Same Result for Same Input Values

Required for Function Based Indexes & Materialized Views

Schema Function

Package Spec Function

Type Specification

Deterministic Functions

```
CREATE OR REPLACE FUNCTION get_tier(p_sal NUMBER)
RETURN NUMBER  AS
   l_return NUMBER;
 BEGIN
     IF p_sal < 40000 THEN
       l_return := 1;
     ELSIF p_sal < 60000 THEN
       l_return := 2;
     ELSE
       l_return := 3;
      END IF;
    DBMS_OUTPUT.PUT_LINE('p_sal '||p_sal);
    RETURN l_return;
  EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      RAISE;
  END get_tier;
```

```
SELECT emp_id,
            emp_sal,
            get_tier(emp_sal) tier
  FROM  employee
  ORDER BY emp_sal;
```

```
EMP_ID            EMP_SAL     TIER
——————————————————————————————————
20                  40000        2
50                  40000        2
10                  70000        3
60                  70000        3
```

```
p_sal 40000
p_sal 40000
p_sal 70000
p_sal 70000
```

Deterministic Functions

```
CREATE OR REPLACE FUNCTION get_tier(p_sal NUMBER)
RETURN NUMBER  DETERMINISTIC AS
   l_return NUMBER;
 BEGIN
     IF p_sal < 40000 THEN
       l_return := 1;
     ELSIF p_sal < 60000 THEN
       l_return := 2;
     ELSE
       l_return := 3;
      END IF;
    DBMS_OUTPUT.PUT_LINE('p_sal '||p_sal);
    RETURN l_return;
  EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      RAISE;
  END get_tier;
```

```
SELECT emp_id,
            emp_sal,
            get_tier(emp_sal) tier
  FROM  employee
  ORDER BY emp_sal;
```

```
EMP_ID            EMP_SAL     TIER
------------------------------------
20                 40000        2
50                 40000        2
10                 70000        3
60                 70000        3
```

```
p_sal 40000
p_sal 70000
```

PARALLEL_ENABLE

```
CREATE OR REPLACE FUNCTION get_tier(p_sal NUMBER)
RETURN NUMBER  PARALLEL_ENABLE AS
   l_return NUMBER;
 BEGIN
     IF p_sal < 40000 THEN
       l_return := 1;
     ELSIF p_sal < 60000 THEN
       l_return := 2;
     ELSE
       l_return := 3;
      END IF;
    DBMS_OUTPUT.PUT_LINE('p_sal '||p_sal);
    RETURN l_return;
  EXCEPTION
     WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      RAISE;
  END get_tier;
```

```
ALTER TABLE employee PARALLEL 3;
```

```
SELECT degree
  FROM  user_tables
WHERE table_name='EMPLOYEE';
```

```
SELECT /*+ parallel (employee,4) */
              emp_id,
              emp_sal,
              get_tier(emp_sal) tier
   FROM  employee
   ORDER BY emp_sal;
```

PRAGMA RESTRICT_REFERENCES

- Prior to Oracle8i
- Use DETERMINISTIC & PARALLEL_ENABLE Instead

Option	Meaning
WNDS	Write No Database State
RNDS	Read No Database State
WNPS	Write No Package State
RNPS	Read No Package State
TRUST	Restrictions Assumed True

PRAGMA RESTRICT_REFERENCES

PRAGMA RESTRICT_REFERENCES (Function_Name,WNDS [, WNPS] [, RNDS] [, RNPS] [, TRUST]);

```
CREATE OR REPLACE PACKAGE hr_mgmt AS

  FUNCTION get_tier(p_sal NUMBER) RETURN NUMBER;
  PRAGMA RESTRICT_REFERENCES (get_tier, WNDS,WNPS,RNDS,RNPS);

  FUNCTION get_diff(p_in NUMBER, p_in2 NUMBER) RETURN NUMBER;
  PRAGMA RESTRICT_REFERENCES (get_diff, WNDS,RNDS);

END hr_mgmt;
```

```
CREATE OR REPLACE PACKAGE BODY hr_mgmt AS

  FUNCTION get_tier(p_sal NUMBER) RETURN NUMBER IS
  BEGIN
    UPDATE employee…..
…..
END hr_mgmt;
```

```
PLS-00452: Subprogram 'GET_TIER''violates its associated pragma
```

Summary

Benefits

Restrictions

Deterministic

Parallel Enable

Roles & Privileges with Subprograms

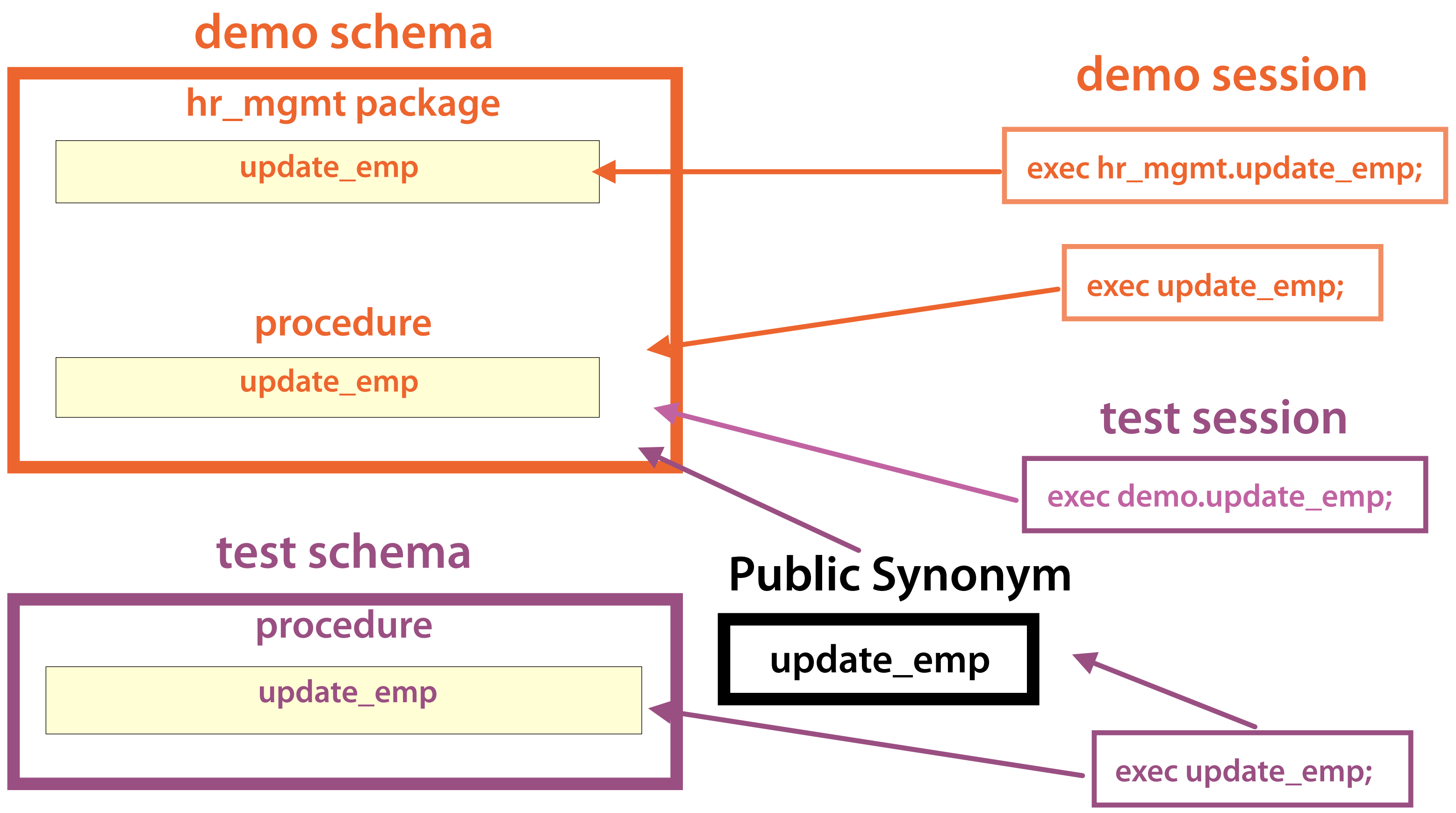
Pankaj Jain

@twit_pankajj

hardcore dev and IT training

Resolution

- Namespace



GRANT execute on update_emp to test;

CREATE PUBLIC SYNONYM update_emp for demo.update_emp;

AUTHID Clause

```
PLW-05018: unit <subprogram_name> omitted optional AUTHID clause: default value DEFINER used
```

- **DEFINER**
  - Default Value
- **CURRENT_USER**

AUTHID Clause

```
AUTHID DEFINER | CURRENT_USER IS | AS
```

- Standalone Procedure

```
CREATE OR REPLACE PROCEDURE update_emp AUTHID DEFINER IS
…
```

- Standalone Function

```
CREATE OR REPLACE FUNCTION get_count RETURN NUMBER AUTHID CURRENT_USER IS
…
```

- Packaged Subprograms

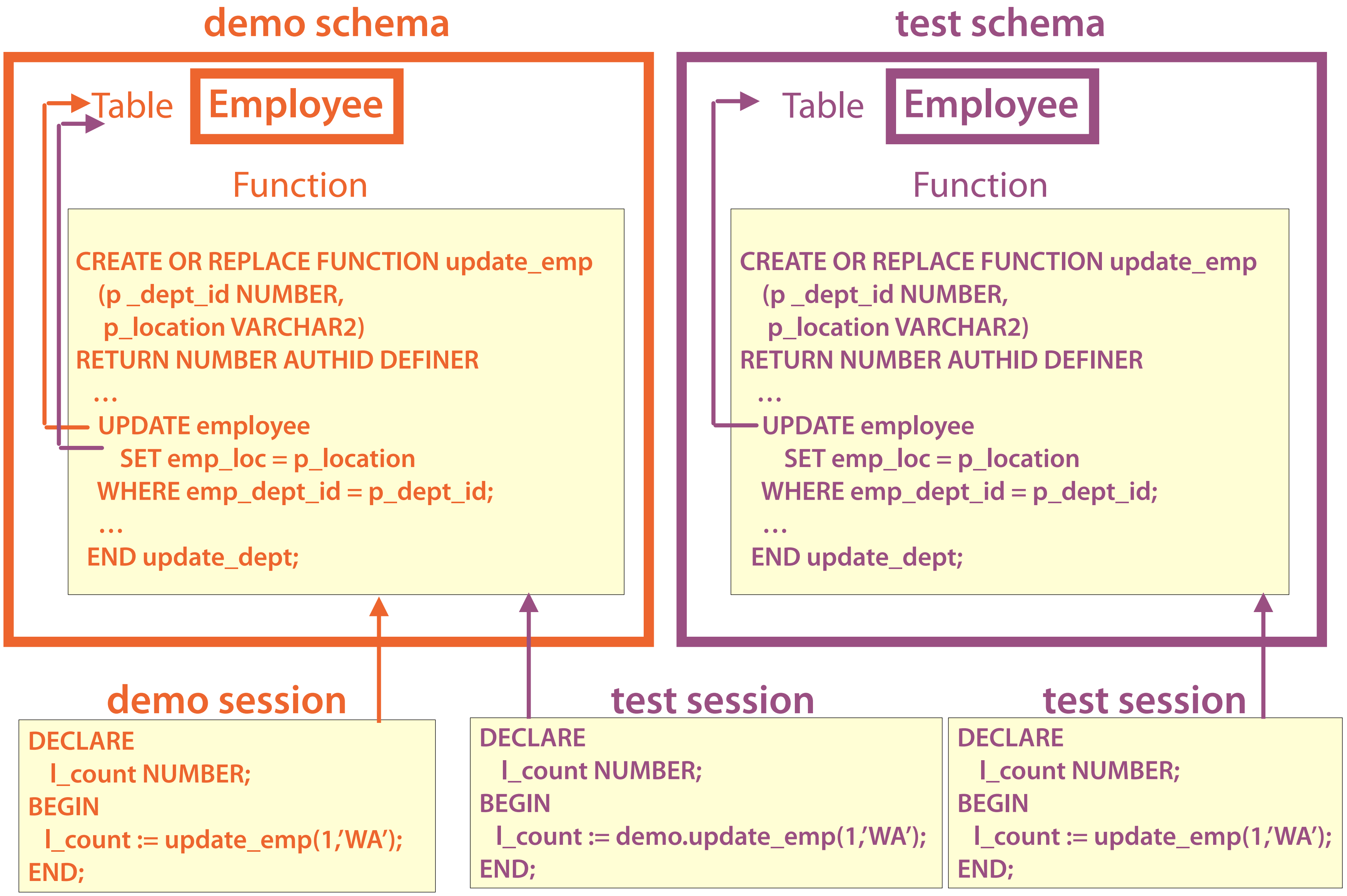
```
CREATE OR REPLACE PACKAGE hr_mgmt  AUTHID CURRENT_USER AS

  FUNCTION get_tier(p_sal NUMBER) RETURN NUMBER;

  PROCEDURE update_emp(p_emp_id NUMBER, p_location VARCHAR2) RETURN NUMBER;

END hr_mgmt;
```

AUTHID DEFINER

- Default Value
- External References Resolved in the Schema of the Owner

```
CREATE OR REPLACE FUNCTION update_emp(p_dept_id NUMBER,
                                      p_location VARCHAR2) RETURN NUMBER AUTHID DEFINER  IS
   l_count NUMBER;
 BEGIN
   UPDATE employee
      SET emp_loc = p_location
   WHERE emp_dept_id = p_dept_id;
   COMMIT;
    RETURN SQL%ROWCOUNT;
  EXCEPTION
     WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
       ROLLBACK;
       RAISE;
  END update_emp;
/
```

AUTHID DEFINER



```
CREATE OR REPLACE FUNCTION update_emp
   (p _dept_id NUMBER,
    p_location VARCHAR2)
RETURN NUMBER AUTHID DEFINER
  …
  UPDATE employee
      SET emp_loc = p_location
  WHERE emp_dept_id = p_dept_id;
   …
 END update_dept;
```

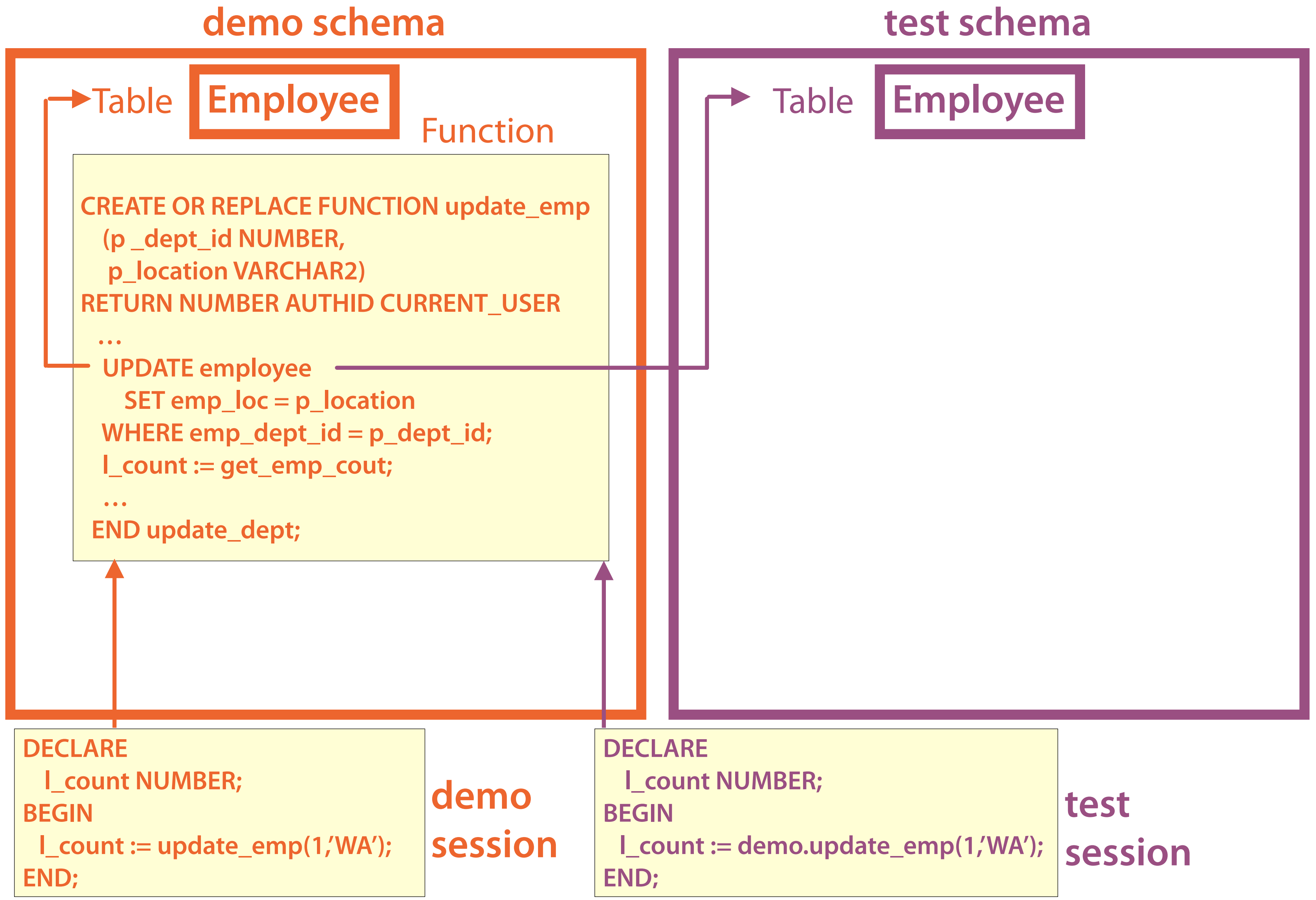
```
CREATE OR REPLACE FUNCTION update_emp
   (p _dept_id NUMBER,
    p_location VARCHAR2)
RETURN NUMBER AUTHID DEFINER
  …
  UPDATE employee
      SET emp_loc = p_location
  WHERE emp_dept_id = p_dept_id;
   …
 END update_dept;
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := update_emp(1,'WA');
END;
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := demo.update_emp(1,'WA');
END;
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := update_emp(1,'WA');
END;
```

AUTHID CURRENT_USER



```
CREATE OR REPLACE FUNCTION update_emp
   (p _dept_id NUMBER,
    p_location VARCHAR2)
RETURN NUMBER AUTHID CURRENT_USER
  …
   UPDATE employee
      SET emp_loc = p_location
   WHERE emp_dept_id = p_dept_id;
   l_count := get_emp_cout;
   …
 END update_dept;
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := update_emp(1,'WA');
END;
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := demo.update_emp(1,'WA');
END;
```

External References for AUTHID CURRENT_USER

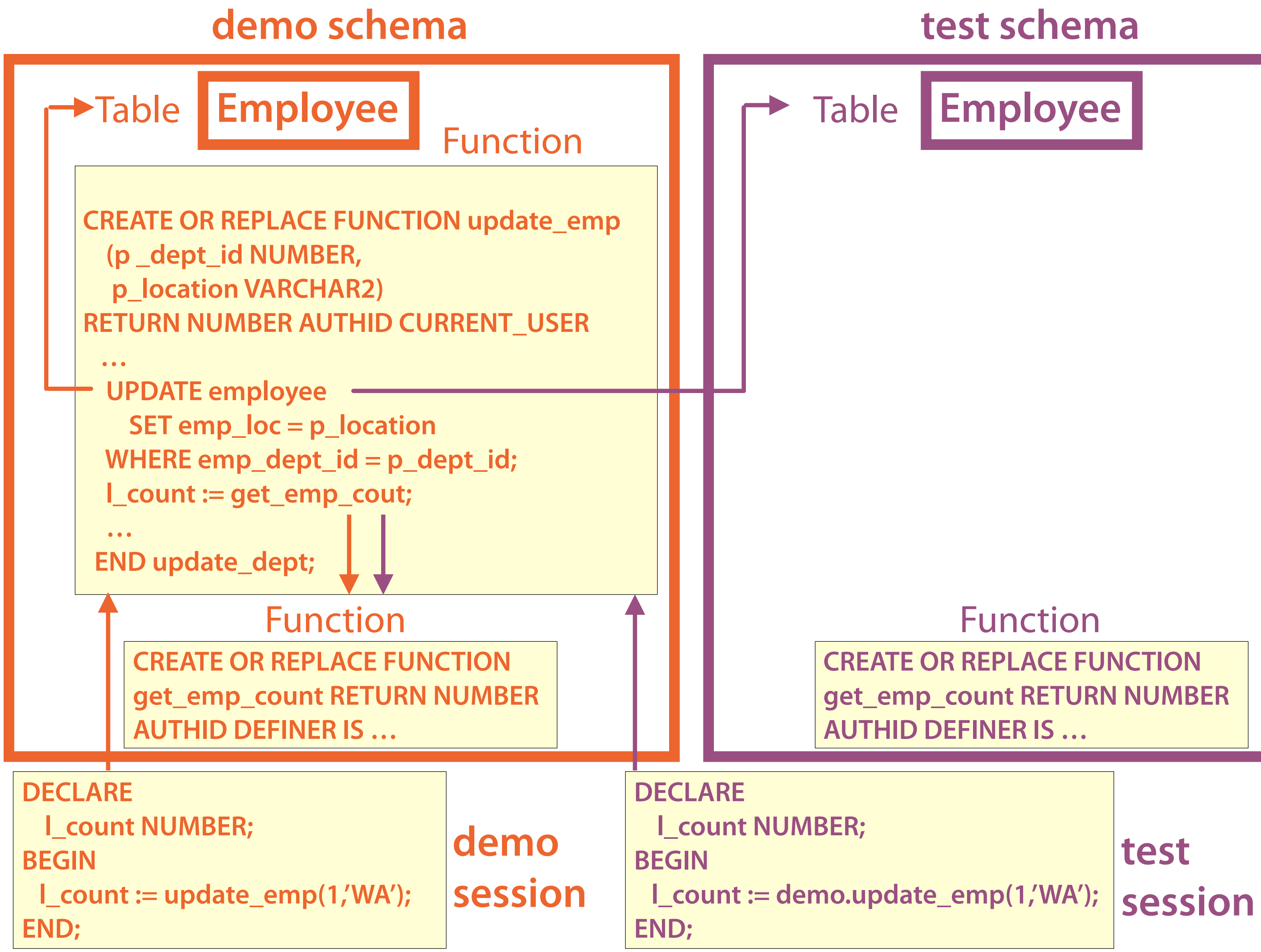
DML Statements

Open & Open for Cursor Statements

Dynamic SQL Statements

Lock Table Statements

External References for AUTHID CURRENT_USER



```
CREATE OR REPLACE FUNCTION update_emp
   (p _dept_id NUMBER,
    p_location VARCHAR2)
RETURN NUMBER AUTHID CURRENT_USER
  …
   UPDATE employee
      SET emp_loc = p_location
   WHERE emp_dept_id = p_dept_id;
   l_count := get_emp_cout;
   …
 END update_dept;
```

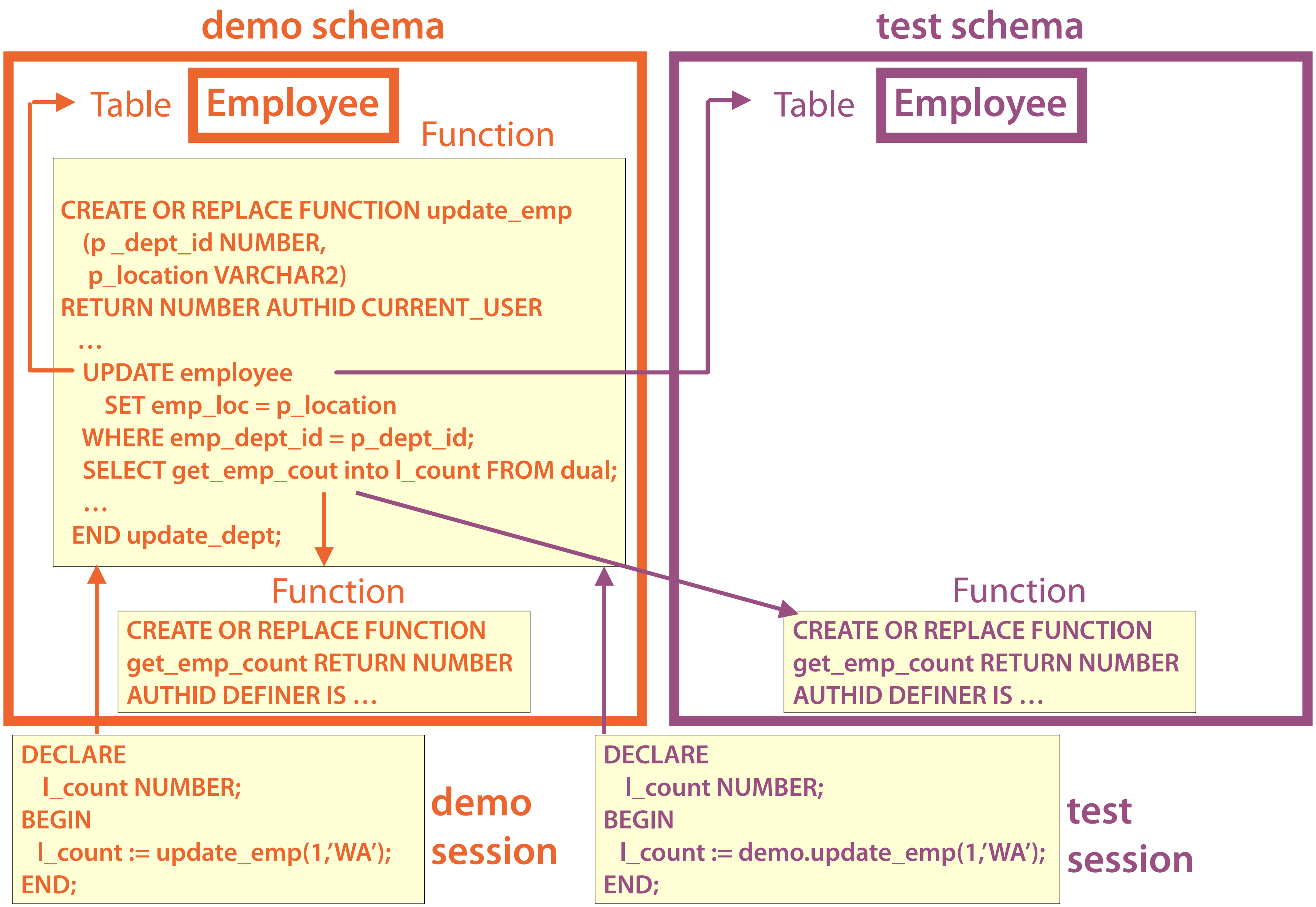
```
CREATE OR REPLACE FUNCTION
get_emp_count RETURN NUMBER
AUTHID DEFINER IS …
```

```
CREATE OR REPLACE FUNCTION
get_emp_count RETURN NUMBER
AUTHID DEFINER IS …
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := update_emp(1,'WA');
END;
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := demo.update_emp(1,'WA');
END;
```

External References For AUTHID CURRENT_USER



```
CREATE OR REPLACE FUNCTION update_emp
   (p _dept_id NUMBER,
    p_location VARCHAR2)
RETURN NUMBER AUTHID CURRENT_USER
  …
  UPDATE employee
      SET emp_loc = p_location
  WHERE emp_dept_id = p_dept_id;
  SELECT get_emp_cout into l_count FROM dual;
  …
 END update_dept;
```

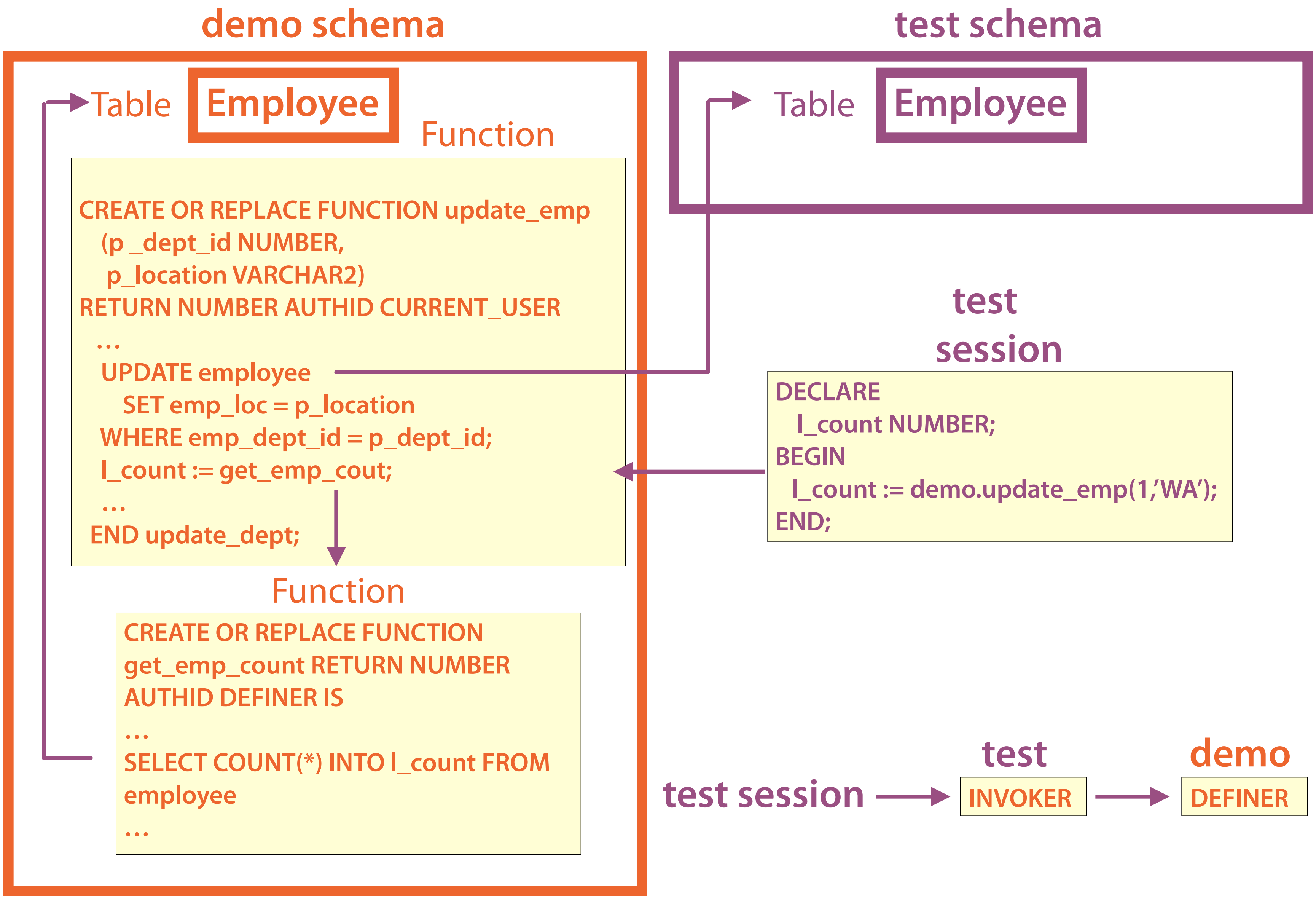
```
CREATE OR REPLACE FUNCTION
get_emp_count RETURN NUMBER
AUTHID DEFINER IS …
```

```
CREATE OR REPLACE FUNCTION
get_emp_count RETURN NUMBER
AUTHID DEFINER IS …
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := update_emp(1,'WA');
END;
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := demo.update_emp(1,'WA');
END;
```

Invoker to Definer

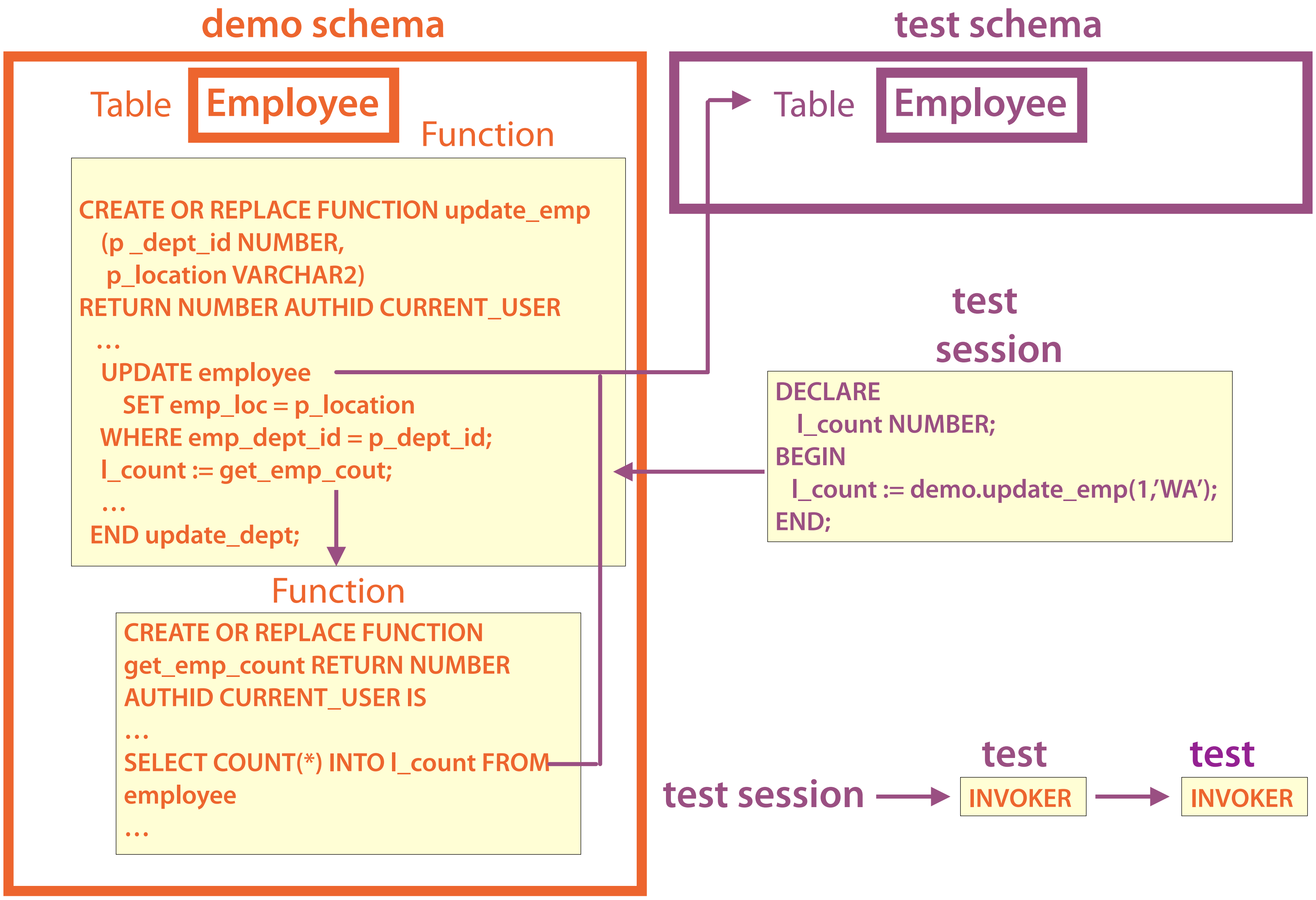


```
CREATE OR REPLACE FUNCTION update_emp
   (p _dept_id NUMBER,
    p_location VARCHAR2)
RETURN NUMBER AUTHID CURRENT_USER
  …
   UPDATE employee
      SET emp_loc = p_location
   WHERE emp_dept_id = p_dept_id;
   l_count := get_emp_cout;
   …
 END update_dept;
```

```
CREATE OR REPLACE FUNCTION
get_emp_count RETURN NUMBER
AUTHID DEFINER IS
…
SELECT COUNT(*) INTO l_count FROM
employee
…
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := demo.update_emp(1,'WA');
END;
```

Invoker to Invoker

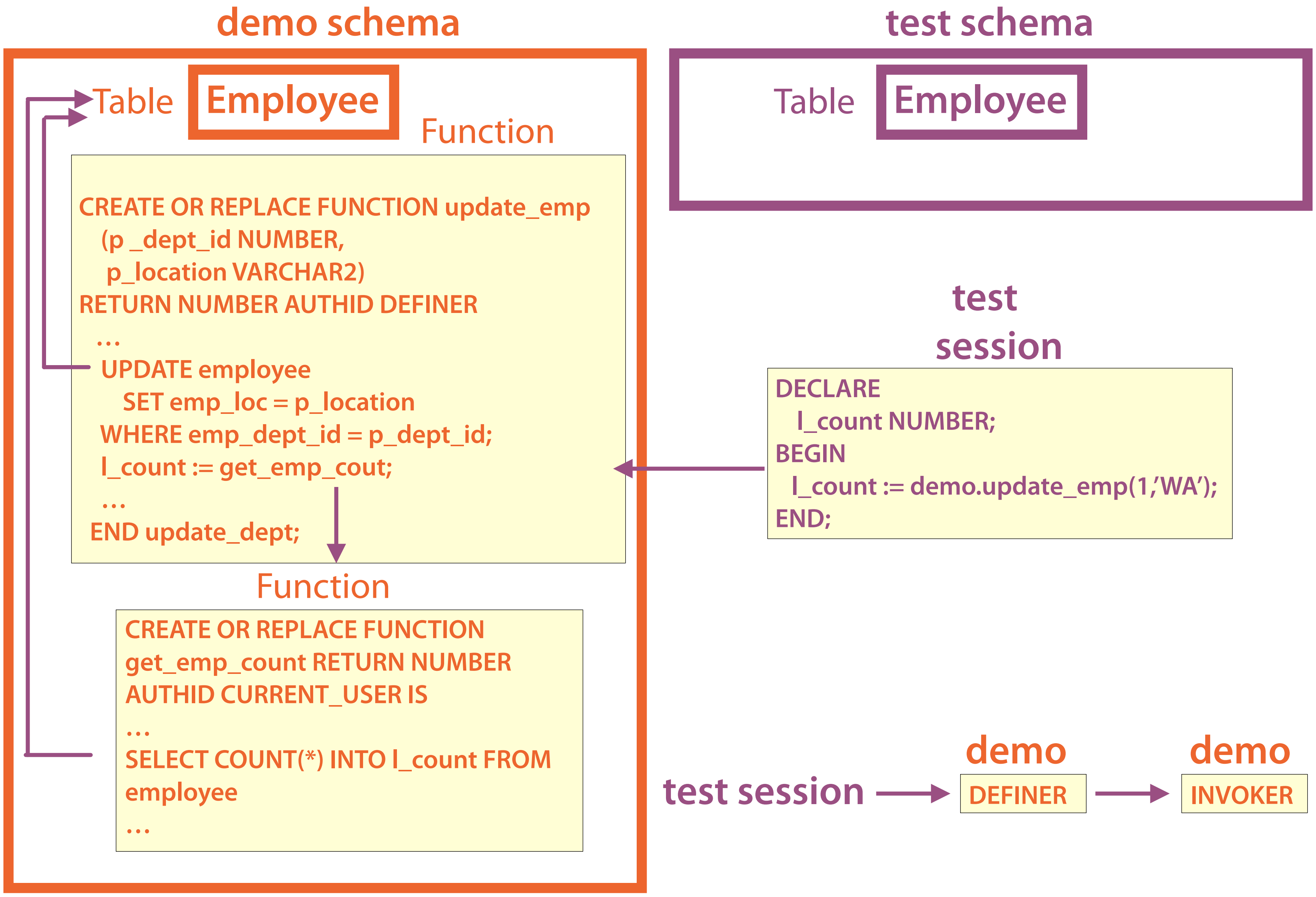


```
CREATE OR REPLACE FUNCTION update_emp
   (p _dept_id NUMBER,
    p_location VARCHAR2)
RETURN NUMBER AUTHID CURRENT_USER
  …
   UPDATE employee
      SET emp_loc = p_location
   WHERE emp_dept_id = p_dept_id;
   l_count := get_emp_cout;
   …
 END update_dept;
```

```
CREATE OR REPLACE FUNCTION
get_emp_count RETURN NUMBER
AUTHID CURRENT_USER IS
…
SELECT COUNT(*) INTO l_count FROM
employee
…
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := demo.update_emp(1,'WA');
END;
```

Definer to Invoker



```
CREATE OR REPLACE FUNCTION update_emp
   (p _dept_id NUMBER,
    p_location VARCHAR2)
RETURN NUMBER AUTHID DEFINER
  …
  UPDATE employee
     SET emp_loc = p_location
  WHERE emp_dept_id = p_dept_id;
  l_count := get_emp_cout;
   …
 END update_dept;
```

```
CREATE OR REPLACE FUNCTION
get_emp_count RETURN NUMBER
AUTHID CURRENT_USER IS
…
SELECT COUNT(*) INTO l_count FROM
employee
…
```

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := demo.update_emp(1,'WA');
END;
```

Direct Grants

Explicitly Granting Privileges to User Directly

```
GRANT SELECT, UPDATE, INSERT, DELETE on demo.employee to test;

GRANT EXECUTE ON demo.get_emp_count to test;
```

Roles

Granting Multiple Privileges to User(s)
Can Be Granted to Another Role
Based on Functions or Business Role

```
CREATE ROLE human_resources;

GRANT SELECT, UPDATE, INSERT, DELETE on demo.employee to human_resources;

GRANT EXECUTE ON demo.get_emp_count to human_resources;

GRANT human_resources to test;
```

Privileges for AUTHID DEFINER

- Roles Disabled
- Only Direct Grants Work

test session

```
CREATE OR REPLACE FUNCTION update_emp(p_dept_id NUMBER,
                                                   p_location VARCHAR2) RETURN
NUMBER AUTHID DEFINER  AS
   l_count NUMBER;
 BEGIN
   UPDATE demo.employee
      SET emp_loc = p_location
   WHERE emp_dept_id = p_dept_id;
   COMMIT;
    l_count := demo.get_emp_count(p_dept_id);
    RETURN l_count;
  EXCEPTION
     WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
       ROLLBACK;
       RAISE;
  END update_emp;
```

Privileges for AUTHID CURRENT_USER

- Roles Enabled for Runtime Evaluation
- Compilation Requires Direct Grants in Compiling Schema

Privileges for AUTHID CURRENT_USER

test session

```
CREATE OR REPLACE FUNCTION update_emp(p_dept_id NUMBER,
                                        p_location VARCHAR2) RETURN
NUMBER AUTHID CURRENT_USER  AS
   l_count NUMBER;
 BEGIN
   UPDATE demo.employee
      SET emp_loc = p_location
   WHERE emp_dept_id = p_dept_id;
   COMMIT;
    l_count := demo.get_emp_count(p_dept_id);
    RETURN l_count;
  EXCEPTION
     WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
       ROLLBACK;
       RAISE;
  END update_emp;
```

```
CREATE ROLE hr_role;
GRANT SELECT, UPDATE, INSERT, DELETE
on demo.employee to hr_role;
```

```
GRANT EXECUTE ON test.update_emp to  dev;
GRANT hr_role to dev;
```

dev session →

dev

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := test.update_emp(1,'WA');
END;
```

Selective Privileges

test session

```
CREATE OR REPLACE FUNCTION update_emp(p_dept_id NUMBER,
                                             p_location VARCHAR2) RETURN
NUMBER AUTHID CURRENT_USER  AS
  l_count NUMBER;
 BEGIN
   UPDATE demo.employee
      SET emp_loc = p_location
   WHERE emp_dept_id = p_dept_id;
   COMMIT;
    l_count := demo.get_emp_count(p_dept_id);
    RETURN l_count;
  EXCEPTION
     WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
       ROLLBACK;
       RAISE;
  END update_emp;
```

```
CREATE ROLE hr_role;
GRANT SELECT, UPDATE, INSERT, DELETE
on demo.employee to hr_role;
```

```
GRANT EXECUTE ON test.update_emp to  dev;
GRANT hr_role to dev;
```

dev

dev session ⟶

```
DECLARE
   l_count NUMBER;
BEGIN
  l_count := test.update_emp(1,'WA');
END;
```

Summary

Name Resolution

AUTHID Clause

Direct Grants vs Roles