Oracle Database 11g

PL SQL – Part II

Subprograms

What Are PL/SQL Program Units?

- Named PL/SQL blocks , Stored in the database
- Three main categories:
  - Procedures to perform actions
  - Functions to compute a value

Subprogram Components

HEADER – Mandatory

Subprogram name, type, and arguments

DECLARATIVE – Optional

Local identifiers

EXECUTABLE – Mandatory

SQL statements

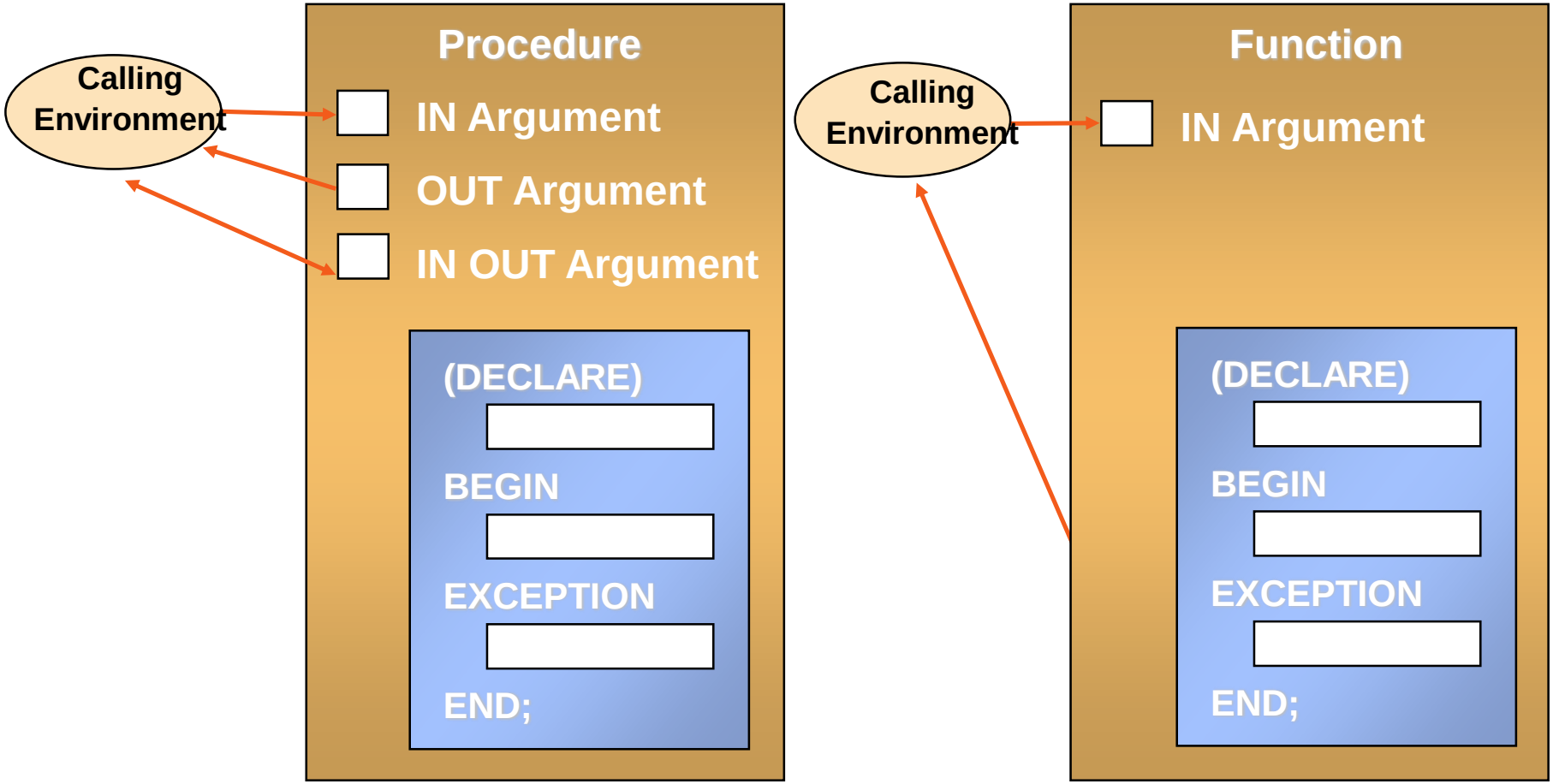
PL/SQL control statements

EXCEPTION – Optional

Actions to perform when errors occur

END; – Mandatory

Procedure or Function?



Stored Procedures

Syntax for Creating Stored Procedures

```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(parameter1 [mode] datatype1,
   parameter2 [mode] datatype2, ...)]
IS|AS
  [local_variable_declarations; …]
BEGIN
  -- actions;
END [procedure_name];
```

PL/SQL Block

- Add the OR REPLACE option to overwrite an existing procedure.

What Are Parameters?

- **Parameters:**
  - **Are declared after the subprogram name in the PL/SQL header**
  - **Pass or communicate data between the caller and the subprogram**
  - **Are used like local variables but are dependent on their parameter-passing mode:**
    - **An `IN` parameter (the default) provides values for a subprogram to process.**
    - **An `OUT` parameter returns a value to the caller.**
    - **An `IN OUT` parameter supplies an input value, which may be returned (output) as a modified value.**

Creating a Procedure: Example

```
CREATE PROCEDURE change_salary
       (v_emp_id IN NUMBER,
       v_new_salary IN NUMBER)
IS
BEGIN
       UPDATE s_emp
       SET    salary = v_new_salary
       WHERE  id = v_emp_id;
       COMMIT;
END change_salary;
```

Invoking Procedures

You can invoke procedure:

```
BEGIN
 change_salary(10, 7900);

END;
```

Removing Procedures

- **You can remove a procedure that is stored in the database.**
  - **Syntax:**

```
DROP PROCEDURE procedure_name
```

  - **Example:**

```
DROP PROCEDURE raise_salary;
```

Stored Functions

Syntax for Creating Stored Functions

- The PL/SQL block **must** have at least one **RETURN** statement.

```
CREATE [OR REPLACE] FUNCTION function_name
 [(parameter1 [mode1] datatype1, ...)]
RETURN datatype
IS|AS
 [local_variable_declarations; …]
BEGIN
  -- actions;
  RETURN expression;
END [function_name];
```

PL/SQL Block

- The RETURN data type must not include a size specification.

Stored Function: Example

- **Create the function:**

```
CREATE OR REPLACE FUNCTION get_sal
 (p1 emp.empno%TYPE)
RETURN NUMBER
IS
  v1 emp.sal%TYPE := 0;
BEGIN
         SELECT sal
         INTO   v1
         FROM   emp
         WHERE  empno = p1;
  RETURN salary;
END get_sal;
```

Ways to Execute Functions

- Using a local variable to obtain the result

```
DECLARE
 v employees.salary%type;
BEGIN
  v := get_sal(100);
dbms_output.put_line(v);
END;
```

- Use as a parameter to another subprogram

```
Begin
dbms_output.put_line(get_sal(100));
End;
```

- Use in a SQL statement (subject to restrictions)

```
SELECT employee_id, get_sal(employee_id) FROM employees;
```

- Gives the same result for

```
SELECT employee_id, salary FROM employees;
```

Function in SQL Expressions: Example

```
CREATE OR REPLACE FUNCTION tax(value IN NUMBER)
RETURN NUMBER
IS
BEGIN
   RETURN (value * 0.08);
END tax;
--
SELECT employee_id, last_name, salary, tax(salary)
FROM   employees
WHERE  department_id = 100;
```

```
Function created.
```

EMPLOYEE_ID	LAST_NAME	SALARY	TAX(SALARY)
108	Greenberg	12000	960
109	Faviet	9000	720
110	Chen	8200	656
111	Sciarra	7700	616
112	Urman	7800	624
113	Popp	6900	552

```
6 rows selected.
```

Removing Functions

- **Removing a stored function:**

```
DROP FUNCTION function_name
```

- **Example:**

```
DROP FUNCTION get_sal;
```

Procedures Versus Functions

Procedures	Functions
Execute as a PL/SQL statement	Invoke as part of an expression
Do not contain `RETURN` clause in the header	Must contain a `RETURN` datatype clause in the header
Can return values (if any) in output parameters	Must return a single value
Can contain a `RETURN` statement without a value In PL/SQL block to exit from procedure	Must contain at least one `RETURN` statement with value In PL/SQL block

Manipulating Data

Make changes to database tables by using DML commands.

- **INSERT**
- **UPDATE (already covered)**
- **DELETE**

Inserting Data: Example

Add a new order to S_ORD table for the specified customer ID

```
Create PROCEDURE cust_order
(v_customer_id  s_ord.customer_id%TYPE,
v_payment_type  s_ord.payment_type%TYPE )
IS
  v_date_ordered  s_ord.date_ordered%TYPE := SYSDATE;
BEGIN
  INSERT INTO s_ord (id, customer_id,
      date_ordered, payment_type)
    VALUES    (s_ord_id.NEXTVAL, v_customer_id,
      v_date_ordered, v_payment_type);
END cust_order;
```

Using The %ROWTYPE Attribute

```
DECLARE
  emp_rec   employees%ROWTYPE;
BEGIN
  SELECT *
  INTO emp_rec
  FROM employees
  WHERE  employee_id = 124;

  INSERT INTO retired_emps(empno, ename, job, mgr,
  hiredate, leavedate, sal, comm, deptno)
  VALUES (emp_rec.employee_id, emp_rec.last_name,
  emp_rec.job_id, emp_rec.manager_id,
  emp_rec.hire_date, SYSDATE, emp_rec.salary,
  emp_rec.commission_pct, emp_rec.department_id);
END;
```

/* retired_emps table has an extra column "Leave_date" that does not exist in employees table */

IF retired_emps table contains the same columns as employees table.

```
INSERT INTO retired_emps VALUES emp_rec;
```

Deleting Data: Example

Delete an order with specific id provided by the user.

```
Create PROCEDURE del_order
  (v_ord_id   s_ord.id%TYPE)
IS
BEGIN
  DELETE FROM s_ord
  WHERE       id = v_ord_id;
END del_order;
```