

# 10

## Creating Functions

# Objectives

**After completing this lesson, you should be able to do the following:**

- **Describe the uses of functions**
- **Create stored functions**
- **Invoke a function**
- **Remove a function**
- **Differentiate between a procedure and a function**

# Overview of Stored Functions

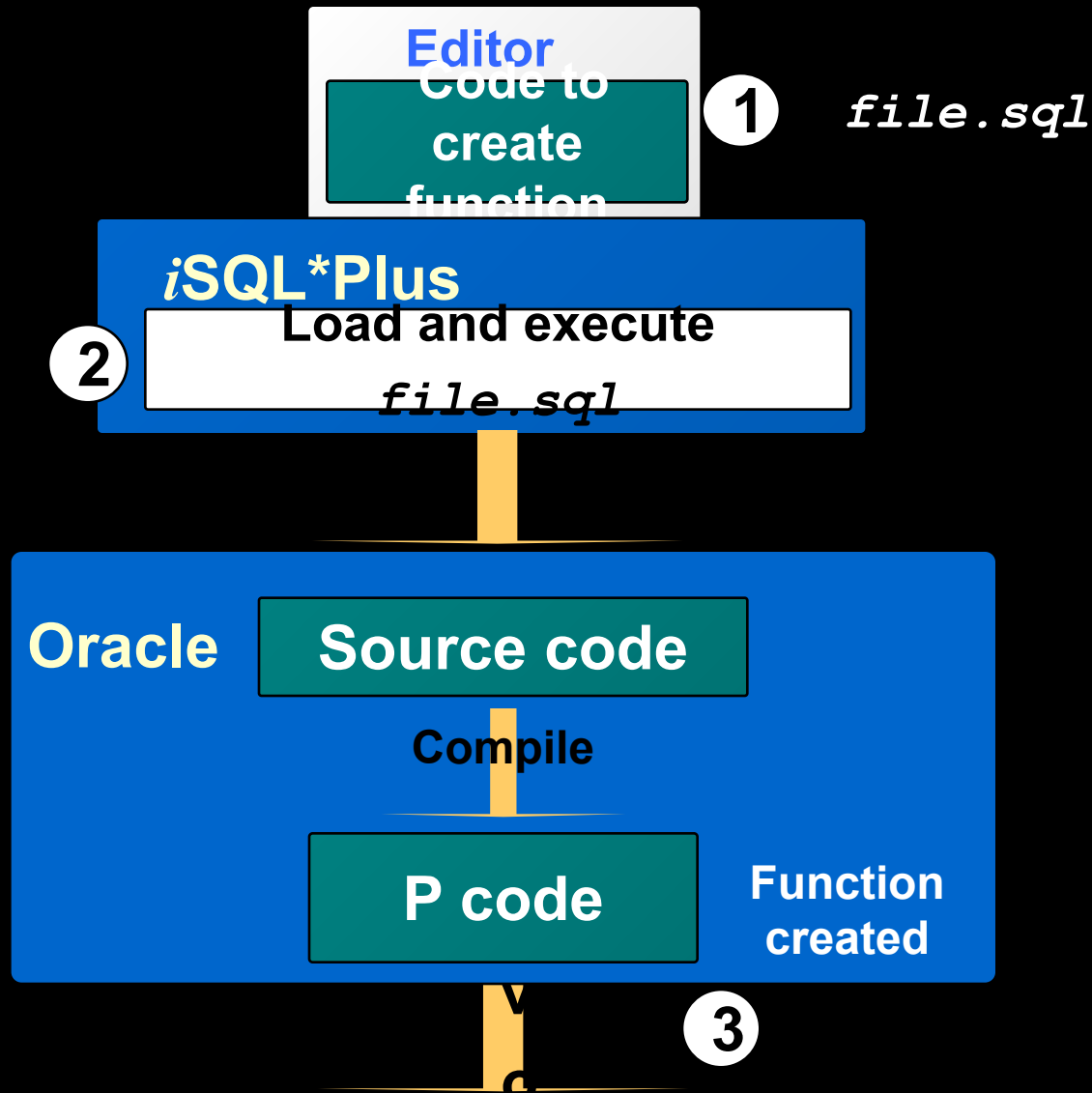
- **A function is a named PL/SQL block that returns a value.**
- **A function can be stored in the database as a schema object for repeated execution.**
- **A function is called as part of an expression.**

# Syntax for Creating Functions

```
CREATE [OR REPLACE] FUNCTION function_name
  [(parameter1 [mode1] datatype1,
    parameter2 [mode2] datatype2,
    . . .)]
RETURN datatype
IS|AS
PL/SQL Block;
```

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

# Creating a Function



# Creating a Stored Function by Using *iSQL\*Plus*

1. Enter the text of the `CREATE FUNCTION` statement in an editor and save it as a SQL script file.
2. Run the script file to store the source code and compile the function.
3. Use `SHOW ERRORS` to see compilation errors.
4. When successfully compiled, invoke the function.

# Creating a Stored Function by Using *iSQL\*Plus*: Example

get\_salary.sql

```
CREATE OR REPLACE FUNCTION get_sal
    (p_id  IN employees.employee_id%TYPE)
    RETURN NUMBER
IS
    v_salary employees.salary%TYPE :=0;
BEGIN
    SELECT salary
    INTO    v_salary
    FROM    employees
    WHERE   employee_id = p_id;
    RETURN v_salary;
END get_sal;
/
```

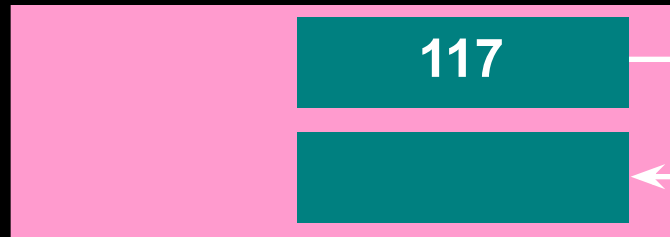
# Executing Functions

- **Invoke a function as part of a PL/SQL expression.**
- **Create a variable to hold the returned value.**
- **Execute the function. The variable will be populated by the value returned through a RETURN statement.**



# Executing Functions: Example

Calling environment



GET\_SAL function



1. Load and run the `get_salary.sql` file to create the function

2 → `VARIABLE g_salary NUMBER`

3 → `EXECUTE :g_salary := get_sal(117)`

4 → `PRINT g_salary`

PL/SQL procedure successfully completed.

G_SALARY
2800

# Advantages of User-Defined Functions in SQL Expressions

- **Extend SQL where activities are too complex, too awkward, or unavailable with SQL**
- **Can increase efficiency when used in the WHERE clause to filter data, as opposed to filtering the data in the application**
- **Can manipulate character strings**

# Invoking Functions in SQL Expressions: Example

```
CREATE OR REPLACE FUNCTION tax(p_value IN NUMBER)
  RETURN NUMBER IS
BEGIN
  RETURN (p_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.

# Locations to Call User-Defined Functions

- **Select list of a `SELECT` command**
- **Condition of the `WHERE` and `HAVING` clauses**
- **`CONNECT BY`, `START WITH`, `ORDER BY`, and `GROUP BY` clauses**
- **`VALUES` clause of the `INSERT` command**
- **`SET` clause of the `UPDATE` command**

# Restrictions on Calling Functions from SQL Expressions

To be callable from SQL expressions, a user-defined function must:

- Be a stored function
- Accept only **IN** parameters
- Accept only valid SQL data types, not PL/SQL specific types, as parameters
- Return data types that are valid SQL data types, not PL/SQL specific types

# Restrictions on Calling Functions from SQL Expressions

- Functions called from SQL expressions cannot contain DML statements.
- Functions called from UPDATE/DELETE statements on a table T cannot contain DML on the same table T.
- Functions called from an UPDATE or a DELETE statement on a table T cannot query the same table.
- Functions called from SQL statements cannot contain statements that end the transactions.
- Calls to subprograms that break the previous restriction are not allowed in the function.

# Restrictions on Calling from SQL

```
CREATE OR REPLACE FUNCTION dml_call_sql (p_sal NUMBER)
  RETURN NUMBER IS
BEGIN
  INSERT INTO employees(employee_id, last_name,
    email,
                                hire_date, job_id, salary)
  VALUES(1, 'employee 1', 'emp1@company.com',
    SYSDATE, 'SA_MAN', 1000);
  RETURN (p_sal + 100);
END;
/
```

Function created.

```
UPDATE employees SET salary = dml_call_sql(2000)
  WHERE employee_id = 170;
```

```
UPDATE employees SET salary = dml_call_sql(2000)
```

\*

ERROR at line 1:

ORA-04091: table PLSQL.EMPLOYEES is mutating, trigger/function may not see it

ORA-06512: at "PLSQL.DML\_CALL\_SQL", line 4

# Removing Functions

Drop a stored function.

**Syntax:**

```
DROP FUNCTION function_name
```

**Example:**

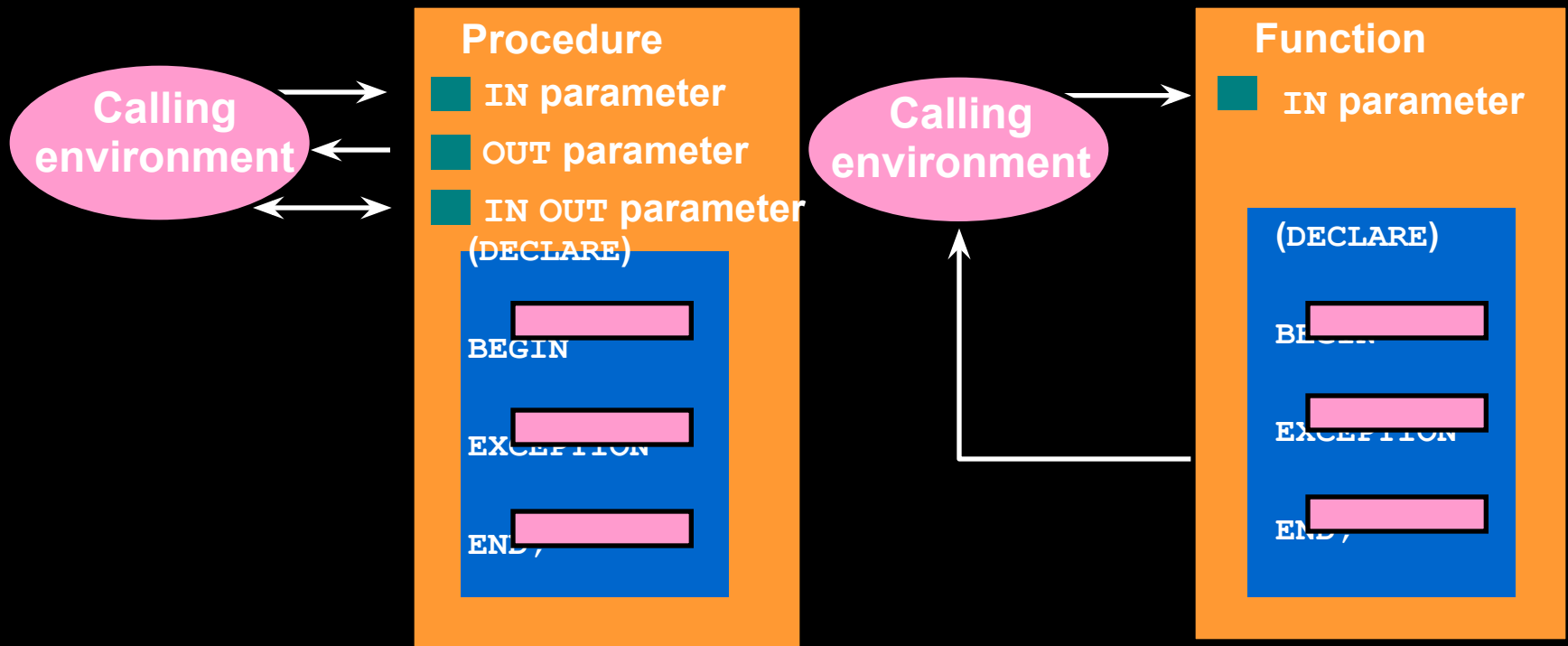
```
DROP FUNCTION get_sal;
```

```
Function dropped.
```

- All the privileges granted on a function are revoked when the function is dropped.
- The **CREATE OR REPLACE** syntax is equivalent to dropping a function and recreating it. Privileges granted on the function remain the same when this syntax is used.



# Procedure or Function?



# Comparing Procedures and Functions

<b>Procedures</b>	<b>Functions</b>
<b>Execute as a PL/SQL statement</b>	<b>Invoke as part of an expression</b>
<b>Do not contain RETURN clause in the header</b>	<b>Must contain a RETURN clause in the header</b>
<b>Can return none, one, or many values</b>	<b>Must return a single value</b>
<b>Can contain a RETURN statement</b>	<b>Must contain at least one RETURN statement</b>

# Benefits of Stored Procedures and Functions

- **Improved performance**
- **Easy maintenance**
- **Improved data security and integrity**
- **Improved code clarity**

# Summary

**In this lesson, you should have learned that:**

- **A function is a named PL/SQL block that must return a value.**
- **A function is created by using the `CREATE FUNCTION` syntax.**
- **A function is invoked as part of an expression.**
- **A function stored in the database can be called in SQL statements.**
- **A function can be removed from the database by using the `DROP FUNCTION` syntax.**
- **Generally, you use a procedure to perform an action and a function to compute a value.**

**ORACLE**

# Practice 10 Overview

**This practice covers the following topics:**

- **Creating stored functions**
  - To query a database table and return specific values
  - To be used in a SQL statement
  - To insert a new row, with specified parameter values, into a database table
  - Using default parameter values
- **Invoking a stored function from a SQL statement**
- **Invoking a stored function from a stored procedure**





