Creating Stored 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

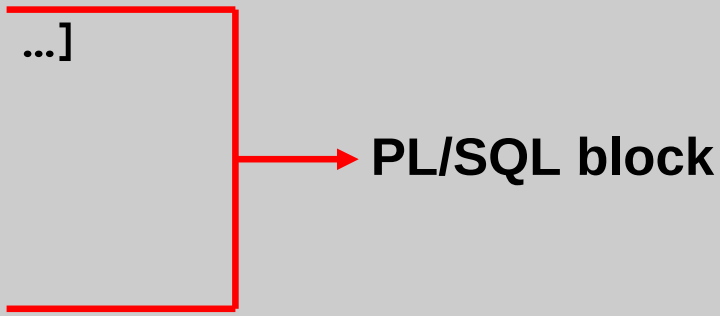
A function:

- Is a named PL/SQL block that returns a value
- Can be stored in the database as a schema object for repeated execution
- Is called as part of an expression or used to provide a parameter value

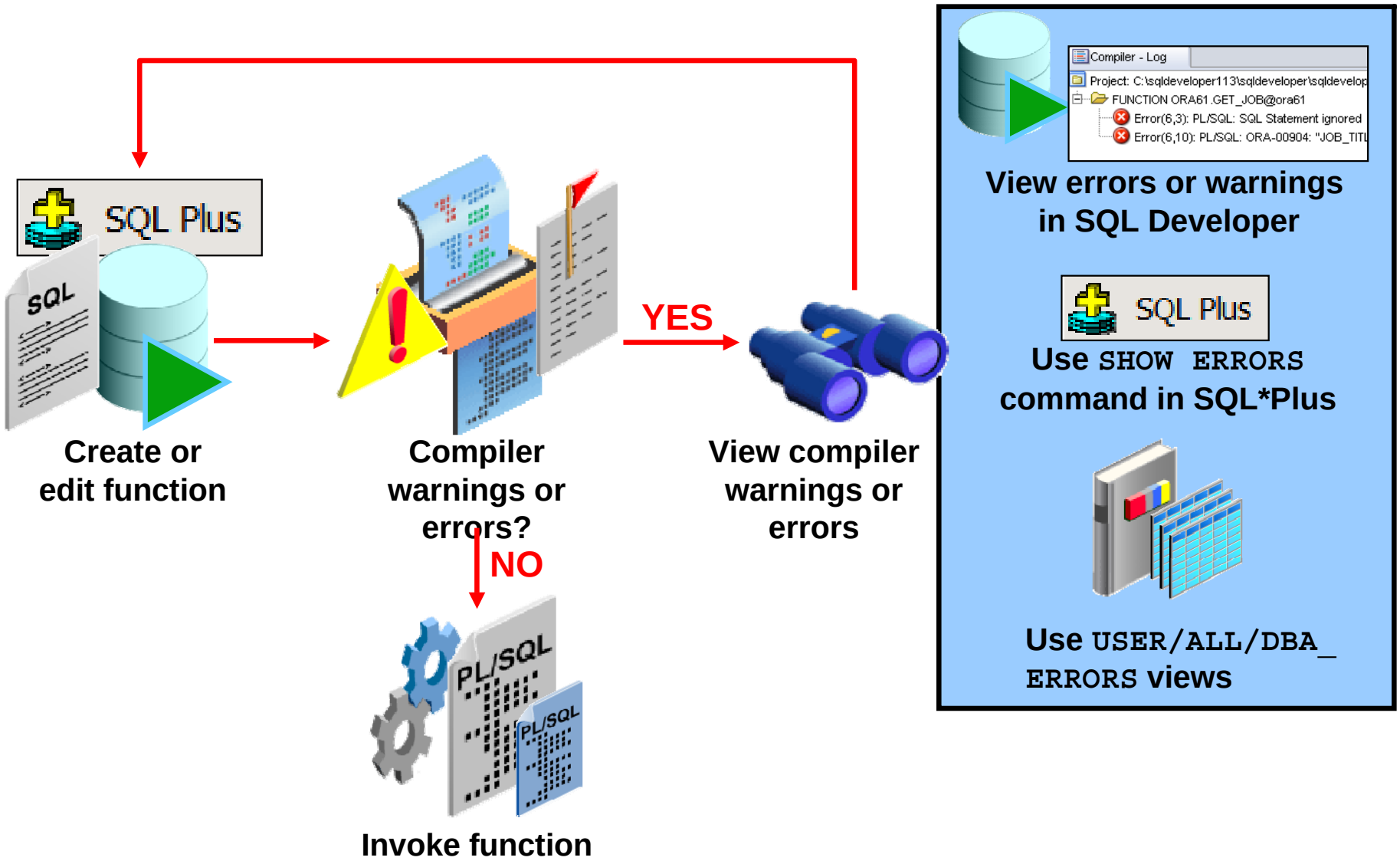
Syntax for Creating Functions

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];
```



Developing Functions



Stored Function: Example

- Create the function:

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

- Invoke the function as an expression or a parameter value:

```
EXECUTE dbms_output.put_line(get_sal(100))
```

Ways to Execute Functions

- Invoke as part of a PL/SQL expression, using a:
  - Host variable to obtain the result:

```
VARIABLE salary NUMBER
EXECUTE :salary := get_sal(100)
```

  - Local variable to obtain the result:

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

- Use as a parameter to another subprogram:

```
EXECUTE dbms_output.put_line(get_sal(100))
```

- Use in a SQL statement (subject to restrictions):

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

Advantages of User-Defined Functions in SQL Statements

- Can 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 data values

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;
```

	EMPLOYEE_ID	LAST_NAME	SALARY	TAX(SALARY)
1	108	Greenberg	12000	960
2	109	Faviet	9000	720
3	110	Chen	8200	656
4	111	Sciarra	7700	616
5	112	Urman	7800	624
6	113	Popp	6900	552

Locations to Call User-Defined Functions

User-defined functions act like built-in single-row functions and can be used in:

- The `SELECT` list or clause of a query
- Conditional expressions of the `WHERE` and `HAVING` clauses
- The `CONNECT BY`, `START WITH`, `ORDER BY`, and `GROUP BY` clauses of a query
- The `VALUES` clause of the `INSERT` statement
- The `SET` clause of the `UPDATE` statement

Restrictions on Calling Functions from SQL Expressions

- User-defined functions that are callable from SQL expressions must:
  - Be stored in the database
  - Accept only `IN` parameters with valid SQL data types, not PL/SQL-specific types
  - Return valid SQL data types, not PL/SQL-specific types
- When calling functions in SQL statements:
  - Parameters must be specified with positional notation
  - You must own the function or have the `EXECUTE` privilege

Controlling Side Effects When Calling Functions from SQL Expressions

Functions called from:

- A `SELECT` statement cannot contain DML statements
- An `UPDATE` or `DELETE` statement on a table `T` cannot query or contain DML on the same table `T`
- SQL statements cannot end transactions (that is, cannot execute `COMMIT` or `ROLLBACK` operations)

Note: Calls to subprograms that break these restrictions are also not allowed in the function.

Restrictions on Calling Functions from SQL: Example

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

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

```
Error report:
SQL Error: ORA-04091: table ORA61.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA61.DML_CALL_SQL", line 4
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
```

Removing Functions

Removing a stored function:

- You can drop a stored function by using the following syntax:

```
DROP FUNCTION function_name
```

Example:

```
DROP FUNCTION get_sal;
```

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

Viewing Functions in the Data Dictionary

Information for PL/SQL functions is stored in the following Oracle data dictionary views:

- You can view source code in the `USER_SOURCE` table for subprograms that you own, or the `ALL_SOURCE` table for functions owned by others who have granted you the `EXECUTE` privilege.

```
SELECT text
FROM   user_source
WHERE  type = 'FUNCTION'
ORDER BY line;
```

- You can view the names of functions by using `USER_OBJECTS`.

```
SELECT object_name
FROM   user_objects
WHERE  object_type = 'FUNCTION';
```

Procedures Versus Functions

Procedures	Functions
Execute as a PL/SQL statement	Invoke as part of an expression
Do not contain the `RETURN` clause in the header	Must contain a `RETURN` 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	Must contain at least one `RETURN` statement

Summary

In this lesson, you should have learned how to:

- Write a PL/SQL function to compute and return a value by using the `CREATE FUNCTION SQL` statement
- Invoke a function as part of a PL/SQL expression
- Use stored PL/SQL functions in SQL statements
- Remove a function from the database by using the `DROP FUNCTION SQL` statement

Practice 2: 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