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 is used to provide a parameter value for another subprogram
- Can be grouped into PL/SQL packages

ORACLE

Overview of Stored Functions

A function is a named PL/SQL block that can accept parameters, be invoked, and return a value. In general, you use a function to compute a value. Functions and procedures are structured alike. A function must return a value to the calling environment, whereas a procedure returns zero or more values to its calling environment. Like a procedure, a function has a header, a declarative section, an executable section, and an optional exception-handling section. A function must have a RETURN clause in the header and at least one RETURN statement in the executable section.

Functions can be stored in the database as schema objects for repeated execution. A function that is stored in the database is referred to as a stored function. Functions can also be created on client-side applications.

Functions promote reusability and maintainability. When validated, they can be used in any number of applications. If the processing requirements change, only the function needs to be updated.

A function may also be called as part of a SQL expression or as part of a PL/SQL expression. In the context of a SQL expression, a function must obey specific rules to control side effects. In a PL/SQL expression, the function identifier acts like a variable whose value depends on the parameters passed

Oracle Database: Develop PL/SQL

to it.

Functions (and procedures) can be grouped into PL/SQL packages. Packages make code even more reusable and maintainable. Packages are covered in the lessons titled "Creating Packages" and "Working with Packages."

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];
```

PL/SQL Block

Syntax for Creating Functions

A function is a PL/SQL block that returns a value. A RETURN statement must be provided to return a value with a data type that is consistent with the function declaration.

You create new functions with the CREATEFUNCTION statement, which may declare a list of parameters, must return one value, and must define the actions to be performed by the standard PL/SQL block.

You should consider the following points about the CREATEFUNCTION statement:

The REPLACE option indicates that if the function exists, it is dropped and replaced with the new version that is created by the statement.

The RETURN data type must not include a size specification. The PL/SQL block starts with a BEGIN after the declaration of any local variables and ends with an END, optionally followed by the function_name.

There must be at least one RETURNexpression statement.

You cannot reference host or bind variables in the PL/SQL block of a stored function.

Note: Although the OUT and INOUT parameter modes can be used with

Oracle Database: Develop PL/SQL

functions, it is not good programming practice to use them with functions. However, if you need to return more than one value from a function, consider returning the values in a composite data structure such as a PL/SQL record or a PL/SQL table.
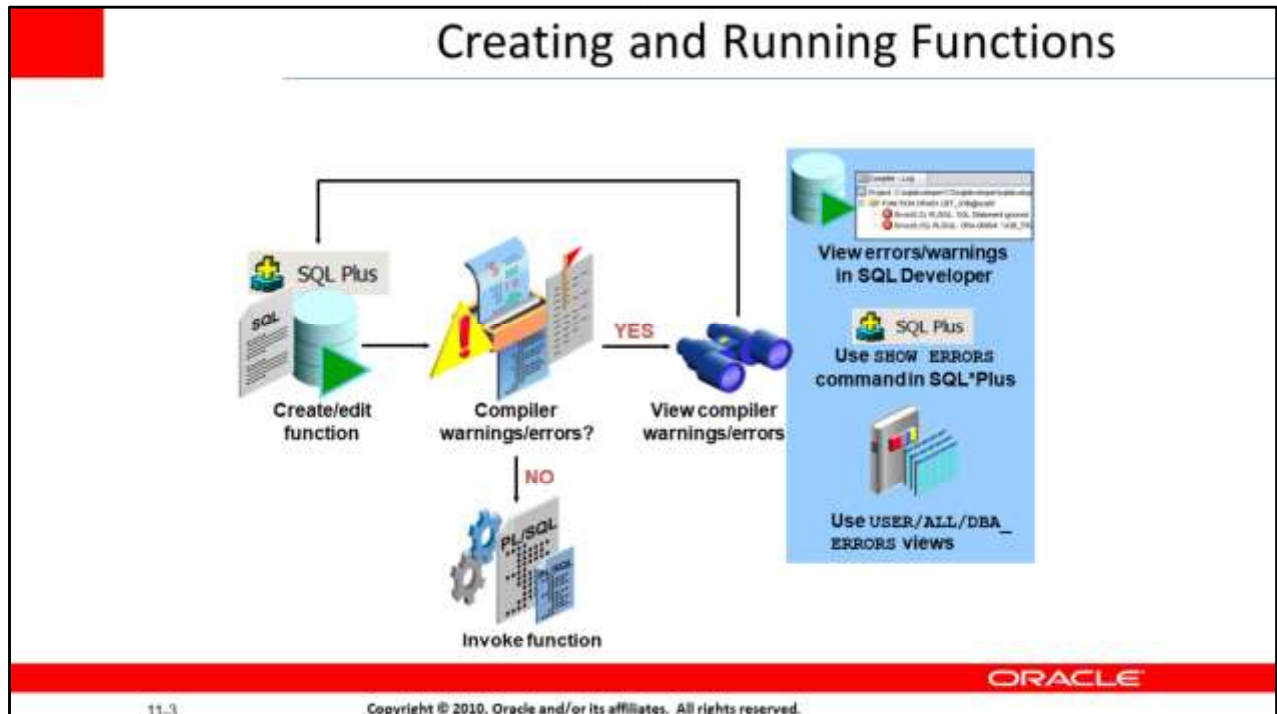
Creating and Running Functions: Overview

The diagram in the slide illustrates the basic steps involved in creating and running a function:

1. Create the function using SQL Developer's Object Navigator tree or the SQL Worksheet area.

2. Compile the function. The function is created in the database. The CREATEFUNCTION statement creates and stores source code and the compiled m-code in the database. To compile the function, right-click the function's name in the Object Navigator tree, and then click Compile.

3. If there are compilation warning or errors, you can view (and then correct) the warnings or errors using one of the following methods:

   a. Using the SQL Developer interface (the Compiler – Log tab)

   b. Using the SHOW ERRORS SQL*Plus command

   c. Using the USER/ALL/DBA_ERRORS views

4. After successful compilation, invoke the function to

Oracle Database: Develop PL/SQL

return the desired value.

Oracle Database: Develop PL/SQL

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

```
-- Invoke the function as an expression or as
-- a parameter value.

EXECUTE dbms_output.put_line(get_sal(100))
```
anonymous block completed
24000

Stored Function: Example

The get_sal function is created with a single input parameter and returns the salary as a number. Execute the command as shown, or save it in a script file and run the script to create the get_sal function.

The get_sal function follows a common programming practice of using a single RETURN statement that returns a value assigned to a local variable. If your function has an exception section, then it may also contain a RETURN statement.

Invoke a function as part of a PL/SQL expression because the function will return a value to the calling environment. The second code box uses the SQL*Plus EXECUTE command to call the DBMS_OUTPUT.PUT_LINE procedure whose argument is the return value from the function get_sal. In this case, get_sal is invoked first to calculate the salary of the employee with ID 100. The salary value returned is supplied as the value of the DBMS_OUTPUT.PUT_LINE parameter, which displays the result (if you have executed a SET SERVEROUTPUT ON).

Note: A function must always return a value. The example does not return a value if a row is not found for a given id. Ideally, create an exception handler to return a value as well.

Using Different Methods for Executing Functions

If functions are well designed, they can be powerful constructs. Functions can be invoked in the following ways:

Using Different Methods for Executing Functions

```
-- 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;
```

11-6

> As part of PL/SQL expressions: You can use host or local variables to hold the returned value from a function. The first example in the slide uses a host variable and the second example uses a local variable in an anonymous block.

Note: The benefits and restrictions that apply to functions when used in a SQL statement are discussed on the next few pages.

Using Different Methods for Executing Functions (continued)

> As a parameter to another subprogram: The first example in the slide demonstrates this usage. The get_sal function with all its arguments is nested in the parameter required by the DBMS_OUTPUT.PUT_LINE procedure. This comes from the concept of nesting functions as discussed in the course titled Oracle Database: SQL Fundamentals I.

> As an expression in a SQL statement: The second example in the slide shows how a function can be used as a single-row function in a SQL statement.

Creating and Compiling Functions Using SQL Developer

You can create a new function in SQL Developer using the following steps:

1. Right-click the Functions node.

2. Select New Function from the shortcut menu. The Create PL/SQL Function dialog box is displayed.

3. Select the schema, function name, and the parameters

Executing Functions Using SQL Developer

11-8   Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE
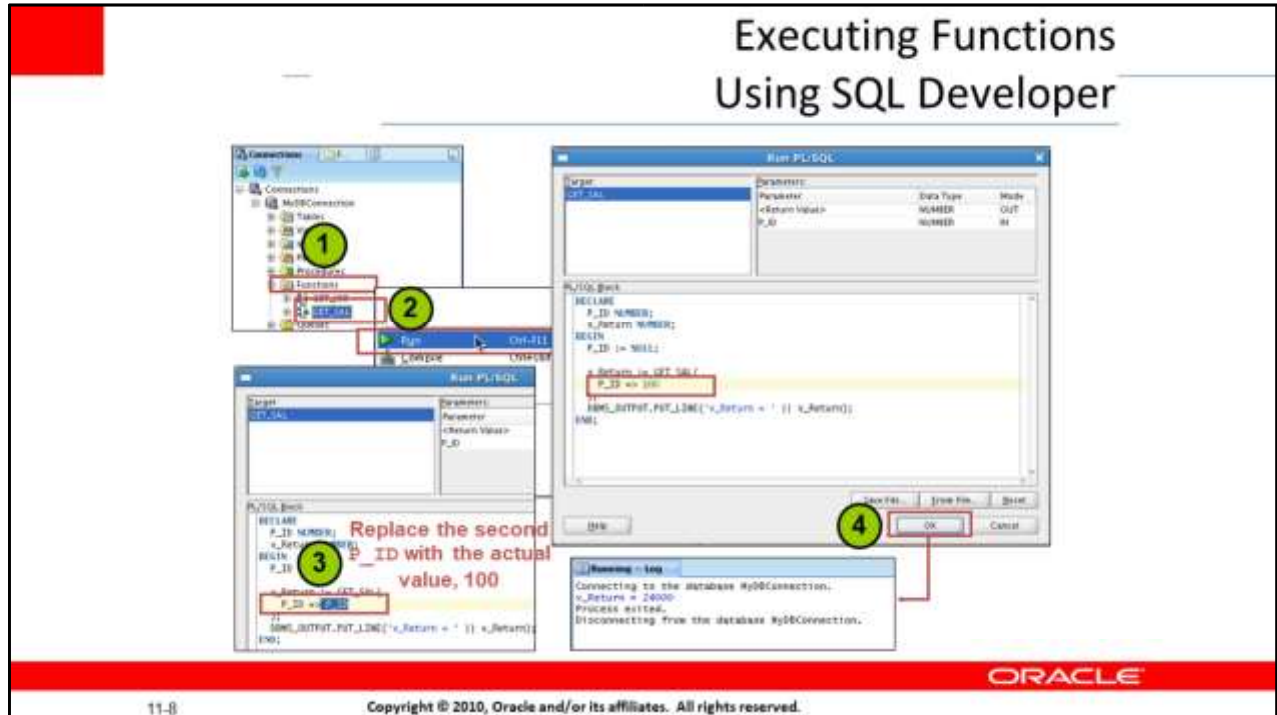
list (using the + icon), and then click OK. The code editor for the function is displayed.

4.   Enter the function's code.

5.   To compile the function, click the Compile icon.

Note

To create a new function in SQL Developer, you can also enter the code in the SQL Worksheet, and then click the Run Script icon.

For additional information about creating functions in SQL Developer, access the appropriate online help topic titled "Create PL/SQL Subprogram Function or Procedure."

Executing Functions Using SQL Developer

You can execute a function in SQL Developer using the following steps:

1.   Click the Functions node.

2.   Right-click the function's name, and then select Run. The Run PL/SQL dialog box is displayed.

3.   Replace the second parameter name with the actual parameter value as shown in the slide example.

4.   Click OK.

Note: For additional information about running functions in SQL Developer, access the online help topic titled "Running and Debugging Functions and Procedures."

Using a Function in a SQL Expression

```
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;
```

11-10

Advantages of User-Defined Functions in SQL Statements

SQL statements can reference PL/SQL user-defined functions anywhere a SQL expression is allowed. For example, a user-defined function can be used anywhere that a built-in SQL function, such as UPPER(), can be placed.

Advantages

Permits calculations that are too complex, awkward, or unavailable with SQL. Functions increase data independence by processing complex data analysis within the Oracle server, rather than by retrieving the data into an application

Increases efficiency of queries by performing functions in the query rather than in the application

Manipulates new types of data (for example, latitude and longitude) by encoding character strings and using functions to operate on the strings

Function in SQL Expressions: Example

The example in the slide shows how to create a tax function to calculate income tax. The function accepts a NUMBER parameter and returns the calculated income tax based on a simple flat tax rate of 8%.

To execute the code shown in the slide example in SQL Developer, enter the code in the SQL Worksheet, and then click the Run Script icon. The tax function is invoked as an expression in the SELECT clause along with the employee ID, last name, and salary for employees in a department with ID 100.

Calling User-Defined Functions in SQL Statements

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

ORACLE

The return result from the tax function is displayed with the regular output from the query.

Calling User-Defined Functions in SQL Statements

A PL/SQL user-defined function can be called from any SQL expression where a built-in single-row function can be called as shown in the following example:

```
SELECT employee_id, tax(salary)
 FROM   employees
  WHERE  tax(salary) > (SELECT MAX(tax(salary))
                 FROM employees
                 WHERE department_id = 30)
  ORDER BY tax(salary) DESC;
```

Restrictions When 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:
  - You must own the function or have the EXECUTE privilege
  - You may need to enable the PARALLEL_ENABLE keyword to allow a parallel execution of the SQL statement

```
EMPLOYEE_ID          TAX(SALARY)
-------------------- --------------------
100                  1920
101                  1360
102                  1360
145                  1120
146                  1080
201                  1040
205                  960
147                  960
108                  960
168                  920

10 rows selected
```

Restrictions When Calling Functions from SQL Expressions

The user-defined PL/SQL functions that are callable from SQL expressions must meet the following requirements:

The function must be stored in the database.

The function parameters must be IN and of valid SQL data types.

The functions must return data types that are valid SQL data types. They cannot be PL/SQL-specific data types such as BOOLEAN, RECORD, or TABLE. The same restriction applies to the parameters of the function.

The following restrictions apply when calling a function in a SQL statement:

Parameters must use positional notation. Named notation is not supported.

You must own or have the EXECUTE privilege on the function. You may need to enable the PARALLEL_ENABLE keyword to allow a parallel execution of the SQL statement using the function. Each parallel slave will have private copies of the function's local variables. Other restrictions on a user-defined function include the following: It cannot be called from the CHECK constraint clause of a CREATETABLE or ALTER TABLE statement. In addition, it cannot be used to specify a default value for

a column. Only stored functions are callable from SQL statements. Stored procedures cannot be called unless invoked from a function that meets the preceding requirements.

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.

ORACLE

Controlling Side Effects When Calling Functions from SQL Expressions To execute a SQL statement that calls a stored function, the Oracle server must know whether the function is free of specific side effects. The side effects are unacceptable changes to database tables.

Additional restrictions apply when a function is called in expressions of SQL statements:

When a function is called from a SELECT statement or a parallel UPDATE or DELETE statement, the function cannot modify database tables.

When a function is called from an UPDATE or DELETE statement, the function cannot query or modify database tables modified by that statement.

When a function is called from a SELECT, INSERT, UPDATE, or DELETE statement, the function cannot execute directly or indirectly through another subprogram or SQL transaction control statements such as:

A COMMIT or ROLLBACK statement

A session control statement (such as SETROLE)

A system control statement (such as ALTERSYSTEM)

Oracle Database: Develop PL/SQL

Program Units   2 - 13

Any DDL statements (such as CREATE) because they are followed by an automatic commit

Restrictions on Calling Functions 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, 'Frost', 'jfrost@company.com',
         SYSDATE, 'SA_MAN', p_sal);
  RETURN (p_sal + 100);
END;
```

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

Restrictions on Calling Functions from SQL: Example

The dml_call_sql function in the slide contains an INSERT statement that inserts a new record into the EMPLOYEES table and returns the input salary value incremented by 100. This function is invoked in the UPDATE statement that modifies the salary of employee 170 to the amount returned from the function. The UPDATE statement fails with an error indicating that the table is mutating (that is, changes are already in progress in the same table). In the following example, the query_call_sql function queries the SALARY column of the EMPLOYEES table:

```
CREATE OR REPLACE FUNCTION query_call_sql(p_a NUMBER)
RETURN NUMBER IS v_s NUMBER;
BEGIN
  SELECT salary INTO v_s FROM employees
      WHERE employee_id = 170;
      RETURN (v_s + p_a);
END;
```

When invoked from the following UPDATE statement, it returns the error

message similar to the error message shown in the slide:

```
UPDATE employees SET salary = query_call_sql(100)
WHERE employee_id = 170;
```

Named and Mixed Notation from SQL

– PL/SQL allows arguments in a subroutine call to be specified using positional, named, or mixed notation.

– Prior to Oracle Database 11g, only the positional notation is supported in calls from SQL.

– Starting in Oracle Database 11g, named and mixed notation can be used for specifying arguments in calls to PL/SQL subroutines from SQL statements.

– For long parameter lists, with most having default values, you can omit values from the optional parameters.

ORACLE

11-15

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Oracle Database: Develop PL/SQL Program Units

2 - 15

Sensitivity: Internal & Restricted

Removing Functions: Using the DROP SQL Statement or SQL Developer

- Using the DROP statement:

```
DROP FUNCTION f;
```

- Using SQL Developer:

11-16
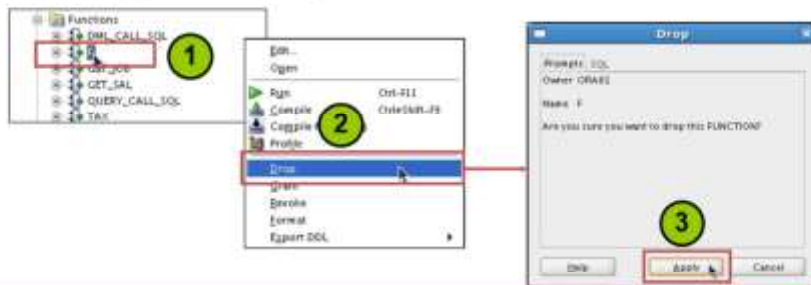
Removing Functions

Using the DROP statement

When a stored function is no longer required, you can use a SQL statement in SQL*Plus to drop it. To remove a stored function by using SQL*Plus, execute the DROP FUNCTION SQL command.

Using CREATE OR REPLACE Versus DROP and CREATE

The REPLACE clause in the CREATE OR REPLACE syntax is equivalent to dropping a function and re-creating it. When you use the CREATE OR REPLACE syntax, the privileges granted on this object to other users remain the same. When you DROP a function and then re-create it, all the privileges granted on this function are automatically revoked.

Using SQL Developer

To drop a function in SQL Developer, right-click the function name in the Functions node, and then select Drop. The Drop dialog box is displayed. To drop the function, click Apply.

Viewing Functions Using Data Dictionary Views

The source code for PL/SQL functions is stored in the data dictionary tables. The source code is accessible for PL/SQL functions that are successfully or unsuccessfully compiled. To view the PL/SQL function code stored in the data dictionary, execute a SELECT statement on the following tables where the TYPE column value is FUNCTION:
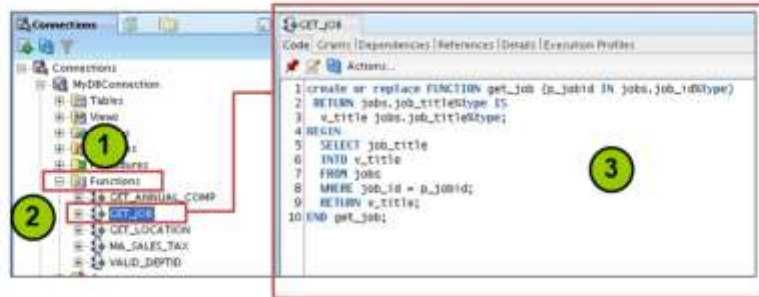
The USER_SOURCE table to display the PL/SQL code that you own
The ALL_SOURCE table to display the PL/SQL code to which you have been granted the EXECUTE right by the owner of that subprogram code

The second example in the slide uses the USER_SOURCE table to display the source code for all the functions in your schema.

You can also use the USER_OBJECTS data dictionary view to display a list of your function names.

Note: The output of the second code example in the slide was generated using the Execute Statement (F9) icon on the toolbar to provide betterformatted output.

Viewing Functions Information Using SQL Developer

Using SQL Developer

To view a function's code in SQL Developer, use the following steps:

1. Click the Functions node in the Connections tab.
2. Click the function's name.
3. The function's code is displayed in the Code tab as shown in the slide.