



# Database Programming with PL/SQL

9-2

Using Functions in SQL Statements



# Objectives

This lesson covers the following objectives:

- List the advantages of user-defined functions in SQL statements
- List where user-defined functions can be called from within a SQL statement
- Describe the restrictions on calling functions from SQL statements

# Purpose

- In this lesson, you learn how to use functions within SQL statements.
- If the SQL statement processes many rows in a table, the function executes once for each row processed by the SQL statement.
- For example, you could calculate the tax to be paid by every employee using just one function.

# What Is a User-Defined Function?

- A user-defined function is a function that is created by the PL/SQL programmer. `GET_DEPT_NAME` and `CALCULATE_TAX` are examples of user-defined functions, whereas `UPPER`, `LOWER`, and `LPAD` are examples of system-defined functions automatically provided by Oracle.
- Most system functions, such as `UPPER`, `LOWER`, and `LPAD` are stored in a package named `SYS`.
- `STANDARD`. Packages are covered in a later section.
- These system functions are often called built-in functions.

# Advantages of Functions in SQL Statements

- If used in the `WHERE` clause of a `SELECT` statement, functions can increase efficiency by insuring all of the desired rows are returned.
- For example, in a large database of employees, you could have more than one employee with the same last name.
- If you use the following code, you find an employee with the last name of "Taylor," but not the employee whose last name was entered as "taylor."

```
SELECT * FROM employees  
WHERE last_name = 'Taylor';
```

# Advantages of Functions in SQL Statements

- How many different ways would you have to search to find all possible examples of "Taylor?"
- By adding the `UPPER` function to the `WHERE` clause, you can find all examples with one search.

```
SELECT * FROM employees  
WHERE UPPER(last_name) = UPPER('TAylor');
```

- This code will return employees whose last name was stored as "Taylor," "TAYLOR," "taylor," or "TAylor."
- Likewise, it doesn't matter how the user enters the search criteria, as in this case, it was entered in all lower case ("TAylor").

# Advantages of Functions in SQL Statements

- Functions in SQL statements can also manipulate data values.
- For example, for an end-of-year social event, you want (just for fun) to print name-tags for every employee with the characters reversed, so “Mary Jones” becomes “senoJ yraM.”
- You can create a user-defined function called `reverse_name`, which does this, then code:

```
SELECT reverse_name(last_name, first_name) FROM employees;
```

# Advantages of Functions in SQL Statements

- User-defined functions can extend SQL where activities are too complex, too awkward, or unavailable with regular SQL.
- Functions can also help us overcome repeatedly writing the same code.
- For example, you want to calculate how long an employee has been working for your business, rounded to a whole number of months.



# Advantages of Functions in SQL Statements

- You could create a user-defined function called `how_many_months` to do this.
- Then, the application programmer can code:

```
SELECT employee_id, how_many_months(hire_date)
FROM employees;
```



# Function in SQL Expressions: Example

Create a function to determine each employee's taxes.

```
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 = 50;
```

EMPLOYEE_ID	LAST_NAME	SALARY	TAX(SALARY)
124	Mourgos	5800	464
141	Rajs	3500	280
142	Davies	3100	248
143	Matos	2600	208
144	Vargas	2500	200

# Where Can You Use User-Defined Functions in a SQL Statement?

- User-defined functions act like built-in single-row functions, such as `UPPER` , `LOWER` , and `LPAD`.
- They can be used in:
  - The `SELECT` column-list of a query
  - Conditional expressions in the `WHERE` and `HAVING` clauses
  - The `ORDER BY` and `GROUP BY` clauses of a query
  - The `VALUES` clause of the `INSERT` statement
  - The `SET` clause of the `UPDATE` statement
  - In short, they can be used anywhere that you have a value or expression.

# Where Can You Use User-Defined Functions in a SQL Statement?

- This example shows the user-defined function `tax` being used in four places within a single SQL statement.

```
SELECT employee_id, tax(salary)
FROM employees
WHERE tax(salary) > (SELECT MAX(tax(salary))
                     FROM employees
                     WHERE department_id = 20)
ORDER BY tax(salary) DESC;
```

- The function makes this code easier to read and much easier to update if the tax rate changes.

# Restrictions on Using Functions in SQL Statements

- To use a user-defined function within a SQL statement, the function must conform to the rules and restrictions of the SQL language.
- The function can accept only valid SQL datatypes as `IN` parameters, and must `RETURN` a valid SQL datatype.
- PL/SQL-specific types, such as `BOOLEAN` and `%ROWTYPE` are not allowed.
- SQL size limits must not be exceeded (PL/SQL allows a `VARCHAR2` variable to be up to 32 KB in size, but prior to Oracle 12c, SQL allowed only 4 KB).

# Restrictions on Using Functions in SQL Statements

- User-defined functions may use positional, named, and mixed notation for identifying arguments.
- Parameters for system functions must be specified with positional notation.
- Example:

```
SELECT employee_id, tax(p_value => salary)
FROM employees;
```

```
SELECT employee_id, UPPER(last_name)
FROM employees;
```

```
SELECT employee_id, UPPER(arg1 => last_name)
FROM employees;
```

- The third `SELECT` statement causes an error.

# Restrictions on Using Functions in SQL Statements

- Functions called from a `SELECT` statement cannot contain DML statements.
- Functions called from an `UPDATE` or `DELETE` statement on a table cannot query or contain DML on the same table.
- Functions called from any SQL statement cannot end transactions (that is, cannot execute `COMMIT` or `ROLLBACK` operations).



# Restrictions on Using Functions in SQL Statements

- Functions called from any SQL statement cannot issue DDL (for example, `CREATE TABLE`) or DCL (for example, `ALTER SESSION`) because they also do an implicit `COMMIT`.
- Calls to subprograms that break these restrictions are also not allowed in a function.



# Restrictions on Using Functions in SQL Statements: Example 1

```
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 dml_call_sql;
```

```
UPDATE employees
  SET salary = dml_call_sql(2000)
WHERE employee_id = 174;
```

```
ORA-04091: table US_1217_S90_PLSQL.EMPLOYEES is mutating, trigger/function may not see it
```

# Restrictions on Using Functions in SQL Statements: Example 2

- The following function queries the EMPLOYEES table.

```
CREATE OR REPLACE FUNCTION query_max_sal (p_dept_id NUMBER)
  RETURN NUMBER IS
  v_num NUMBER;
BEGIN
  SELECT MAX(salary) INTO v_num FROM employees
    WHERE department_id = p_dept_id;
  RETURN (v_num);
END;
```

- When used within the following DML statement, it returns the “mutating table” error message similar to the error message shown in the previous slide.

```
UPDATE employees SET salary = query_max_sal(department_id)
  WHERE employee_id = 174;
```

# Terminology

Key terms used in this lesson included:

- User-defined function

# Summary

In this lesson, you should have learned how to:

- List the advantages of user-defined functions in SQL statements
- List where user-defined functions can be called from within a SQL statement
- Describe the restrictions on calling functions from SQL statements

