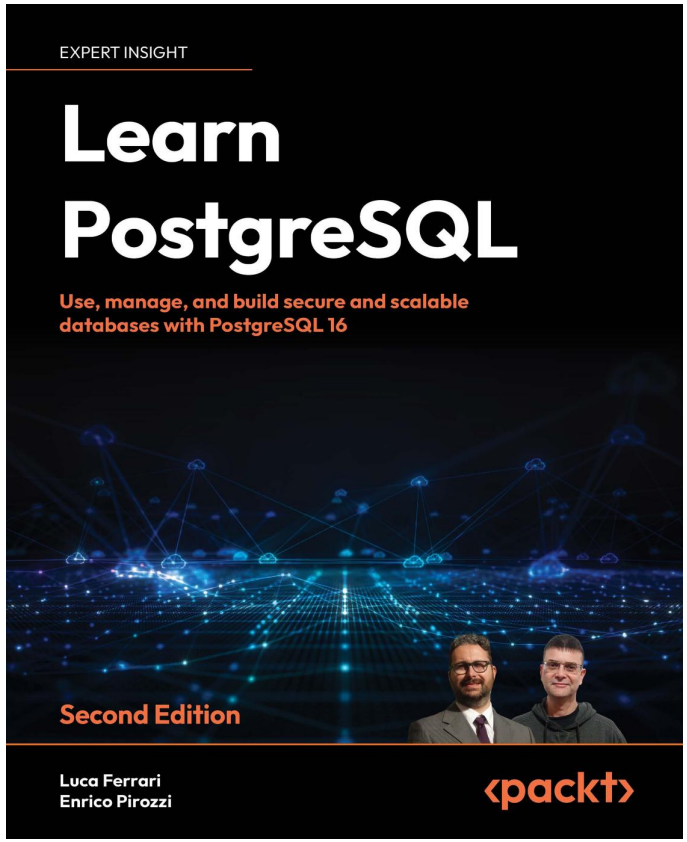
# Database Administration

## Lecture 09: SQL Functions.

Ferrari & Pirozzi

31 de marzo de 2025

# Database Administration: SQL Functions.




Content has been extracted from *Learn PostgreSQL: Use, manage, and build secure and scalable databases with PostgreSQL 16 (Chapter 7)*, by Luca Ferrari & Enrico Pirozzi, 2023. Visit https://www.packtpub.com/en-co/product/learn-postgresql-9781837635641.

# Overview

- PostgreSQL supports server-side programming via functions.
- Built-in languages: SQL, PL/pgSQL, C.
- Optional: PL/Python, PL/Perl, PL/Java, etc.
- This chapter focuses on SQL and PL/pgSQL functions.

# The Function Syntax

```
1  CREATE FUNCTION function_name(p1 type, p2 type, p3 type, ..., pn type)
2  RETURNS type AS
3  BEGIN
4  -- function logic
5  END;
6  LANGUAGE language_name
```

# The Function Steps

The following steps always apply to any type of function we want to create:

# The Function Steps

The following steps always apply to any type of function we want to create:

1. Specify the name of the function after the `CREATE FUNCTION` keywords.

# The Function Steps

The following steps always apply to any type of function we want to create:

1. Specify the name of the function after the `CREATE FUNCTION` keywords.
2. Make a list of parameters separated by commas.

# The Function Steps

The following steps always apply to any type of function we want to create:

1. Specify the name of the function after the `CREATE FUNCTION` keywords.
2. Make a list of parameters separated by commas.
3. Specify the return data type after the `RETURNS` keyword.

# The Function Steps

The following steps always apply to any type of function we want to create:

1. Specify the name of the function after the `CREATE FUNCTION` keywords.
2. Make a list of parameters separated by commas.
3. Specify the return data type after the `RETURNS` keyword.
4. For the PL/pgSQL language, put some code between the `BEGIN` and `END` blocks.

# The Function Steps

The following steps always apply to any type of function we want to create:

1. Specify the name of the function after the `CREATE FUNCTION` keywords.
2. Make a list of parameters separated by commas.
3. Specify the return data type after the `RETURNS` keyword.
4. For the PL/pgSQL language, put some code between the `BEGIN` and `END` blocks.
5. For the PL/pgSQL language, the function has to end with the `END` keyword followed by a semicolon.

# The Function Steps

The following steps always apply to any type of function we want to create:

1. Specify the name of the function after the `CREATE FUNCTION` keywords.
2. Make a list of parameters separated by commas.
3. Specify the return data type after the `RETURNS` keyword.
4. For the PL/pgSQL language, put some code between the `BEGIN` and `END` blocks.
5. For the PL/pgSQL language, the function has to end with the `END` keyword followed by a semicolon.
6. Define the language in which the function was written – for example, sql or plpgsql, plperl, plpython, and so on.

# Basic SQL Function Example

```
1  CREATE OR REPLACE FUNCTION my_sum(x integer, y integer)
2  RETURNS integer AS
3  $$
4    SELECT x + y;
5  $$ LANGUAGE SQL;
```

**Call:** `SELECT my_sum(1, 2);`

# Function Returning a Set

Returns a set of primary keys of deleted records.

```
1  CREATE OR REPLACE FUNCTION delete_posts(p_title text)
2  RETURNS SETOF integer AS
3  $$
4    DELETE FROM posts WHERE title = p_title
5    RETURNING pk;
6  $$ LANGUAGE SQL;
```

# Function Returning a Table

```
1  CREATE OR REPLACE FUNCTION delete_posts_table(p_title text)
2  RETURNS TABLE (ret_key integer, ret_title text) AS
3  $$
4    DELETE FROM posts WHERE title = p_title
5    RETURNING pk, title;
6  $$ LANGUAGE SQL;
```

# Polymorphic SQL Function

- Polymorphic functions are useful for DBAs when we need to write a function that has to work with different types of data.
- We want to create a function that accepts two parameters and replaces the first parameter with the second one if the first parameter is NULL (Oracle NVL or PostgreSQL Coalesce).
- The problem is that we want to write a single function that is valid for all types of data (integer, real, text, and so on).

# Polymorphic SQL Function

```
1  CREATE OR REPLACE FUNCTION nvl(anyelement, anyelement)
2  RETURNS anyelement AS
3  $$
4    SELECT COALESCE($1, $2);
5  $$ LANGUAGE SQL;
```

Works with multiple data types.

# PL/pgSQL Function Structure

- The PL/pgSQL language is the default built-in procedural language for PostgreSQL.
- It can do the following:
  - Can be used to create functions and trigger procedures.
  - Add new control structures.
  - Add new data types to the SQL language.
- It supports the following:
  - Variable declarations.
  - Expressions.
  - Control structures as conditional structures or loop structures.
  - Cursors.

# PL/pgSQL Function Structure

```
1  CREATE FUNCTION my_sum(x integer, y integer)
2  RETURNS integer AS
3  $$
4    DECLARE
5      ret integer;
6    BEGIN
7      ret := x + y;
8      RETURN ret;
9    END;
10 $$ LANGUAGE 'plpgsql';
```

# Using IN/OUT Parameters

```
1  CREATE FUNCTION my_sum_3_params(IN x integer, IN y integer, OUT
   ↪  z integer) AS
2  $$
3    BEGIN
4      z := x + y;
5    END;
6  $$ LANGUAGE plpgsql;
```

# Using IN/OUT Parameters

```
1  CREATE OR REPLACE FUNCTION my_sum_mul(IN x integer, IN y
   ↪  integer, OUT w integer, OUT z integer) AS
2  $$
3    BEGIN
4      z := x + y;
5      w := x * y;
6    END;
7  $$
8  language 'plpgsql';
```

# Function Volatility Categories

- **VOLATILE** – default; can modify the database; result can change.
- **STABLE** – cannot modify the databae; same result for same input in a transaction.
- **IMMUTABLE** – cannot modify the databae; result is constant forever for same input.

# Conditional Logic - IF Statement

```
1  IF x > y THEN
2    RETURN 'x > y';
3  ELSIF x < y THEN
4    RETURN 'x < y';
5  ELSE
6    RETURN 'x = y';
7  END IF;
```

# Conditional Logic - IF Statement

```
1  CREATE OR REPLACE FUNCTION my_check(x integer default 0, y
   ↪  integer default 0) RETURNS text AS
2  $BODY$
3    BEGIN
4      IF x > y THEN
5      return 'first parameter is greater than second parameter';
6      ELSIF x < y THEN
7      return 'second parameter is greater than first parameter';
8      ELSE
9      return 'the 2 parameters are equals';
10     END IF;
11   END;
12 $BODY$
13 language 'plpgsql';
```

# Loop Example with Composite Return

```
1  CREATE TYPE my_ret_type AS (
2    id integer, title text, record_data hstore
3  );
4
5  CREATE FUNCTION my_first_fun(p_id integer)
6  RETURNS SETOF my_ret_type AS
7  $$
8    DECLARE
9      rw posts%ROWTYPE;
10     ret my_ret_type;
11   BEGIN
12     FOR rw IN SELECT * FROM posts WHERE pk = p_id LOOP
13       ret.id := rw.pk;
14       ret.title := rw.title;
15       ret.record_data := hstore(
16         ARRAY['title', rw.title,
17               'Title and Content', format('%s %s', rw.title, rw.content)]
18       );
19       RETURN NEXT ret;
20     END LOOP;
21     RETURN;
22   END;
23 $$
24 LANGUAGE 'plpgsql';
```

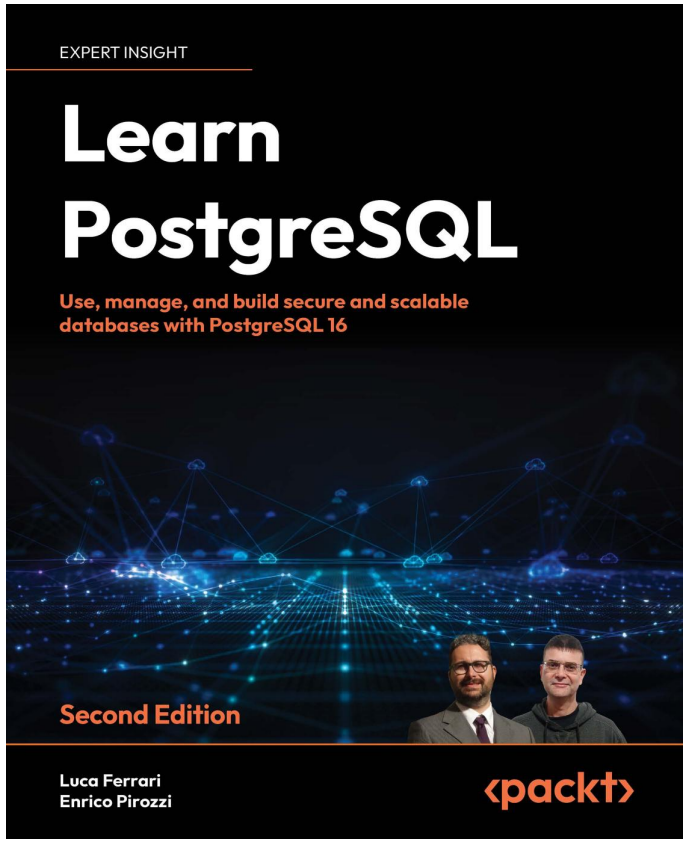
# Exception Handling

```
1  CREATE FUNCTION my_second_except(x real, y real)
2  RETURNS real AS
3  $$
4    DECLARE
5      ret real;
6    BEGIN
7      ret := x / y;
8      RETURN ret;
9    EXCEPTION
10     WHEN division_by_zero THEN
11       RAISE INFO 'DIVISION BY ZERO';
12       RAISE INFO 'Error % %', SQLSTATE, SQLERRM;
13       RETURN 0;
14   END;
15 $$
16 LANGUAGE 'plpgsql';
```

# Summary

- PostgreSQL supports powerful server-side functions.
- SQL and PL/pgSQL allow complex logic.
- Polymorphism and exception handling options are built-in.
- Functions are integral to performance and access control.

# Database Administration: SQL Functions.




Content has been extracted from *Learn PostgreSQL: Use, manage, and build secure and scalable databases with PostgreSQL 16 (Chapter 7)*, by Luca Ferrari & Enrico Pirozzi, 2023. Visit https://www.packtpub.com/en-co/product/learn-postgresql-9781837635641.