☑ 1. What is a Stored Procedure?

A **stored procedure** is a saved block of SQL and PL/pgSQL code that performs one or more actions.

In PostgreSQL, a **stored procedure** is a schema object that allows you to group a sequence of SQL statements and procedural logic (like loops and conditionals) into a single, callable unit.

While they are similar to functions, procedures have distinct capabilities, most notably the ability to manage transactions.

Key Characteristics

Transaction Control: Procedures can COMMIT and ROLLBACK transactions internally. This is the primary reason to use a procedure over a function.

No Return Value: Procedures do not return a single value. They can, however, return data via INOUT parameters.

Execution: They are executed using the CALL command rather than being used in a SELECT statement.

✔ What procedures can do

- Insert data
- Update data
- Delete data
- Run multiple SQL statements
- Print messages (`RAISE NOTICE`)
- Accept input parameters
- Give output using `OUT` parameters
- Manage transactions (unlike functions)

See also

- Advantages of Using Stored Procedures in PostgreSQL

! Difference between FUNCTION and PROCEDURE

Feature	FUNCTION	PROCEDURE
Returns a value	Yes	No direct return (can use OUT params)
Supports `CALL`	No	Yes
Transaction control (`COMMIT`, `ROLLBACK`)	❌ No	☑ Yes
Used for calculations	Common	Less common

★ 2. Creating a Simple Stored Procedure

Example 1: Display a welcome message

```
CREATE OR REPLACE PROCEDURE welcome_message()
LANGUAGE plpgsql
AS $$
BEGIN
    RAISE NOTICE 'Welcome to dvdrental database!';
END;
$$;
```

▶ Run it

```
CALL welcome_message();
```

⭐ 3. Stored Procedure With Input Parameter

🎯 Example 2: Show total rentals for a customer

This procedure takes a **customer_id** and prints the number of rentals.

```
CREATE OR REPLACE PROCEDURE get_customer_rentals(IN p_customer_id INT)
LANGUAGE plpgsql
AS $$
DECLARE
    rental_count INT;
BEGIN
    SELECT COUNT(*) INTO rental_count
    FROM rental
    WHERE customer_id = p_customer_id;

    RAISE NOTICE 'Customer % has total rentals: %', p_customer_id, rental_count;
END;
$$;
```

▶ Run it

```
CALL get_customer_rentals(1);
```

⭐ 4. Stored Procedure With INSERT Operation

🎯 Example 3: Add a new customer

This inserts a row into the `customer` table.

```
CREATE OR REPLACE PROCEDURE add_new_customer(
    IN p_first_name VARCHAR,
    IN p_last_name VARCHAR,
    IN p_email VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO customer(first_name, last_name, email, store_id, active,
create_date)
    VALUES (p_first_name, p_last_name, p_email, 1, 1, NOW());

    RAISE NOTICE 'New customer % % added successfully', p_first_name, p_last_name;
END;
$$;
```

▶ Run it

```
CALL add_new_customer('Ali', 'Khan', 'ali.khan@example.com');
```

⭐ 5. Stored Procedure With UPDATE Operation

🎯 Example 4: Update a film's rental rate

```
CREATE OR REPLACE PROCEDURE update_film_rental_rate(
    IN p_film_id INT,
    IN p_new_rate NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE film
    SET rental_rate = p_new_rate
    WHERE film_id = p_film_id;

    RAISE NOTICE 'Film % rental rate updated to %', p_film_id, p_new_rate;
END;
$$;
```

▶ Run it

```
CALL update_film_rental_rate(10, 4.99);
```

⭐ 6. Procedure With OUT Parameters

🎯 Example 5: Get film title and rental rate

```
CREATE OR REPLACE PROCEDURE get_film_info(
    IN p_film_id INT,
    OUT film_title VARCHAR,
    OUT rental_rate NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT title, rental_rate
    INTO film_title, rental_rate
    FROM film
    WHERE film_id = p_film_id;
END;
$$;
```

▶️ Run it

```
CALL get_film_info(5);
```

(pgAdmin will show OUT parameters in the Data Output panel.)

⭐ 7. Procedure With Loop

🎯 Example 6: Print all film titles for a given category

```
CREATE OR REPLACE PROCEDURE list_films_by_category(IN p_category VARCHAR)
LANGUAGE plpgsql
AS $$
DECLARE
    film_record RECORD;
BEGIN
    FOR film_record IN
        SELECT title
        FROM film f
        JOIN film_category fc ON f.film_id = fc.film_id
```

```
        JOIN category c ON c.category_id = fc.category_id
        WHERE c.name = p_category
    LOOP
        RAISE NOTICE 'Film: %', film_record.title;
    END LOOP;
END;
$$;
```

▶ Run it

```
CALL list_films_by_category('Action');
```

⭐ 8. Procedure With DELETE Operation

🎯 Example 7: Delete old rental history

Deletes rental records older than given days.

```
CREATE OR REPLACE PROCEDURE delete_old_rentals(IN p_days INT)
LANGUAGE plpgsql
AS $$
BEGIN
    DELETE FROM rental
    WHERE rental_date < NOW() - INTERVAL '1 day' * p_days;

    RAISE NOTICE 'Old rentals older than % days deleted', p_days;
END;
$$;
```

▶ Run it

```
CALL delete_old_rentals(3000);
```

🎓 Summary for Students

Concept	Meaning
`CREATE PROCEDURE`	Create stored procedure
`CALL procedure_name()`	Execute procedure

Concept	Meaning
`RAISE NOTICE`	Print message
`IN` parameter	Input to procedure
`OUT` parameter	Output from procedure
`DECLARE`	Create local variables
`BEGIN … END`	Procedure code block
Loops & Cursors	Used to process multiple rows