Deccan Education Society's (DES)
Pune University, Pune
School of Engineering and Technology
Department of Computer Engineering and Technology
Program: B. Tech in Computer Science and Engineering

Academic Year: 2024-25	**Year:** Third Year	**Semester:** II
PRN No.: 1012412079	**Name: Ratnajeet Patil**	
Subject: Database Management System		
Assignment No.: 7		
Date:		

Lab Assignment: 07

Title: Stored Procedure and Functions
Write and Execute Stored Procedures to perform following kind of operations:

Theory:

What is Stored Procedure?

A **Stored Procedure** is a set of SQL statements that are stored in the database and can be executed repeatedly. It is a **routine** that can accept parameters, execute SQL queries, and perform tasks like inserting, updating, or deleting data. Stored procedures are mainly used to **encapsulate complex logic** and to **improve performance** by minimizing the number of calls made between the application and the database.

What is Function?

A **Function** is a stored routine that performs a specific task and **returns a value**. Functions are similar to stored procedures, but they are designed to **return a single value** (usually a scalar value or a table) and are usually used in SELECT queries or expressions.

Compare Stored Procedure with Function

Feature	**Stored Procedure**	**Function**
Return Type	Can return multiple result sets or output parameters (no return value by default)	Always returns a single value or a table
Can Modify Database	Yes, can perform actions like INSERT, UPDATE, DELETE	No, cannot modify the database (unless specifically allowed)
Usage in Queries	Cannot be used directly in a SELECT statement	Can be used in SELECT, WHERE, JOIN, or any SQL expression
Calling Method	Invoked explicitly using CALL or similar command	Invoked in SQL expressions or queries (e.g., SELECT function_name())
Side Effects	Can have side effects (e.g., modifying tables or changing the state of the database)	Should not have side effects, should only calculate and return a value
Return Value	No direct return value, but can return output parameters	Always returns a value (or set of values)
Transaction Control	Can control transactions (begin, commit, rollback)	Cannot control transactions

Execution:

1. Retrieve Records with Condition (Stored Procedure)

Question:
Create a stored procedure get_employees_by_dept that retrieves all employee records from the employees table where the department_id matches the input parameter.

2. Input and Output Parameters (Stored Procedure)

Question:
Write a stored procedure get_salary_by_id that takes an emp_id as an input parameter and returns the corresponding employee's name and salary as output parameters from the employees table.

3. Insert a New Record (Stored Procedure)

Question:
Design a stored procedure add_new_product to insert a new product into the products table. The procedure should accept parameters for product name, category, price, and quantity, and insert them into the table.

4. Conditional Logic with IF/CASE (Stored Procedure)

Question:
Create a stored procedure check_order_status that accepts an order_id and checks its status from the orders table.

- If the status is 'Pending', return a message 'Order is yet to be processed'.
- If the status is 'Shipped', return 'Order has been shipped'.
- If status is anything else, return 'Unknown status'.

Use IF or CASE statements for branching logic.

5. Calculation Using Function

Question:
Write a stored function calculate_bonus that accepts an employee's salary and bonus percentage as parameters, calculates the bonus amount, and returns the result.

6. String Manipulation (Function)

Question:
Design a stored function format_full_name that accepts first name and last name as input and returns a formatted full name in the format: LASTNAME, Firstname (all uppercase for last name).

7. Data Validation Before Insert (Stored Procedure)

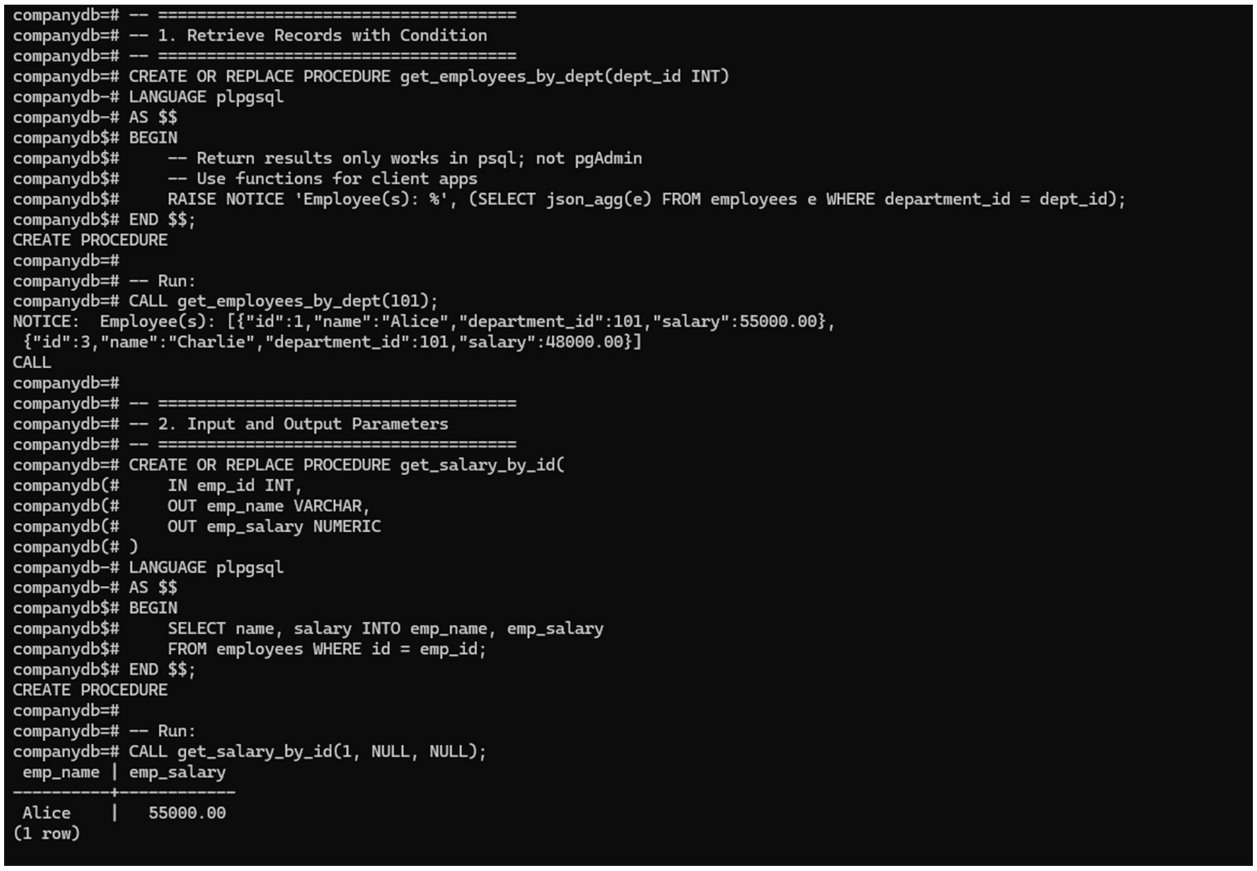
Question:
Develop a stored procedure register_user that accepts user details (username, email, age). Before inserting into the users table:

- Validate that age is at least 18.

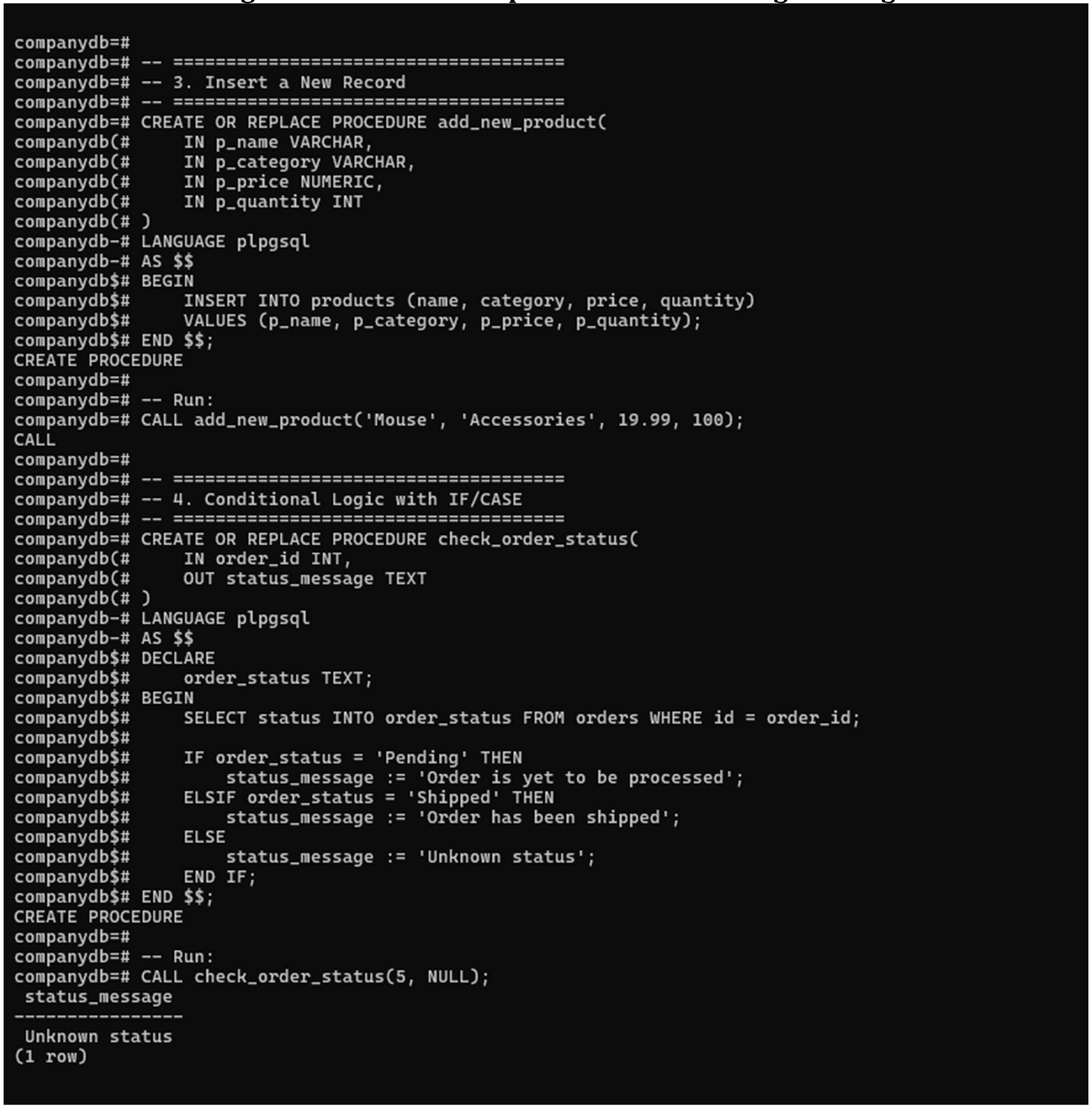
- Check that the username does not already exist.
  If validations pass, insert the record; otherwise, return an error message.

8. Error Handling in Stored Procedure

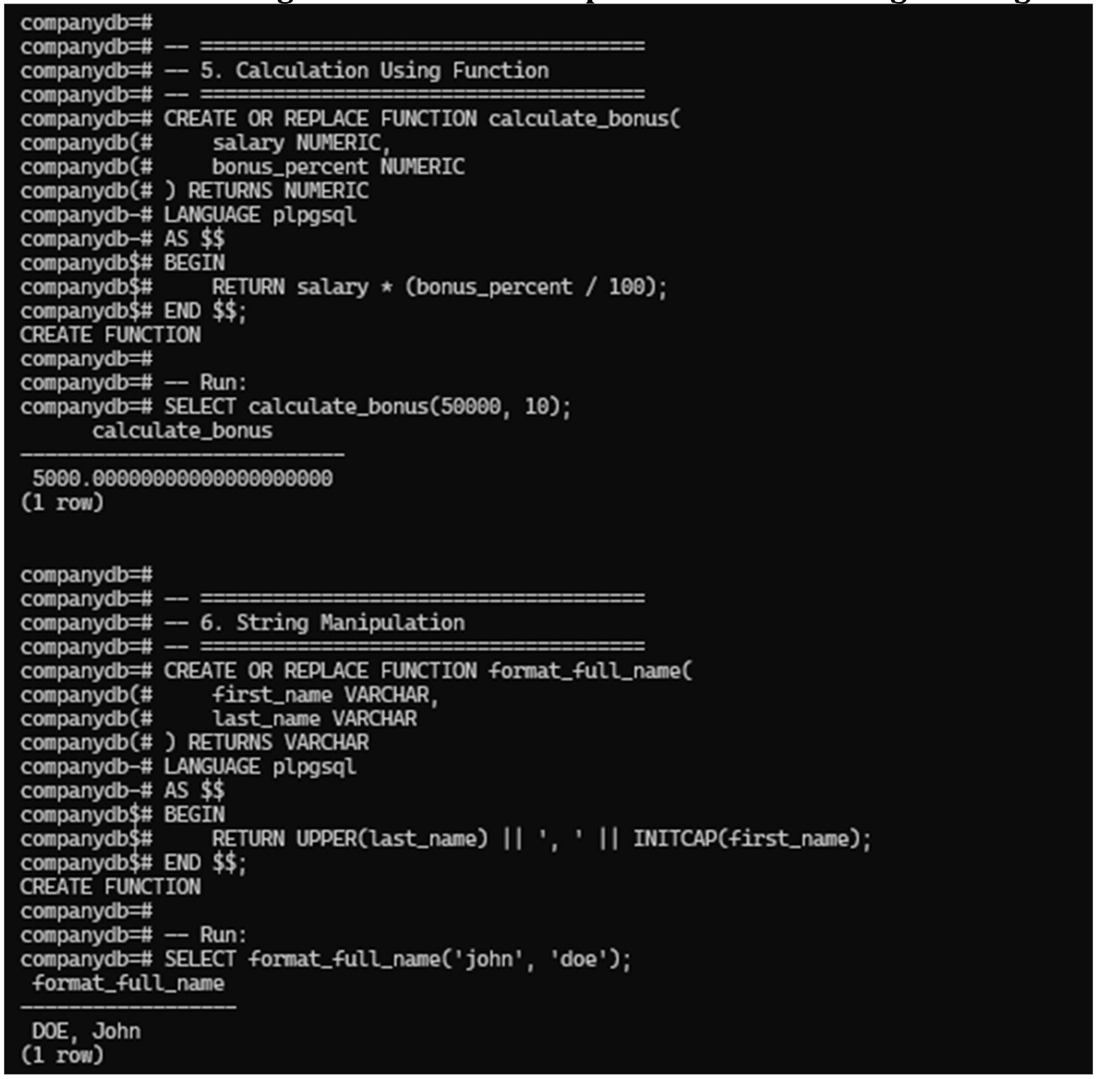
- **Question:**
  Write a stored procedure update_product_price that updates the price of a product in the products table.
  Include error handling mechanism to display appropriate error messages.



```
companydb=# -- ====================================
companydb=# -- 1. Retrieve Records with Condition
companydb=# -- ====================================
companydb=# CREATE OR REPLACE PROCEDURE get_employees_by_dept(dept_id INT)
companydb-# LANGUAGE plpgsql
companydb-# AS $$
companydb$# BEGIN
companydb$#     -- Return results only works in psql; not pgAdmin
companydb$#     -- Use functions for client apps
companydb$#     RAISE NOTICE 'Employee(s): %', (SELECT json_agg(e) FROM employees e WHERE department_id = dept_id);
companydb$# END $$;
CREATE PROCEDURE
companydb=#
companydb=# -- Run:
companydb=# CALL get_employees_by_dept(101);
NOTICE:  Employee(s): [{"id":1,"name":"Alice","department_id":101,"salary":55000.00},
 {"id":3,"name":"Charlie","department_id":101,"salary":48000.00}]
CALL
companydb=#
companydb=# -- ====================================
companydb=# -- 2. Input and Output Parameters
companydb=# -- ====================================
companydb=# CREATE OR REPLACE PROCEDURE get_salary_by_id(
companydb(#     IN emp_id INT,
companydb(#     OUT emp_name VARCHAR,
companydb(#     OUT emp_salary NUMERIC
companydb(# )
companydb-# LANGUAGE plpgsql
companydb-# AS $$
companydb$# BEGIN
companydb$#     SELECT name, salary INTO emp_name, emp_salary
companydb$#     FROM employees WHERE id = emp_id;
companydb$# END $$;
CREATE PROCEDURE
companydb=#
companydb=# -- Run:
companydb=# CALL get_salary_by_id(1, NULL, NULL);
 emp_name | emp_salary
----------+------------
 Alice    |   55000.00
(1 row)
```

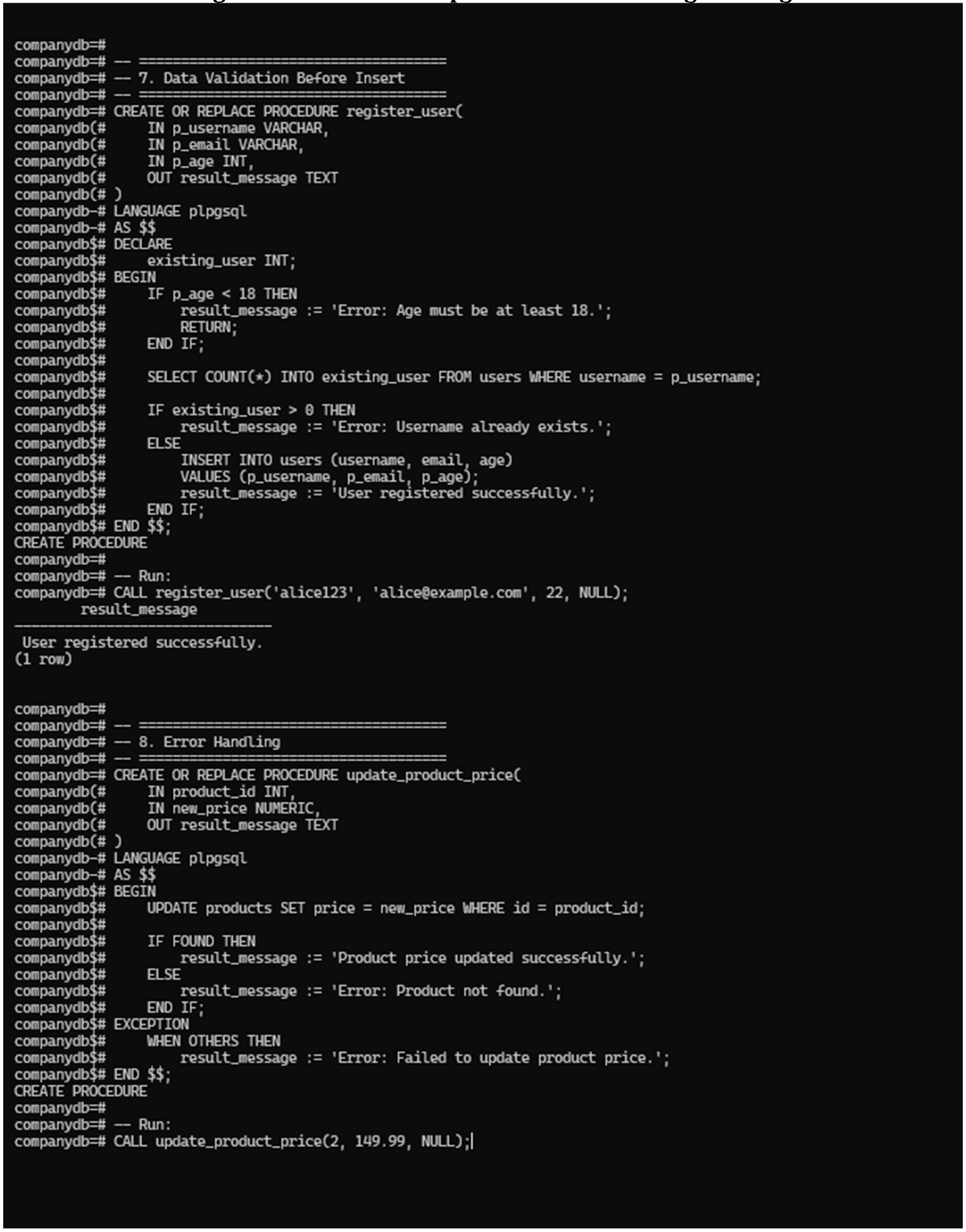


```
companydb=#
companydb=# -- =====================================
companydb=# -- 3. Insert a New Record
companydb=# -- =====================================
companydb=# CREATE OR REPLACE PROCEDURE add_new_product(
companydb(#     IN p_name VARCHAR,
companydb(#     IN p_category VARCHAR,
companydb(#     IN p_price NUMERIC,
companydb(#     IN p_quantity INT
companydb(# )
companydb-# LANGUAGE plpgsql
companydb-# AS $$
companydb$# BEGIN
companydb$#     INSERT INTO products (name, category, price, quantity)
companydb$#     VALUES (p_name, p_category, p_price, p_quantity);
companydb$# END $$;
CREATE PROCEDURE
companydb=#
companydb=# -- Run:
companydb=# CALL add_new_product('Mouse', 'Accessories', 19.99, 100);
CALL
companydb=#
companydb=# -- =====================================
companydb=# -- 4. Conditional Logic with IF/CASE
companydb=# -- =====================================
companydb=# CREATE OR REPLACE PROCEDURE check_order_status(
companydb(#     IN order_id INT,
companydb(#     OUT status_message TEXT
companydb(# )
companydb-# LANGUAGE plpgsql
companydb-# AS $$
companydb$# DECLARE
companydb$#     order_status TEXT;
companydb$# BEGIN
companydb$#     SELECT status INTO order_status FROM orders WHERE id = order_id;
companydb$#
companydb$#     IF order_status = 'Pending' THEN
companydb$#         status_message := 'Order is yet to be processed';
companydb$#     ELSIF order_status = 'Shipped' THEN
companydb$#         status_message := 'Order has been shipped';
companydb$#     ELSE
companydb$#         status_message := 'Unknown status';
companydb$#     END IF;
companydb$# END $$;
CREATE PROCEDURE
companydb=#
companydb=# -- Run:
companydb=# CALL check_order_status(5, NULL);
 status_message
----------------
 Unknown status
(1 row)
```



```
companydb=#
companydb=# -- ===================================
companydb=# -- 5. Calculation Using Function
companydb=# -- ===================================
companydb=# CREATE OR REPLACE FUNCTION calculate_bonus(
companydb(#     salary NUMERIC,
companydb(#     bonus_percent NUMERIC
companydb(# ) RETURNS NUMERIC
companydb-# LANGUAGE plpgsql
companydb-# AS $$
companydb$# BEGIN
companydb$#     RETURN salary * (bonus_percent / 100);
companydb$# END $$;
CREATE FUNCTION
companydb=#
companydb=# -- Run:
companydb=# SELECT calculate_bonus(50000, 10);
      calculate_bonus
---------------------------
 5000.00000000000000000000
(1 row)


companydb=#
companydb=# -- ===================================
companydb=# -- 6. String Manipulation
companydb=# -- ===================================
companydb=# CREATE OR REPLACE FUNCTION format_full_name(
companydb(#     first_name VARCHAR,
companydb(#     last_name VARCHAR
companydb(# ) RETURNS VARCHAR
companydb-# LANGUAGE plpgsql
companydb-# AS $$
companydb$# BEGIN
companydb$#     RETURN UPPER(last_name) || ', ' || INITCAP(first_name);
companydb$# END $$;
CREATE FUNCTION
companydb=#
companydb=# -- Run:
companydb=# SELECT format_full_name('john', 'doe');
 format_full_name
------------------
 DOE, John
(1 row)
```



```
companydb=#
companydb=# -- ===================================
companydb=# -- 7. Data Validation Before Insert
companydb=# -- ===================================
companydb=# CREATE OR REPLACE PROCEDURE register_user(
companydb(#     IN p_username VARCHAR,
companydb(#     IN p_email VARCHAR,
companydb(#     IN p_age INT,
companydb(#     OUT result_message TEXT
companydb(# )
companydb-# LANGUAGE plpgsql
companydb-# AS $$
companydb$# DECLARE
companydb$#     existing_user INT;
companydb$# BEGIN
companydb$#     IF p_age < 18 THEN
companydb$#         result_message := 'Error: Age must be at least 18.';
companydb$#         RETURN;
companydb$#     END IF;
companydb$#
companydb$#     SELECT COUNT(*) INTO existing_user FROM users WHERE username = p_username;
companydb$#
companydb$#     IF existing_user > 0 THEN
companydb$#         result_message := 'Error: Username already exists.';
companydb$#     ELSE
companydb$#         INSERT INTO users (username, email, age)
companydb$#         VALUES (p_username, p_email, p_age);
companydb$#         result_message := 'User registered successfully.';
companydb$#     END IF;
companydb$# END $$;
CREATE PROCEDURE
companydb=#
companydb=# -- Run:
companydb=# CALL register_user('alice123', 'alice@example.com', 22, NULL);
        result_message
-------------------------------
 User registered successfully.
(1 row)


companydb=#
companydb=# -- ===================================
companydb=# -- 8. Error Handling
companydb=# -- ===================================
companydb=# CREATE OR REPLACE PROCEDURE update_product_price(
companydb(#     IN product_id INT,
companydb(#     IN new_price NUMERIC,
companydb(#     OUT result_message TEXT
companydb(# )
companydb-# LANGUAGE plpgsql
companydb-# AS $$
companydb$# BEGIN
companydb$#     UPDATE products SET price = new_price WHERE id = product_id;
companydb$#
companydb$#     IF FOUND THEN
companydb$#         result_message := 'Product price updated successfully.';
companydb$#     ELSE
companydb$#         result_message := 'Error: Product not found.';
companydb$#     END IF;
companydb$# EXCEPTION
companydb$#     WHEN OTHERS THEN
companydb$#         result_message := 'Error: Failed to update product price.';
companydb$# END $$;
CREATE PROCEDURE
companydb=#
companydb=# -- Run:
companydb=# CALL update_product_price(2, 149.99, NULL);
```

Conclusion:

FAQs:

1. What is PL/SQL?

   PL/SQL (Procedural Language/SQL) is Oracle's procedural extension to SQL. It combines the power of SQL with the procedural features of programming languages, such as variables, control structures (loops, conditionals), error handling, and more. PL/SQL allows users to write complex SQL operations within procedures, functions, and triggers.

2. What are different types of Parameters used in stored procedure?

   There are three main types of parameters used in stored procedures:
   - **IN Parameters**:
     - These are **input parameters**.
     - They provide data to the procedure when it is called.
     - They are **read-only** during the procedure execution.
     - Example: IN dept_id INT
   - **OUT Parameters**:
     - These are **output parameters**.
     - They are used to **return values** from the procedure back to the caller.
     - They are **initialized inside the procedure**.
     - Example: OUT result_message TEXT
   - **INOUT Parameters**:
     - These parameters act as both **input and output**.
     - They are used to **pass a value to the procedure** and also **receive a modified value** after the procedure execution.
     - Example: INOUT balance NUMERIC

3. Compare SQL and PL/SQL.

Feature	**SQL**	**PL/SQL**
Purpose	A **query language** used to manage and manipulate relational databases.	A **procedural language** used to write logic for executing SQL operations with control structures, error handling, etc.

Usage	Used for writing **queries** and **DML operations** (INSERT, UPDATE, DELETE).	Used to create **procedures**, **functions**, **triggers**, and control the flow of execution of SQL queries.
Execution Model	Executes one SQL statement at a time.	Executes a block of SQL statements as a program with **variables**, **loops**, and **conditionals**.
Control Structures	Does not support procedural constructs like loops or conditionals (except in queries like CASE).	Supports loops, conditionals, exception handling, and other procedural constructs.
Error Handling	Error handling is done using **SQLSTATE** codes or **exception clauses** (though limited).	Error handling is more robust with **exception blocks** that allow managing errors explicitly.
Performance	Executes individual SQL queries that can be optimized by the database.	**Performance** is enhanced by minimizing the number of calls to the database through batch operations.
Variables	Does not support the use of variables.	Supports variables, constants, and cursors.
Modifying Data	Can modify data through SQL commands like INSERT, UPDATE, and DELETE.	Can execute SQL commands and can also modify data, but more suited for complex logic.

4. How error handling works in stored procedure?

   Error handling in stored procedures can be done using **Exception Handling** in PL/SQL. The key component for error handling is the **EXCEPTION** block, which allows you to handle errors gracefully and take appropriate actions when an error occurs during procedure execution.