Part 1

--Exercise 1: Advanced Single-row Subquery

SELECT department\_name, first\_name, last\_name, salary

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

WHERE salary = (SELECT MAX(salary)

FROM employees

WHERE department\_id = e.department\_id);

--------------------------

SELECT d.department\_name, e.first\_name, e.last\_name, e.salary

FROM (

SELECT department\_id, MAX(salary) AS max\_salary

FROM employees

GROUP BY department\_id

) m

JOIN employees e ON e.department\_id = m.department\_id AND e.salary = m.max\_salary

JOIN departments d ON e.department\_id = d.department\_id;

------------------

--Exercise 2: Complex Multiple-row Subquery

--Task: List the employees who work in departments located in either New York or London.

SELECT first\_name, last\_name

FROM employees e

WHERE EXISTS (

SELECT 1

FROM departments d

JOIN locations l ON d.location\_id = l.location\_id

WHERE e.department\_id = d.department\_id

AND l.city in ('London', 'New York')

);

---------------------

SELECT e.employee\_id, e.first\_name, e.last\_name, l.city

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

JOIN locations l ON d.location\_id = l.location\_id

WHERE l.city IN ('New York', 'London');

—---------------------------------------------------------

--Exercise 3: Correlated Subquery for Performance Analysis

--Task: Find employees whose salary is above the departmental average, but only for departments located in the USA.

SELECT e.employee\_id, e.first\_name, e.last\_name, e.salary

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

JOIN locations l ON d.location\_id = l.location\_id

WHERE l.country = 'USA'

AND e.salary > (SELECT AVG(salary)

FROM employees

WHERE department\_id = e.department\_id) ;

—---------------------------------------

Part 2

--Exercise 1: Subqueries in INSERT Statements

--Task: Insert new employees into a promotion list if their salary is above the department average.

INSERT INTO promotion\_list (employee\_id, first\_name, last\_name, salary)

SELECT employee\_id, first\_name, last\_name, salary

FROM employees e

WHERE salary > (SELECT AVG(salary)

FROM employees

WHERE department\_id = e.department\_id);

--------------------------------------------------------------------------------

--Exercise 2: Subqueries in UPDATE Statements

--Task: Increase the salary of employees in departments with below-average total sales.

UPDATE employees

SET salary = salary \* 1.05

WHERE department\_id IN (SELECT department\_id

FROM departments

WHERE (SELECT SUM(sales)

FROM sales

WHERE sales.department\_id = departments.department\_id) < (SELECT AVG(total\_sales)

FROM (SELECT SUM(sales) AS total\_sales

FROM sales

GROUP BY department\_id) subquery));

-------------------------------------------------------------------------------------------------------------------------------

--Exercise 3: Subqueries in DELETE Statements

--Task: Delete products from inventory that have not been ordered in the last year.

DELETE FROM inventory

WHERE product\_id NOT IN (SELECT product\_id

FROM orders

WHERE order\_date >= DATE\_SUB(CURDATE(), INTERVAL 1 YEAR));

-----------------------------------------------------------------------------------

--Exercise 4: Complex Nested Subqueries

--Task: Find employees who work in departments with total sales greater than the overall average sales but only if their own salary is above the department average.

SELECT employee\_id, first\_name, last\_name, salary

FROM employees e

WHERE department\_id IN (SELECT department\_id

FROM sales s

GROUP BY department\_id

HAVING SUM(s.sales) > (SELECT AVG(total\_sales)

FROM (SELECT SUM(sales) AS total\_sales

FROM sales

GROUP sBY department\_id) subquery))

AND salary > (SELECT AVG(salary)

FROM employees

WHERE department\_id = e.department\_id);

—----------------------------------------------------------------------------

Part 3

1.-- Create the original products table

CREATE TABLE products (

product\_id SERIAL PRIMARY KEY, -- Auto-incrementing product ID

product\_name VARCHAR(100) NOT NULL, -- Product name (cannot be null)

price DECIMAL(10, 2) NOT NULL -- Product price (cannot be null)

);

-- Insert sample data into the products table

INSERT INTO products (product\_name, price)

VALUES

('Laptop', 1200.00),

('Smartphone', 800.00),

('Tablet', 400.00),

('Headphones', 150.00),

('Smartwatch', 250.00);

-- Verify the data in the products table

SELECT \* FROM products;

-- Step 1: Create a temporary table

CREATE TEMP TABLE temp\_products (

product\_id SERIAL PRIMARY KEY,

product\_name VARCHAR(100),

original\_price DECIMAL(10, 2),

increased\_price DECIMAL(10, 2)

);

-- Step 2: Insert data into the temporary table

INSERT INTO temp\_products (product\_name, original\_price, increased\_price)

SELECT

product\_name,

price AS original\_price,

price \* 1.10 AS increased\_price -- Increase price by 10%

FROM

products;

-- Step 3: Query the temporary table

SELECT \* FROM temp\_products;

-- Step 4: Perform additional operations on the temporary table (if needed)

-- Example: Filter products with increased price above a certain value

SELECT \*

FROM temp\_products

WHERE increased\_price > 100;

-- Temporary tables are automatically dropped when the session ends.

—---------------------------------------

--3.2

-- Create the main orders table

CREATE TABLE orders (

order\_id SERIAL PRIMARY KEY, -- Auto-incrementing order ID

order\_date DATE NOT NULL, -- Order date

customer\_name VARCHAR(100) NOT NULL, -- Customer name

order\_total DECIMAL(10, 2) NOT NULL -- Order total amount

);

-- Insert sample data into the orders table

INSERT INTO orders (order\_date, customer\_name, order\_total)

VALUES

('2024-12-01', 'Alice', 150.00),

('2024-11-15', 'Bob', 300.00),

('2024-11-10', 'Charlie', 200.00),

('2024-10-25', 'Diana', 450.00),

('2024-12-20', 'Eve', 100.00);

-- Verify the data in the orders table

SELECT \* FROM orders;

-- Create a global temporary-like table

CREATE TEMP TABLE temp\_orders (

order\_id SERIAL PRIMARY KEY, -- Auto-incrementing order ID

order\_date DATE NOT NULL, -- Order date

customer\_name VARCHAR(100) NOT NULL, -- Customer name

order\_total DECIMAL(10, 2) NOT NULL -- Order total amount

);

-- Insert orders from the past month into the temporary table

INSERT INTO temp\_orders (order\_date, customer\_name, order\_total)

SELECT

order\_id, order\_date, customer\_name, order\_total

FROM

orders

WHERE

order\_date >= CURRENT\_DATE - INTERVAL '1 month';

-- Query the temporary table to verify the data

SELECT \* FROM temp\_orders;

—---------------------------------------------------------------------------------------

--3.4

-- Create the customers table with sample data

CREATE TABLE customers (

customer\_id SERIAL PRIMARY KEY, -- Unique customer ID

first\_name VARCHAR(50) NOT NULL, -- Customer first name

last\_name VARCHAR(50) NOT NULL, -- Customer last name

total\_spent DECIMAL(10, 2), -- Total amount spent by the customer

last\_purchase\_date DATE -- Date of the customer's last purchase

);

-- Insert sample data into the customers table

INSERT INTO customers (first\_name, last\_name, total\_spent, last\_purchase\_date)

VALUES

('Alice', 'Brown', 1500.50, '2024-11-15'),

('Bob', 'Smith', 800.00, '2024-12-01'),

('Charlie', 'Johnson', 2500.00, '2024-11-20'),

('Diana', 'White', 600.00, '2024-10-30'),

('Eve', 'Black', 3000.00, '2024-12-10');

-- Verify the data

SELECT \* FROM customers;

-- Create a local temporary table for staging and transforming customer data

CREATE TEMP TABLE temp\_customers (

customer\_id SERIAL PRIMARY KEY, -- Unique customer ID

full\_name VARCHAR(100), -- Concatenated full name

total\_spent DECIMAL(10, 2), -- Total amount spent

last\_purchase\_date DATE, -- Date of last purchase

customer\_value\_category VARCHAR(50) -- Categorized customer value

);

-- Populate the temporary table with transformed data

INSERT INTO temp\_customers (full\_name, total\_spent, last\_purchase\_date, customer\_value\_category)

SELECT

CONCAT(first\_name, ' ', last\_name) AS full\_name, -- Concatenate first and last name

total\_spent,

last\_purchase\_date,

CASE

WHEN total\_spent >= 2000 THEN 'High-Value'

WHEN total\_spent >= 1000 THEN 'Medium-Value'

ELSE 'Low-Value'

END AS customer\_value\_category

FROM

customers;

-- Verify the data in the temporary table

SELECT \* FROM temp\_customers;

-- Filter high-value customers from the temporary table

SELECT

full\_name,

total\_spent,

last\_purchase\_date,

customer\_value\_category

FROM

temp\_customers

WHERE

customer\_value\_category = 'High-Value';