**Exercise 1: Creating and Using Temporary Tables**

Task:

Create a temporary table named **temp\_table** with columns **id** and **value**. Insert some sample data into it, and then retrieve and display the contents.

**Solution:**

-- Create temporary table

CREATE TEMPORARY TABLE temp\_table (

id INT,

value VARCHAR(255)

);

-- Insert sample data

INSERT INTO temp\_table (id, value) VALUES (1, 'First'), (2, 'Second'), (3, 'Third');

-- Retrieve and display contents

SELECT \* FROM temp\_table;

**Exercise 2: Using Temporary Tables in a Query**

Task:

Create a temporary table **even\_numbers** that stores even numbers from 1 to 10. Then, use this temporary table to retrieve and display only the even numbers from a separate table named **numbers\_table**.

Solution:

create temporary table even\_numbers(

number int not null

);

create table numbers\_table(

number int not null

);

insert into numbers\_table values(1),(2),(3),(4),(5),(6),(7),(8),(9),(10);

insert into even\_numbers

select \* from numbers\_table

where number % 2 = 0;

-- Retrieve and display even numbers from numbers\_table using the temporary table

[SELECT n.\*

FROM numbers\_table n

JOIN even\_numbers e ON n.number = e.number;]

**Exercise 3: Aggregating Data Using Temporary Tables**

Task:

Create a temporary table **monthly\_sales** that contains the total sales for each month. Use this temporary table to find the month with the highest sales.

You need to create this table from the orders table of northwind database.

-- Create temporary table for monthly sales

create temporary table monthly\_sales

as select

month(orders.orderDate) as month,

sum(order\_details.unitprice \* order\_details.quantity) as total\_sales

from orders join order\_details using(orderid)

group by month;

select \* from monthly\_sales

order by total\_sales desc

limit 1;

**Exercise 4: Create a Temporary Table for Expensive Products**

Task:

Create a temporary table named **expensive\_products** to store products with a unit price greater than $50. Insert the relevant data, and then retrieve and display the product information.

Solution:

-- Create temporary table for expensive products

CREATE TEMPORARY TABLE expensive\_products AS

SELECT \*

FROM products

WHERE unit\_price > 50;

-- Retrieve and display expensive products

SELECT \*

FROM expensive\_products;

**Exercise 5: Calculate Order Totals Using Temporary Tables**

Task:

Create a temporary table **order\_totals** to store the total amount for each order. Insert the relevant data, and then retrieve and display the order numbers along with their total amounts.

Solution:

-- Create temporary table for order totals

create temporary table order\_totals(

orderid int not null primary key,

total\_amount decimal(12, 2)

);

insert into order\_totals

select

orderid,

sum(unitprice \* quantity) as total\_amount

from order\_details

group by orderid;

select \* from order\_totals;

**Exercise 6: Identify Customers with Multiple Orders**

Task:

Create a temporary table **multiple\_orders\_customers** to store customers who have placed more than one order. Insert the relevant data, and then retrieve and display the customer information.

Solution:

-- Create temporary table for customers with multiple orders

CREATE TEMPORARY TABLE multiple\_orders\_customers AS

SELECT customer\_id, COUNT(DISTINCT order\_id) AS order\_count

FROM orders

GROUP BY customer\_id

HAVING order\_count > 1;

-- Retrieve and display customers with multiple orders

SELECT c.\*

FROM customers c

JOIN multiple\_orders\_customers m ON c.customer\_id = m.customer\_id;

**Exercise 7: Temporary Table Join for Product Categories**

Task:

Create a temporary table **category\_and\_products** to store product names along with their corresponding category names. Insert the relevant data, and then retrieve and display the product and category information.

Solution:

-- Create temporary table for product categories

CREATE TEMPORARY TABLE category\_and\_products AS

SELECT p.product\_name, c.category\_name

FROM products p

JOIN categories c ON p.category\_id = c.category\_id;

-- Retrieve and display product and category information

SELECT \*

FROM category\_and\_products;

**Exercise 8: Temporary Table for Employee Sales Performance**

Task:

Create a temporary table named **employee\_sales** to store the total sales amount for each employee. Insert the relevant data, and then retrieve and display employee names along with their total sales.

Solution:

-- Create temporary table for employee sales

CREATE TEMPORARY TABLE employee\_sales AS

SELECT e.employee\_id, CONCAT(e.first\_name, ' ', e.last\_name) AS employee\_name, SUM(od.unit\_price \* od.quantity) AS total\_sales

FROM employees e

JOIN orders o ON e.employee\_id = o.employee\_id

JOIN order\_details od ON o.order\_id = od.order\_id

GROUP BY e.employee\_id;

-- Retrieve and display employee sales

SELECT \*

FROM employee\_sales;

**Exercise 9: Temporary Table for Incomplete Orders**

Task:

Create a temporary table named **incomplete\_orders** to store information about orders that are missing some details (e.g., ship date). Insert the relevant data, and then retrieve and display the order information.

Solution:

-- Create temporary table for incomplete orders

CREATE TEMPORARY TABLE incomplete\_orders AS

SELECT o.order\_id, o.customer\_id, o.order\_date

FROM orders o

WHERE o.shipped\_date IS NULL;

-- Retrieve and display incomplete orders

SELECT \*

FROM incomplete\_orders;

**Exercise 10: Temporary Table for Customer Rankings**

Task:

Create a temporary table named **customer\_rankings** to store the ranking of customers based on the total amount they have spent. Insert the relevant data, and then retrieve and display customer names along with their rankings.

Solution:

-- Create temporary table for customer rankings

CREATE TEMPORARY TABLE customer\_rankings AS

SELECT c.customer\_id, CONCAT(c.company\_name, ' (', c.contact\_name, ')') AS customer\_name, SUM(od.unit\_price \* od.quantity) AS total\_spent

FROM customers c

JOIN orders o ON c.customer\_id = o.customer\_id

JOIN order\_details od ON o.order\_id = od.order\_id

GROUP BY c.customer\_id

ORDER BY total\_spent DESC;

-- Add a rank column

SET @rank = 0;

UPDATE customer\_rankings SET rank = (@rank := @rank + 1);

-- Retrieve and display customer rankings

SELECT \*

FROM customer\_rankings;