# A Comprehensive Exercise on Grouping, Filtering, and Analyzing Data in MySQL #50

Part 1

Task 1:

Query:

USE classicmodels;

SELECT productName, productLine

FROM products

WHERE

productLine LIKE 'Classic%Cars';

Explanation:

* Retrieves the product name and product line for products in the "Classic Cars" category.
* Utilizes the LIKE operator to filter product lines that start with "Classic."

Task 2:

Query:

SELECT customerName, addressLine1

FROM customers

WHERE

addressLine1 LIKE "%Street%"

OR addressLine1 LIKE "%Avenue%";

Explanation:

* Retrieves customer names and address lines for customers whose addresses contain either "Street" or "Avenue."
* Uses the LIKE operator with wildcard % for pattern matching.

Part 2

Task 1:

Query:

SELECT orderDetails.orderNumber, SUM(priceEach \* quantityOrdered) AS totalAmount

FROM orderDetails, orders

WHERE

orderDetails.orderNumber = orders.orderNumber

GROUP BY

orderDetails.orderNumber

HAVING

totalAmount BETWEEN 1000 AND 10000;

Explanation:

* Calculates the total amount for each order and filters orders with total amounts between 1000 and 10000.
* Utilizes the GROUP BY clause and the HAVING clause for filtering aggregated results.

Task 2:

Query:

SELECT \*

FROM payments

WHERE

paymentDate BETWEEN '2004-01-01' AND '2004-03-31';

Explanation:

* Retrieves payment records for the specified date range between January 1, 2004, and March 31, 2004.

Part 3

Task 1:

Query:

SELECT orderNumber, SUM(quantityOrdered \* priceEach) AS orderTotal

FROM orderdetails

GROUP BY

orderNumber

HAVING

SUM(quantityOrdered \* priceEach) > ANY (

SELECT AVG(orderTotal)

FROM (

SELECT orderNumber, SUM(quantityOrdered \* priceEach) AS orderTotal

FROM orderdetails

GROUP BY

orderNumber

) AS average\_sales

);

Explanation:

* Calculates the total sales for each order and compares it to the average sales using a subquery.
* Uses the HAVING clause to filter orders with sales greater than any average sales.

Task 2:

Query:

SELECT \*

FROM products

WHERE

quantityInStock = ALL (

SELECT MAX(quantityInStock)

FROM products

);

Explanation:

* Retrieves products where the quantity in stock is equal to the maximum quantity in stock across all products.

Part 4

Task 1: Monthly Sales Growth

Query:

WITH

CustomerPayments AS (

SELECT

customerName, country, (

SELECT SUM(amount)

FROM payments p

WHERE

c.customerNumber = p.customerNumber

GROUP BY

p.customerNumber

HAVING

SUM(amount) > ANY (

SELECT 0.9 \* MAX(amount)

FROM payments

)

) AS paymentAmount

FROM customers c

WHERE

country LIKE 'USA%'

OR country LIKE 'Canada%'

Explanation:

* A query using subqueries to filter customers from the USA and Canada based on a payment amount condition.

Task 2: Quarterly Sales Analysis

Query:

SELECT

p.productCode,

p.productName,

AVG(od.quantityOrdered) AS annualAverage,

AVG(CASE

WHEN o.orderDate BETWEEN 'start\_date\_of\_season' AND 'end\_date\_of\_season' THEN od.quantityOrdered

ELSE NULL

END) AS seasonalAverage

FROM

products p

JOIN

orderdetails od ON p.productCode = od.productCode

JOIN

orders o ON od.orderNumber = o.orderNumber

GROUP BY

p.productCode, p.productName;

Explanation:

* Calculates the annual average and seasonal average quantity ordered for each product.
* Utilizes a CASE statement to calculate seasonal average within a specific date range.