Thhere is a small online store that sells various products. You need to analyze sales data to understand customer purchases better. Your goal is to:

1. Find the total sales for each product.
2. Identify which products have been sold more than a certain threshold.
3. Determine how many different customers have purchased each product.

**DDL (Data Definition Language)**

1. **Creating the Tables**

sql

Copy code

CREATE TABLE customers (

customer\_id SERIAL PRIMARY KEY,

customer\_name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL

);

CREATE TABLE products (

product\_id SERIAL PRIMARY KEY,

product\_name VARCHAR(100) NOT NULL,

price NUMERIC(10, 2) NOT NULL

);

CREATE TABLE sales (

sale\_id SERIAL PRIMARY KEY,

customer\_id INT REFERENCES customers(customer\_id),

product\_id INT REFERENCES products(product\_id),

quantity INT NOT NULL,

sale\_date DATE NOT NULL

);

**Sample Data Insertion**

1. **Inserting Sample Data**

sql

Copy code

-- Inserting customers

INSERT INTO customers (customer\_name, email) VALUES

('John Doe', 'john@example.com'),

('Jane Smith', 'jane@example.com'),

('Alice Johnson', 'alice@example.com');

-- Inserting products

INSERT INTO products (product\_name, price) VALUES

('Laptop', 999.99),

('Smartphone', 499.99),

('Tablet', 299.99),

('Smartwatch', 199.99);

-- Inserting sales data

INSERT INTO sales (customer\_id, product\_id, quantity, sale\_date) VALUES

(1, 1, 1, '2024-01-15'),

(1, 2, 2, '2024-01-16'),

(2, 1, 1, '2024-01-17'),

(2, 3, 1, '2024-01-18'),

(3, 2, 1, '2024-01-19'),

(3, 3, 2, '2024-01-20'),

(1, 4, 3, '2024-01-21'); -- John purchased Smartwatch

**Queries**

1. **Using GROUP BY to Find Total Sales for Each Product**

sql

Copy code

SELECT p.product\_name,

SUM(s.quantity) AS total\_sales

FROM products p

JOIN sales s ON p.product\_id = s.product\_id

GROUP BY p.product\_name;

**Answer**: This query will return the total number of each product sold.

**Sample Output**

| **product\_name** | **total\_sales** |
| --- | --- |
| Laptop | 2 |
| Smartphone | 3 |
| Tablet | 3 |
| Smartwatch | 3 |

1. **Finding Products Sold More Than a Threshold**

To find products sold more than 2 units:

sql

Copy code

SELECT p.product\_name,

SUM(s.quantity) AS total\_sales

FROM products p

JOIN sales s ON p.product\_id = s.product\_id

GROUP BY p.product\_name

HAVING SUM(s.quantity) > 2;

**Answer**: This query filters products that have total sales greater than 2.

**Sample Output**

| **product\_name** | **total\_sales** |
| --- | --- |
| Smartphone | 3 |
| Tablet | 3 |
| Smartwatch | 3 |

1. **Using Subquery to Determine Customer Purchase Counts**

To find out how many different customers have purchased each product:

sql

Copy code

SELECT p.product\_name,

COUNT(DISTINCT s.customer\_id) AS customer\_count

FROM products p

JOIN sales s ON p.product\_id = s.product\_id

GROUP BY p.product\_name;

**Answer**: This query will return the count of distinct customers for each product.

**Sample Output**

| **product\_name** | **customer\_count** |
| --- | --- |
| Laptop | 2 |
| Smartphone | 2 |
| Tablet | 2 |
| Smartwatch | 1 |

**Summary**

* **GROUP BY**: Used to aggregate data based on specific columns (e.g., total sales per product).
* **HAVING**: Used to filter results after aggregation (e.g., products sold more than a certain threshold).
* **Subquery**: A nested query that provides data for the main query (e.g., counting distinct customers purchasing products).

This example provides a simple business case scenario, DDL statements, and SQL queries with answers that utilize GROUP BY, HAVING, and subqueries!