Write a query for the following

1. Write a query to create database with name pizza\_sales\_analysis.
2. Write a query to see list of all databases.
3. Write a query to create following table.

       Table Name: order

|  |  |
| --- | --- |
| Columns or Field | Datatype |
| id | int |
| date | date |

   4. Write a query to add column time with datatype time after date column in above order table.

   5. Write a query to rename the table order to orders.

   6. Add primary key constraint to the column id on existing table orders.

1. Create a Database Named pizza\_sales\_analysis

CREATE DATABASE pizza\_sales\_analysis;

2. View the List of All Databases

SHOW DATABASES;

*Note: This command is applicable in MySQL. If you're using a different database system, the command to list databases may vary.*

3. Create a Table Named order with Columns id and date

CREATE TABLE `order` (

id INT,

date DATE

);

*Note: The table name order is a reserved keyword in SQL. To avoid conflicts, it's enclosed in backticks. Alternatively, consider using a different table name.*

4. Add a Column time with Data Type TIME After the date Column

ALTER TABLE `order`

ADD COLUMN `time` TIME AFTER `date`;

5. Rename the Table order to orders

RENAME TABLE `order` TO `orders`;

6. Add a Primary Key Constraint to the Column id on the Existing Table orders

ALTER TABLE `orders`

ADD PRIMARY KEY (id);

INSERT INTO customers (name, phone, address, pincode)

VALUES ('Jiten Turalkar', '9090909090', 'Mumbai', '400003');

INSERT INTO Products (name, category, price, stock\_quantity, is\_available)

VALUES ('Wireless Mouse', 'Electronics', 799, 50, TRUE);

UPDATE Products

SET stock\_quantity = 200

WHERE name = 'Basmati Rice 5kg';

UPDATE Orders

SET total\_amount = 2500.00

WHERE order\_id = 15;

DELETE FROM Products

WHERE category = 'Personal Care';

Write a query to fetch all customer names and phone numbers from Customers Table.

SELECT name, phone

FROM Customers;

Write a query to fetch complete details of customers who live in the pincode 380009.

SELECT customer\_id, name, phone, address, pincode

FROM Customers

WHERE pincode = '380009';

Write a query to display all product records from the Products table where the price is less than ₹20. The result should include all columns for budget-friendly items.

SELECT \*

FROM Products

WHERE price < 20;

Write a query to list the names and available stock quantity of all products that belong to the Grocery Category.

SELECT name, stock\_quantity

FROM Products

WHERE category = 'Grocery';

Write a query to fetch all order records where the total order amount is greater than ₹300.

SELECT order\_id, customer\_id, order\_date, delivery\_pincode, total\_amount

FROM Orders

WHERE total\_amount > 300;

Write a query to find the names of customers whose names start with the letter S. Use the LIKE operator for pattern matching.

SELECT name

FROM Customers

WHERE name LIKE 'S%';

Write a query to fetch the details of products that belong to either the Snack or Grocery categories. Use the IN clause to filter multiple categories.

SELECT \*

FROM Products

WHERE category IN ('Snack', 'Grocery');

Write a query to display all products sorted by price in ascending order. This will help identify the cheapest to most expensive products.

SELECT \*

FROM Products

ORDER BY price ASC;

Write a query to fetch the 5 most recent orders from the Orders table based on the order date. Display full order details sorted by newest orders first.

SELECT \*

FROM Orders

ORDER BY order\_date DESC

LIMIT 5;

Write a query to retrieve the names and addresses of the first two customers whose names start with the letter R, sorted alphabetically by name. fetch from Customers Table.

SELECT name, address

FROM Customers

WHERE name LIKE 'R%'

ORDER BY name ASC

LIMIT 2;

Write a query to display the product reviews where customers have given a rating of 5. Return the product ID, customer ID, and the review comment, ratings from the QuickReviews table.

SELECT product\_id, customer\_id, comment, rating

FROM QuickReviews

WHERE rating = 5;

Write an SQL query to retrieve the distinct names and categories of products from the Products table. Only include products that are currently available (is\_available = 1) and priced above ₹200. The query should return the results with the column names ProductName and ProductCategory.

SELECT DISTINCT

name AS ProductName,

category AS ProductCategory

FROM Products

WHERE is\_available = 1

AND price > 200;

Unlock the Built-Ins

Write a query to get the full name of customers by combining their name and address. Return Name, Address, and the alias FullDetails. Use the CONCAT function to join name and address with a hyphen.

SELECT

  name,

  address,

  CONCAT(name, ' - ', address) AS FullDetails

FROM Customers;

Write a query to display product names and prices after applying a 10% discount. Return Name, Price, and the alias DiscountedPrice.

SELECT

  name AS Name,

  price AS Price,

  price \* 0.9 AS DiscountedPrice

FROM Products;

Write a query to fetch the full product name and the first 4 letters of each name as the type\_code.

SELECT

  name AS name,

  SUBSTRING(name, 1, 4) AS type\_code

FROM Products;

Write a query to fetch each product’s name along with the number of characters in the name. Label the calculated length as name\_length.

SELECT

  name,

  LENGTH(name) AS name\_length

FROM Products;

Write query show the latest 5 orders from the Orders Table. The results should be sorted by the order date in descending order

SELECT \*

FROM Orders

ORDER BY order\_date DESC

LIMIT 5;

 Connecting Data for Insights

write an sql query to generate a list of all customers orders. for each order include order Id, customer name and total amount.

SELECT

o.order\_id,

c.name AS customer\_name,

o.order\_date,

o.total\_amount

FROM Orders o

JOIN Customers c

ON o.customer\_id = c.customer\_id;

Write an SQL query to display each products name, the total quantity sold, and the number of times it was ordered, grouped by product

SELECT

p.name AS name,

SUM(oi.quantity) AS total\_sold,

COUNT(\*) AS order\_count

FROM Products p

JOIN OrderItems oi

ON p.product\_id = oi.product\_id

GROUP BY p.product\_id, p.name;

**How it works**:

* JOIN ensures each OrderItems row is matched to its product in Products.
* SUM(oi.quantity) computes the total units sold per product.
* COUNT(\*) counts the number of order-line entries per product (i.e. how many times the product was ordered).
* GROUP BY p.product\_id, p.name ensures aggregation happens at the product level.

Write a query to fetch OrderID, OrderDate, and ProductName for orders that included products in the Snacks category.

SELECT DISTINCT

o.order\_id,

o.order\_date,

p.name AS ProductName

FROM Orders o

JOIN OrderItems oi

ON o.order\_id = oi.order\_id

JOIN Products p

ON oi.product\_id = p.product\_id

WHERE p.category = 'Snacks';

Why This Works

* JOIN Operations:
  + Orders o JOIN OrderItems oi ON o.order\_id = oi.order\_id—links each order to its items.
  + JOIN Products p ON oi.product\_id = p.product\_id—fetches product details for each item.
* Filtering by Category:
  + WHERE p.category = 'Snacks' ensures only items in the "Snacks" category are included.
* Avoiding Duplicates:
  + DISTINCT prevents the same order-product pair from appearing multiple times if the product appears multiple times within that order.

Write a query to fetch the List of ProductName and Rating for each product where rating is below 4.

SELECT

p.name AS ProductName,

qr.rating AS rating

FROM Products p

JOIN QuickReviews qr

ON p.product\_id = qr.product\_id

WHERE qr.rating < 4;

Why This Works

* JOIN: We use an INNER JOIN (simply written as JOIN) to combine rows from Products and QuickReviews where product\_id matches. This ensures each product is paired with its review(s).
* WHERE qr.rating < 4: Filters the joined rows to only include reviews with a rating less than 4.
* Selected Fields:
  + p.name AS ProductName: Retrieves product names and labels the column as ProductName.
  + qr.rating AS rating: Retrieves the rating value.

write query to fetch number of orders and total sales for each delivery pincode.

SELECT

delivery\_pincode,

COUNT(\*) AS total\_orders,

SUM(total\_amount) AS total\_sales

FROM Orders

GROUP BY delivery\_pincode;

Why This Works

* COUNT(\*): Counts how many orders exist per delivery\_pincode—this gives the total number of orders.
* SUM(total\_amount): Adds up the total\_amount values for each delivery\_pincode, giving you total sales.
* The GROUP BY delivery\_pincode clause aggregates the data correctly by unique pincode groups.  
  This pattern is a fundamental SQL technique for summarizing and analyzing grouped data.

write a query to fetch list of customers who have placed more than 3 orders along with their total spend

SELECT

c.name,

COUNT(o.order\_id) AS total\_orders,

SUM(o.total\_amount) AS total\_spent

FROM Customers c

JOIN Orders o

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

GROUP BY c.customer\_id, c.name

HAVING COUNT(o.order\_id) > 3;

How It Works

* JOIN Orders o ON c.customer\_id = o.customer\_id links each order to the corresponding customer.
* COUNT(o.order\_id) AS total\_orders tallies how many orders each customer has placed.
* SUM(o.total\_amount) AS total\_spent sums up the total amount each customer has spent.
* GROUP BY c.customer\_id, c.name aggregates this data by each customer.
* HAVING COUNT(o.order\_id) > 3 filters the results to include only those customers with more than three orders.