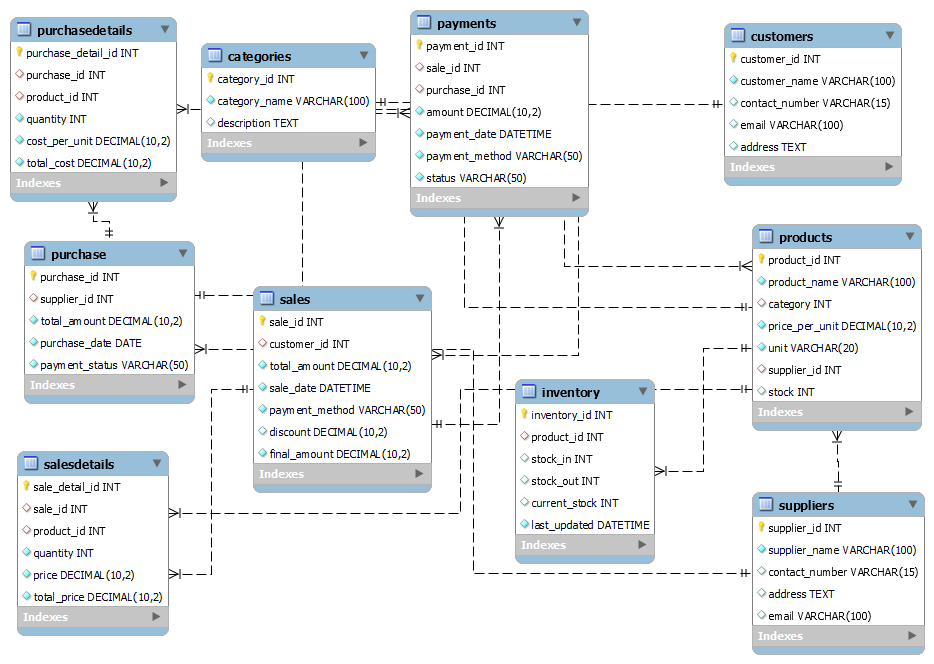
Kirana\_Store

Database Management System

A **Kirana Store** (or grocery store) database typically refers to a structured collection of data related to the operations, sales, inventory, and customer management of a small neighborhood grocery store. Such databases are essential for managing day-to-day operations efficiently and can help in analyzing sales trends, managing stock, and understanding customer preferences.

**Overview of a Kirana Store Database**

1. **Purpose**:
   * To keep track of inventory (items available for sale).
   * To manage customer data (loyalty programs, preferences).
   * To record sales transactions.
   * To analyze sales trends and stock levels.
   * To facilitate reordering of stock based on sales patterns.
2. **Key Components**:
   * **Tables**: The database typically consists of multiple tables that store different types of data. Common tables include:
     + **Customers**: Information about customers, including names, contact details, and purchase history.
     + **Products**: Details of items sold, such as product ID, name, category, price, and stock level.
     + **Categories**: Categories of products (e.g., dairy, grains, snacks) to help organize inventory.
     + **Sales**: Records of each sale transaction, including date, customer ID, total amount, and payment method.
     + **SalesDetails**: A table that breaks down each sale into its individual products, linking to the Sales table.
3. **Relationships**:
   * **One-to-Many Relationships**: For example, one customer can have multiple sales transactions, and one sale can include multiple products.
   * **Many-to-One Relationships**: Multiple products can belong to one category.
4. **Functionality**:
   * **Inventory Management**: Track quantities of products, update stock levels after sales, and reorder low-stock items.
   * **Customer Management**: Store customer information, manage loyalty programs, and analyze purchasing behavior.
   * **Sales Analysis**: Generate reports on sales performance, identify best-selling products, and analyze seasonal trends.
   * **Reporting**: Create reports for business analysis, such as sales by category, top customers, or inventory turnover rates.
5. **Technology**:
   * The database can be built using various database management systems (DBMS) like MySQL, PostgreSQL, Microsoft SQL Server, or SQLite, depending on the scale and requirements of the store.

**ER Diagram**

Create database **Kirana\_Store**;

use **Kirana\_Store**;

-- 1 Create **Categories** Table

CREATE TABLE **Categories** (

category\_id INT AUTO\_INCREMENT PRIMARY KEY,

category\_name VARCHAR(100) NOT NULL,

description TEXT

);

-- 2 Create **Suppliers** Table

CREATE TABLE **Suppliers** (

supplier\_id INT AUTO\_INCREMENT PRIMARY KEY,

supplier\_name VARCHAR(100) NOT NULL,

contact\_number VARCHAR(15),

address TEXT,

email VARCHAR(100)

);

-- 3 Create **Products** Table

CREATE TABLE **Products** (

product\_id INT AUTO\_INCREMENT PRIMARY KEY,

product\_name VARCHAR(100) NOT NULL,

category int,

price\_per\_unit DECIMAL(10, 2) NOT NULL,

unit VARCHAR(20) NOT NULL,

supplier\_id INT,

stock INT DEFAULT 0,

FOREIGN KEY (category) REFERENCES Categories(category\_id),

FOREIGN KEY (supplier\_id) REFERENCES Suppliers(supplier\_id)

);

-- 4 Create **Customers** Table

CREATE TABLE **Customers** (

customer\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_name VARCHAR(100) NOT NULL,

contact\_number VARCHAR(15),

email VARCHAR(100),

address TEXT

);

-- 5 Create **Sales** Table

CREATE TABLE **Sales** (

sale\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_id INT,

total\_amount DECIMAL(10, 2) NOT NULL,

sale\_date DATETIME NOT NULL,

payment\_method VARCHAR(50) NOT NULL,

discount DECIMAL(10, 2) DEFAULT 0,

final\_amount DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

);

-- 6 Create **SalesDetails** Table

CREATE TABLE **SalesDetails** (

sale\_detail\_id INT AUTO\_INCREMENT PRIMARY KEY,

sale\_id INT,

product\_id INT,

quantity INT NOT NULL,

price DECIMAL(10, 2) NOT NULL,

total\_price DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (sale\_id) REFERENCES Sales(sale\_id),

FOREIGN KEY (product\_id) REFERENCES Products(product\_id)

);

-- 7 Create **Inventory** Table

CREATE TABLE **Inventory** (

inventory\_id INT AUTO\_INCREMENT PRIMARY KEY,

product\_id INT,

stock\_in INT DEFAULT 0,

stock\_out INT DEFAULT 0,

current\_stock INT DEFAULT 0,

last\_updated DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (product\_id) REFERENCES Products(product\_id)

);

-- 8 Create **Purchase** Table

CREATE TABLE **Purchase** (

purchase\_id INT AUTO\_INCREMENT PRIMARY KEY,

supplier\_id INT,

total\_amount DECIMAL(10, 2) NOT NULL,

purchase\_date DATE NOT NULL,

payment\_status VARCHAR(50) NOT NULL,

FOREIGN KEY (supplier\_id) REFERENCES Suppliers(supplier\_id)

);

-- 9 Create **PurchaseDetails** Table

CREATE TABLE **PurchaseDetails** (

purchase\_detail\_id INT AUTO\_INCREMENT PRIMARY KEY,

purchase\_id INT,

product\_id INT,

quantity INT NOT NULL,

cost\_per\_unit DECIMAL(10, 2) NOT NULL,

total\_cost DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (purchase\_id) REFERENCES Purchase(purchase\_id),

FOREIGN KEY (product\_id) REFERENCES Products(product\_id)

);

-- 10 Create **Payments** Table

CREATE TABLE **Payments** (

payment\_id INT AUTO\_INCREMENT PRIMARY KEY,

sale\_id INT,

purchase\_id INT,

amount DECIMAL(10, 2) NOT NULL,

payment\_date DATETIME NOT NULL,

payment\_method VARCHAR(50) NOT NULL,

status VARCHAR(50) NOT NULL,

FOREIGN KEY (sale\_id) REFERENCES Sales(sale\_id),

FOREIGN KEY (purchase\_id) REFERENCES Purchase(purchase\_id)

);

-- Insert Values

INSERT INTO Products (product\_id, product\_name, category, price\_per\_unit, unit, supplier\_id, stock) VALUES

(1, 'Rice', 1, 50, 'kg', 1, 100),

(2, 'Sugar', 1, 45, 'kg', 2, 150),

(3, 'Milk', 2, 25, 'liter', 3, 200),

(4, 'Biscuits', 3, 10, 'piece', 4, 500),

(5, 'Tea', 4, 200, 'kg', 5, 50),

(6, 'Wheat Flour', 1, 40, 'kg', 1, 200),

(7, 'Butter', 2, 50, 'piece', 3, 120),

(8, 'Chips', 3, 20, 'packet', 4, 300),

(9, 'Green Tea', 4, 250, 'kg', 5, 40),

(10, 'Dish Soap', 5, 30, 'bottle', 6, 150);

INSERT INTO Suppliers (supplier\_id, supplier\_name, contact\_number, address, email) VALUES

(1, 'ABC Grains', '1234567890', 'Street 1, City A', 'abcgrains@example.com'),

(2, 'XYZ Sugars', '0987654321', 'Street 2, City B', 'xyzsugars@example.com'),

(3, 'Dairy Farm', '1122334455', 'Street 3, City C', 'dairyfarm@example.com'),

(4, 'Snack Foods Ltd.', '2233445566', 'Street 4, City D', 'snackfoods@example.com'),

(5, 'Tea House', '3344556677', 'Street 5, City E', 'teahouse@example.com'),

(6, 'Home Essentials', '4455667788', 'Street 6, City F', 'homeessentials@example.com'),

(7, 'Green Grocers', '5566778899', 'Street 7, City G', 'greengrocers@example.com'),

(8, 'SnackHub', '6677889900', 'Street 8, City H', 'snackhub@example.com'),

(9, 'Daily Needs', '7788990011', 'Street 9, City I', 'dailyneeds@example.com'),

(10, 'Fresh Foods', '8899001122', 'Street 10, City J', 'freshfoods@example.com');

INSERT INTO Customers (customer\_id, customer\_name, contact\_number, email, address) VALUES

(1, 'John Doe', '9990011223', 'john@example.com', 'Area 51, City F'),

(2, 'Jane Smith', '8881122334', 'jane@example.com', 'Area 52, City G'),

(3, 'Bob Johnson', '7772233445', 'bob@example.com', 'Area 53, City H'),

(4, 'Alice Williams', '6663344556', 'alice@example.com', 'Area 54, City I'),

(5, 'Mike Brown', '5554455667', 'mike@example.com', 'Area 55, City J'),

(6, 'Sam Taylor', '1112233445', 'sam@example.com', 'Area 56, City K'),

(7, 'Linda Brown', '2223344556', 'linda@example.com', 'Area 57, City L'),

(8, 'Tom Harris', '3334455667', 'tom@example.com', 'Area 58, City M'),

(9, 'Emily Clark', '4445566778', 'emily@example.com', 'Area 59, City N'),

(10, 'Michael Lee', '5556677889', 'michael@example.com', 'Area 60, City O');

INSERT INTO Sales (sale\_id, customer\_id, total\_amount, sale\_date, payment\_method, discount, final\_amount) VALUES

(1, 1, 500, '2024-10-01', 'Cash', 0, 500),

(2, 2, 750, '2024-10-02', 'UPI', 50, 700),

(3, 3, 300, '2024-10-03', 'Credit Card', 20, 280),

(4, 4, 450, '2024-10-04', 'Cash', 0, 450),

(5, 5, 1200, '2024-10-05', 'Debit Card', 100, 1100),

(6, 6, 650, '2024-10-06', 'Cash', 0, 650),

(7, 7, 500, '2024-10-07', 'Credit Card', 25, 475),

(8, 8, 800, '2024-10-08', 'UPI', 50, 750),

(9, 9, 1000, '2024-10-09', 'Debit Card', 0, 1000),

(10, 10, 350, '2024-10-10', 'Cash', 0, 350);

INSERT INTO SalesDetails (sale\_detail\_id, sale\_id, product\_id, quantity, price, total\_price) VALUES

(1, 1, 1, 5, 50, 250),

(2, 1, 3, 2, 25, 50),

(3, 2, 2, 10, 45, 450),

(4, 2, 4, 5, 10, 50),

(5, 3, 5, 1, 200, 200),

(6, 4, 3, 5, 25, 125),

(7, 5, 1, 10, 50, 500),

(8, 6, 6, 5, 40, 200),

(9, 6, 7, 3, 50, 150),

(10, 7, 8, 10, 20, 200),

(11, 7, 2, 5, 45, 225),

(12, 8, 9, 2, 250, 500),

(13, 8, 4, 5, 10, 50),

(14, 9, 3, 15, 25, 375),

(15, 9, 10, 10, 30, 300);

INSERT INTO Inventory (inventory\_id, product\_id, stock\_in, stock\_out, current\_stock, last\_updated) VALUES

(1, 1, 50, 10, 90, '2024-10-01'),

(2, 2, 100, 20, 130, '2024-10-02'),

(3, 3, 200, 50, 150, '2024-10-03'),

(4, 4, 500, 100, 400, '2024-10-04'),

(5, 5, 60, 10, 50, '2024-10-05'),

(6, 6, 200, 20, 180, '2024-10-06'),

(7, 7, 150, 30, 120, '2024-10-07'),

(8, 8, 300, 50, 250, '2024-10-08'),

(9, 9, 40, 10, 30, '2024-10-09'),

(10, 10, 150, 50, 100, '2024-10-10');

INSERT INTO Purchase (purchase\_id, supplier\_id, total\_amount, purchase\_date, payment\_status) VALUES

(1, 1, 2500, '2024-09-01', 'Paid'),

(2, 2, 4000, '2024-09-02', 'Unpaid'),

(3, 3, 5000, '2024-09-03', 'Paid'),

(4, 4, 600, '2024-09-04', 'Pending'),

(5, 5, 1200, '2024-09-05', 'Paid'),

(6, 6, 1800, '2024-09-06', 'Paid'),

(7, 7, 2500, '2024-09-07', 'Unpaid'),

(8, 8, 6000, '2024-09-08', 'Paid'),

(9, 9, 900, '2024-09-09', 'Pending'),

(10, 10, 3500, '2024-09-10', 'Paid');

INSERT INTO PurchaseDetails (purchase\_detail\_id, purchase\_id, product\_id, quantity, cost\_per\_unit, total\_cost) VALUES

(1, 1, 1, 50, 50, 2500),

(2, 2, 2, 100, 40, 4000),

(3, 3, 3, 200, 25, 5000),

(4, 4, 4, 60, 10, 600),

(5, 5, 5, 30, 40, 1200),

(6, 6, 6, 45, 40, 1800),

(7, 7, 8, 125, 20, 2500),

(8, 8, 9, 24, 250, 6000),

(9, 9, 10, 30, 30, 900),

(10, 10, 4, 350, 10, 3500);

INSERT INTO Payments (payment\_id, sale\_id, purchase\_id, amount, payment\_date, payment\_method, status) VALUES

(1, 1, NULL, 500, '2024-10-01', 'Cash', 'Completed'),

(2, NULL, 2, 4000, '2024-09-02', 'Bank Transfer', 'Pending'),

(3, 2, NULL, 700, '2024-10-02', 'UPI', 'Completed'),

(4, NULL, 4, 600, '2024-09-04', 'Credit Card', 'Pending'),

(5, 5, NULL, 1100, '2024-10-05', 'Debit Card', 'Completed'),

(6, 6, NULL, 650, '2024-10-06', 'Cash', 'Completed'),

(7, NULL, 7, 2500, '2024-09-07', 'Bank Transfer', 'Unpaid'),

(8, 8, NULL, 750, '2024-10-08', 'UPI', 'Completed'),

(9, NULL, 9, 900, '2024-09-09', 'Credit Card', 'Pending'),

(10, 10, NULL, 350, '2024-10-10', 'Cash', 'Completed');

INSERT INTO Categories (category\_id, category\_name, description) VALUES

(1, 'Grains', 'Includes rice, wheat, and other grains'),

(2, 'Dairy', 'Includes milk, cheese, and other dairy products'),

(3, 'Snacks', 'Includes biscuits, chips, and other snacks'),

(4, 'Beverages', 'Includes tea, coffee, and other beverages'),

(5, 'Household', 'Household essentials like soap, detergent'),

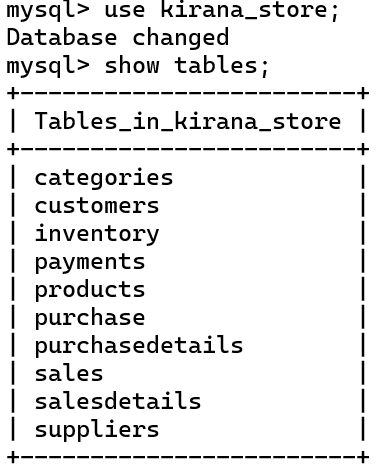
(6, 'Condiments', 'Spices, sauces, and other flavoring agents'),

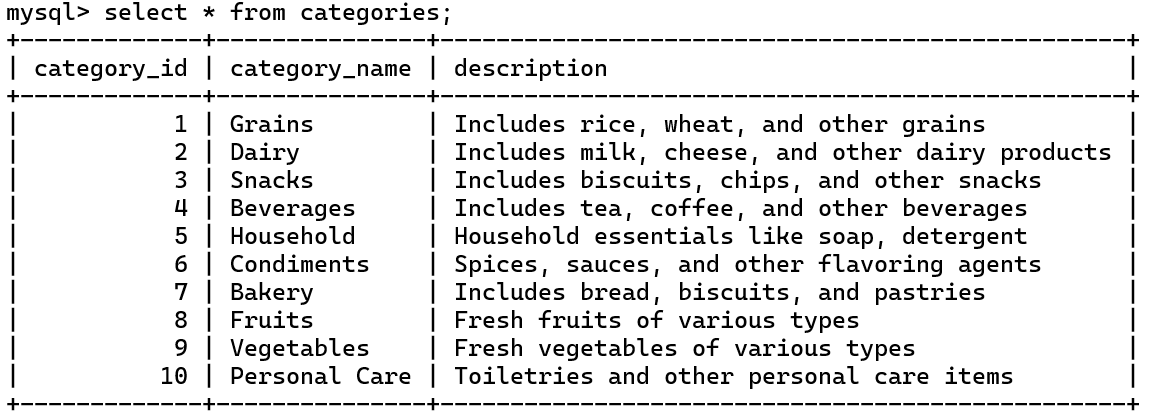
(7, 'Bakery', 'Includes bread, biscuits, and pastries'),

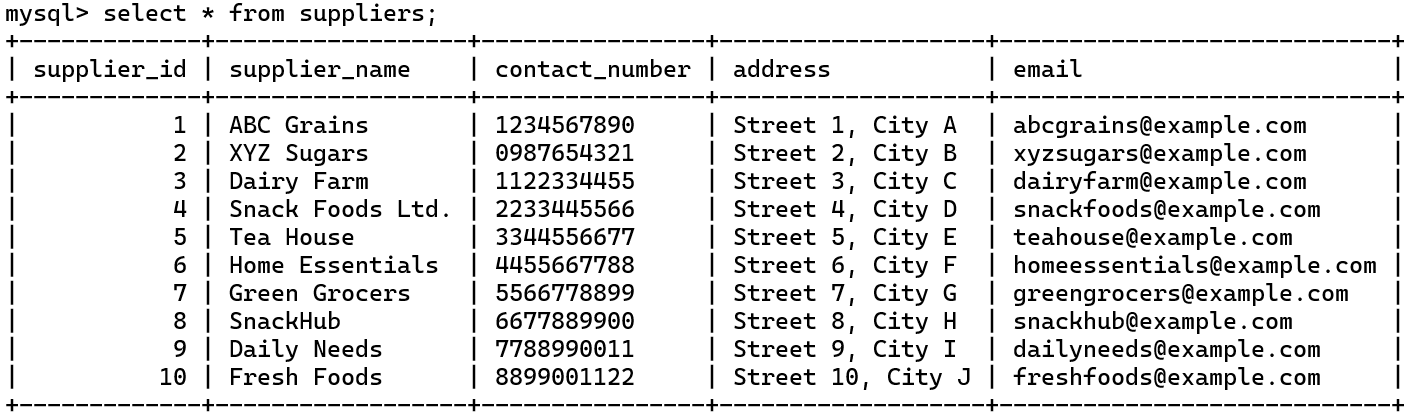
(8, 'Fruits', 'Fresh fruits of various types'),

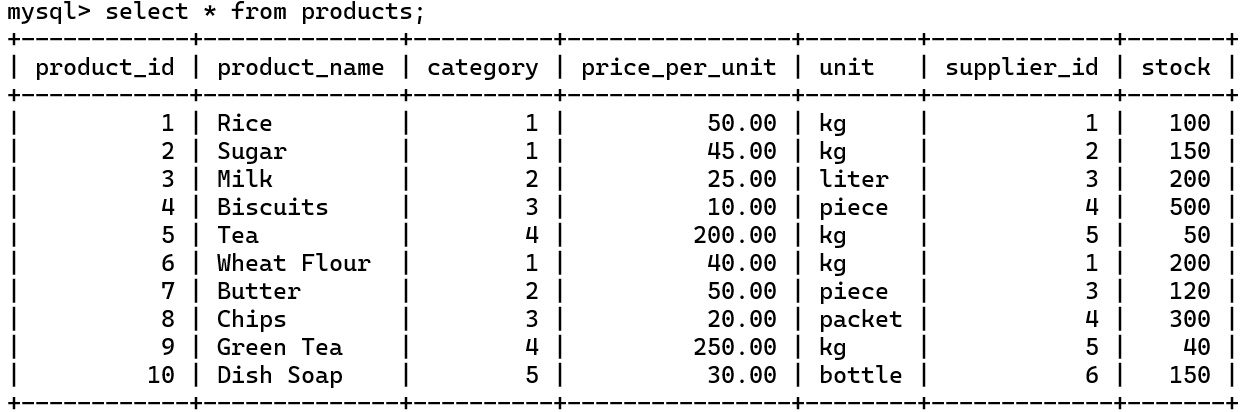
(9, 'Vegetables', 'Fresh vegetables of various types'),

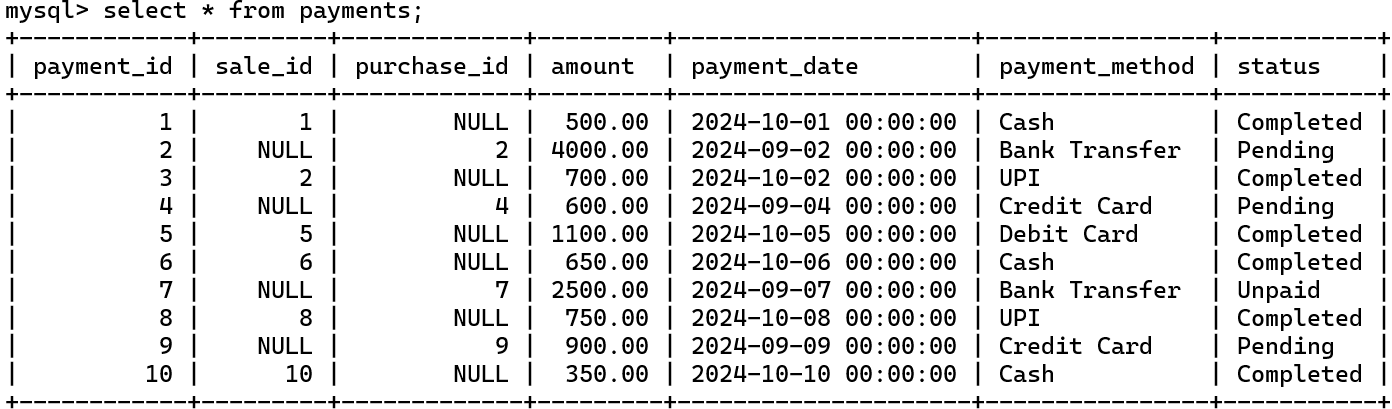
(10, 'Personal Care', 'Toiletries and other personal care items');

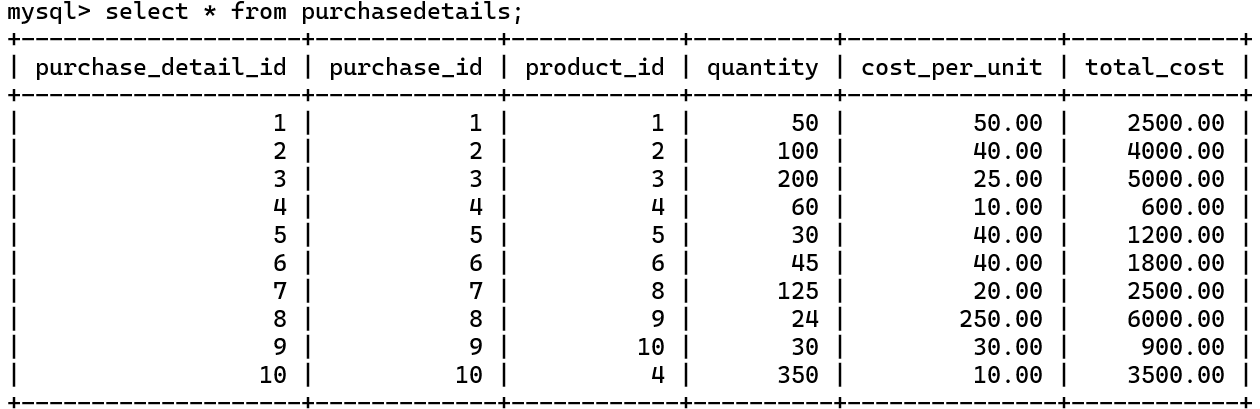
**SELECT & Desc**

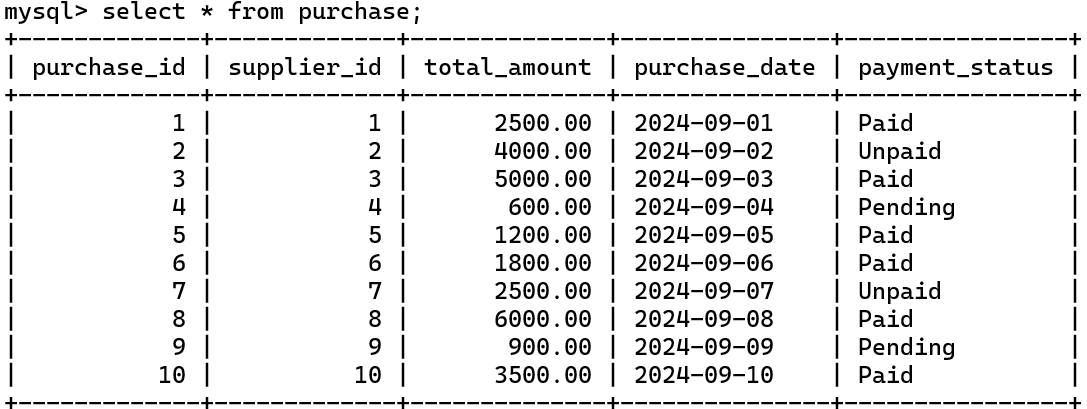
Select \* from Categories ;

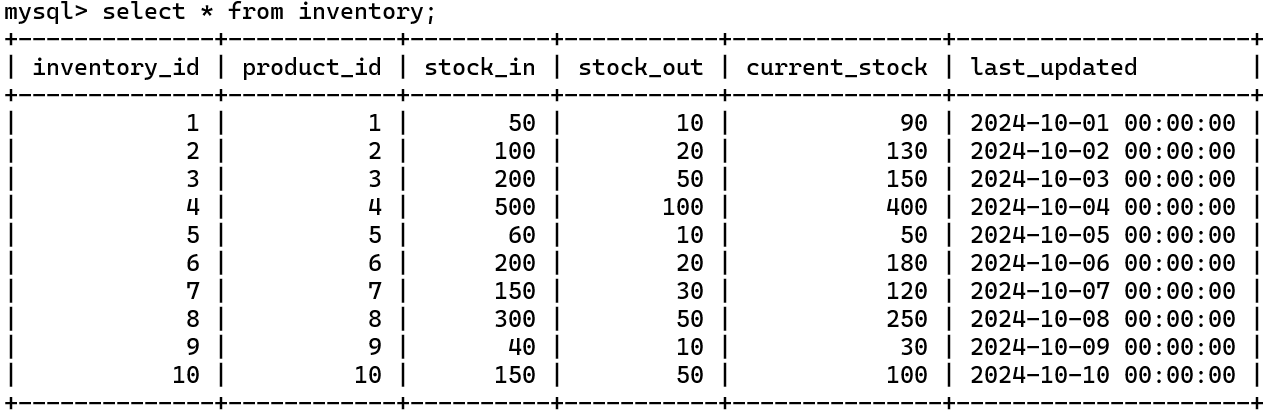
Select \* from Suppliers ;

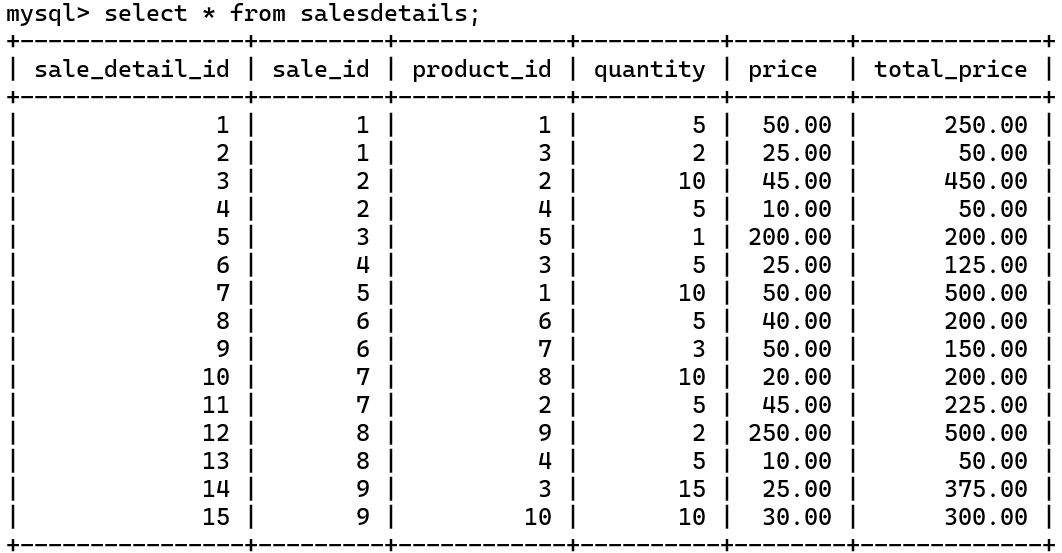
select \* from Products ;

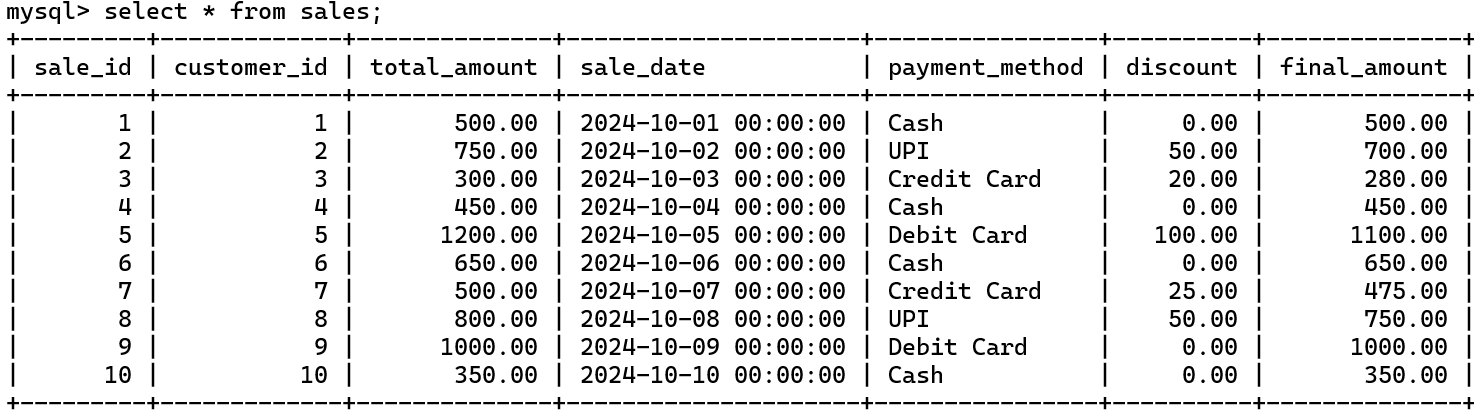
select \* from Payments ;

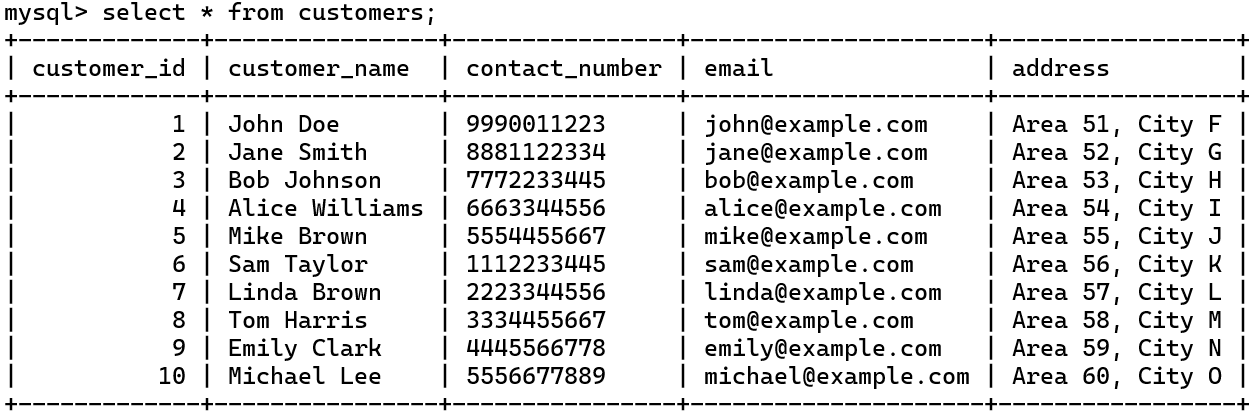
select \* from PurchaseDetails ;

select \* from Purchase ;

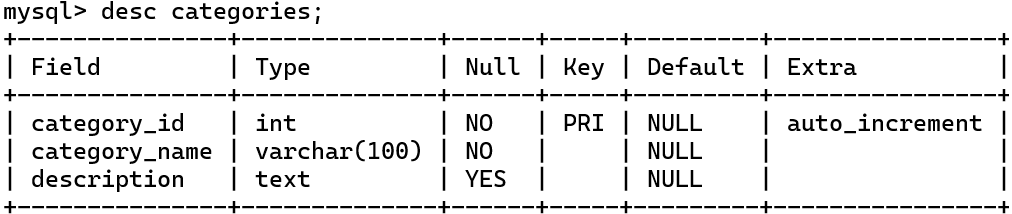
select \* from Inventory ;

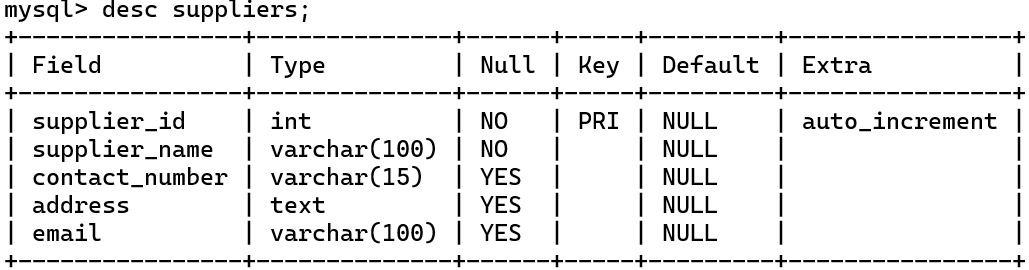
select \* from SalesDetails ;

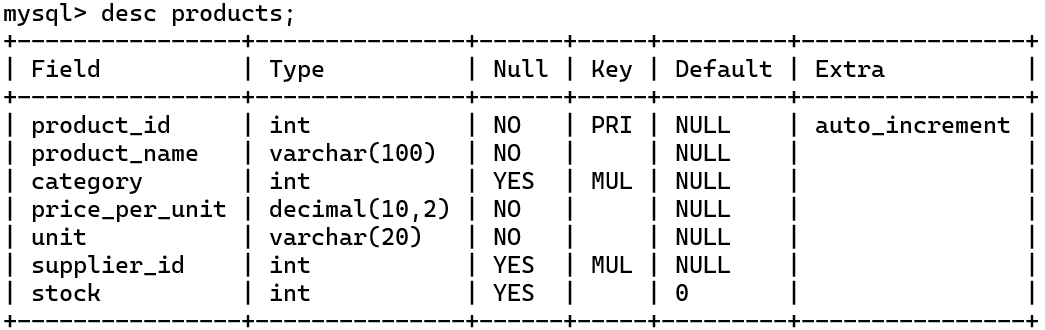
select \* from sales ;

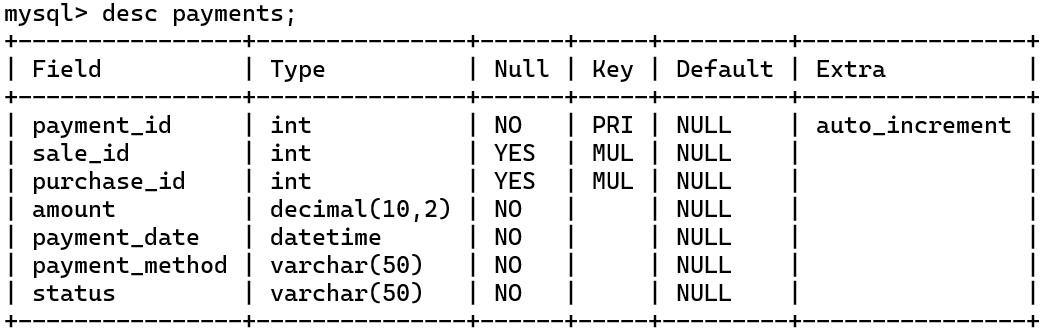
select \* from Customers ;

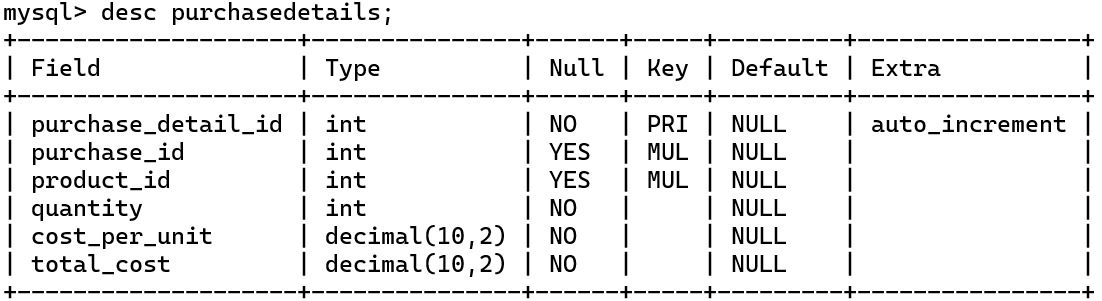
**DESC**

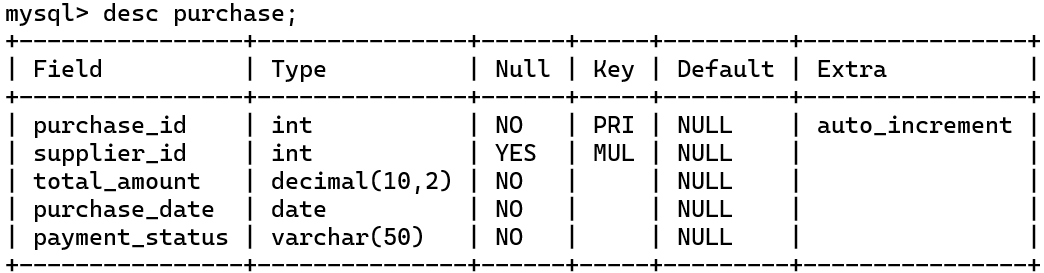
desc Categories ;

desc Suppliers ;

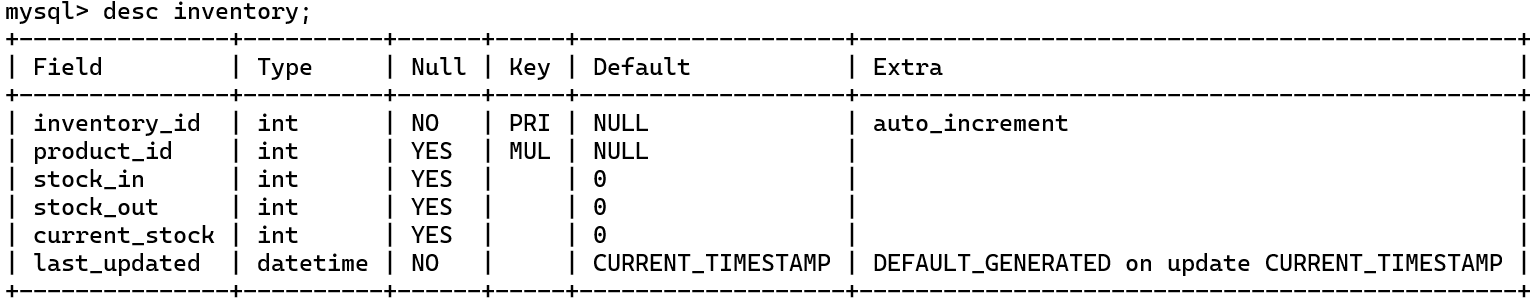
desc Products ;

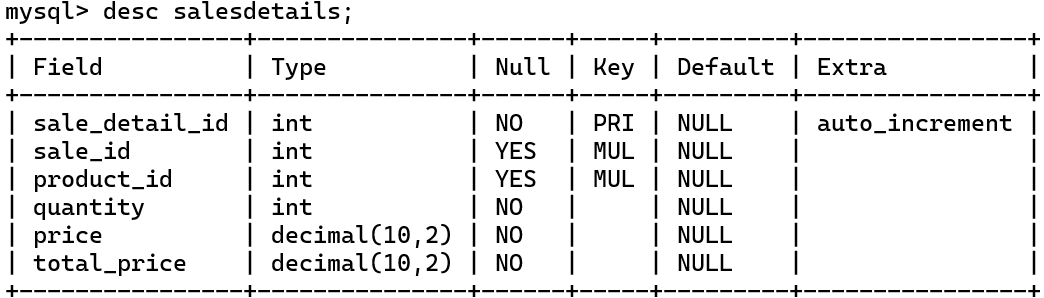
desc Payments ;

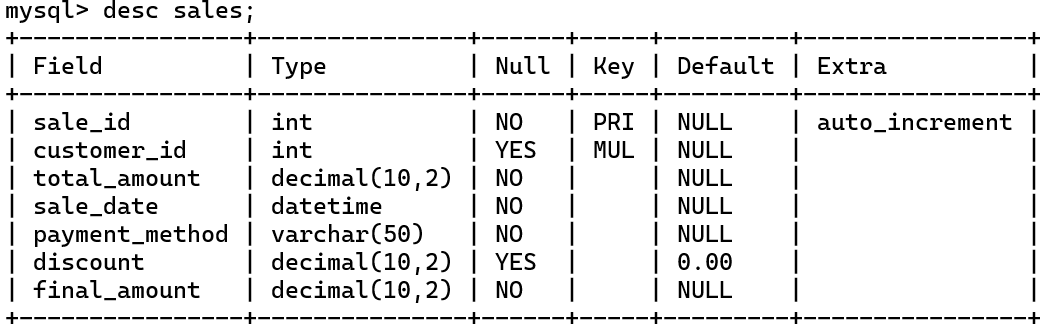
desc PurchaseDetails ;

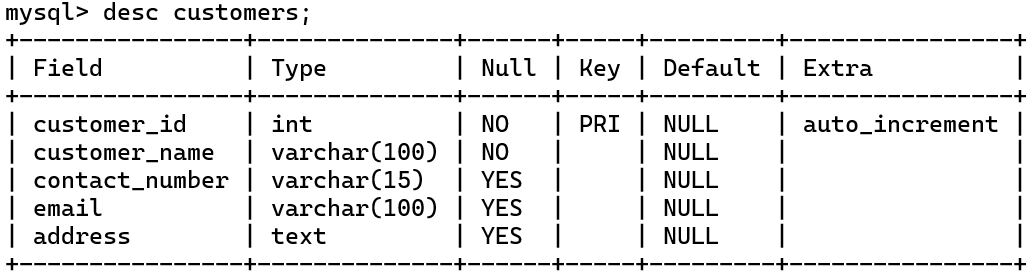
desc Purchase ;

desc Inventory ;



desc SalesDetails ;

desc Sales ;

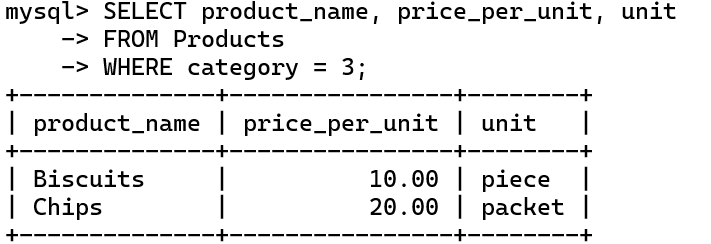
desc Customers ;

\*\*\* Questions \*\*\*

-- Fetch all products in the "Snacks" category.

SELECT product\_name, price\_per\_unit, unit

FROM Products

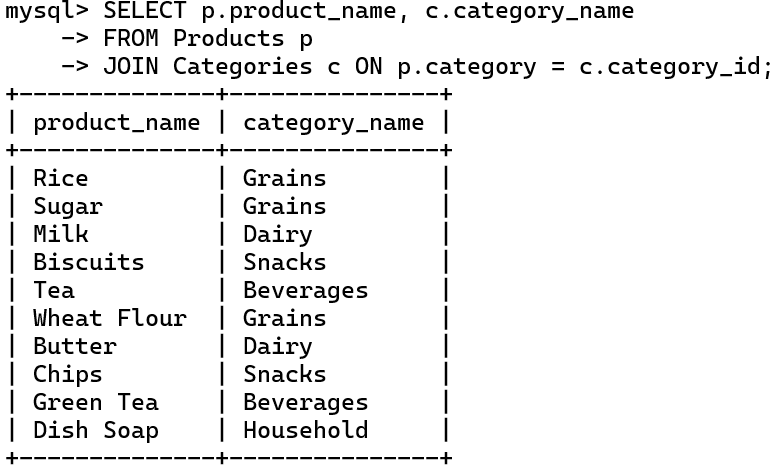
WHERE category = 3;

-- Retrieve all products and their categories

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

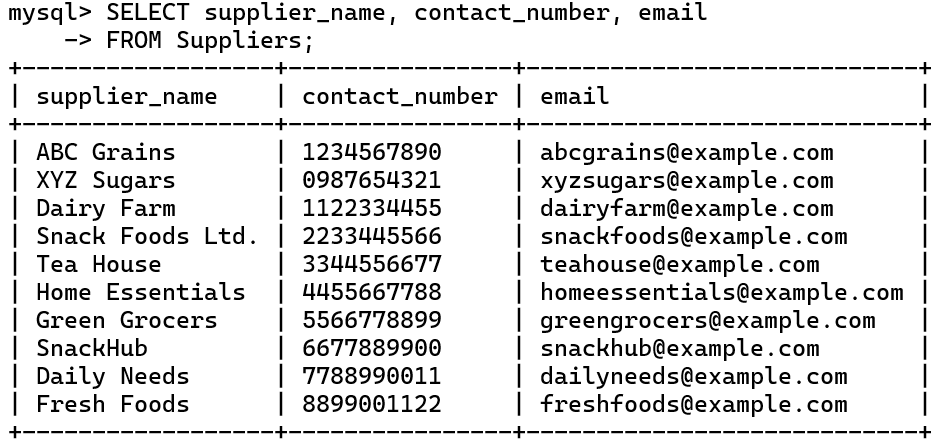
FROM Products p

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



-- List all suppliers with their contact information

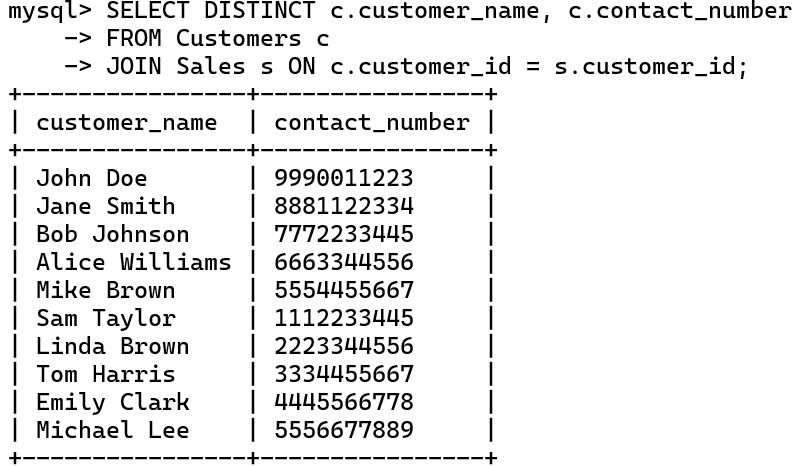
SELECT supplier\_name, contact\_number, email

FROM Suppliers;

-- Get all customers who have made purchases

SELECT DISTINCT c.customer\_name, c.contact\_number

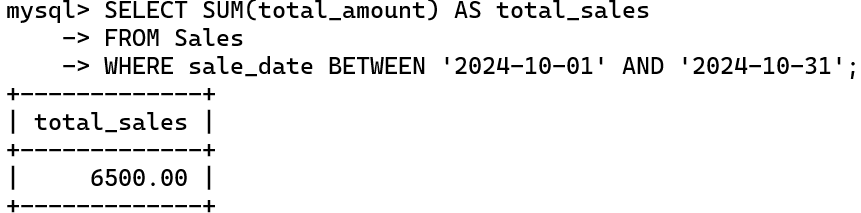
FROM Customers c

JOIN Sales s ON c.customer\_id = s.customer\_id;

-- Find total sales made in a specific month (October 2024)

SELECT SUM(total\_amount) AS total\_sales

FROM Sales

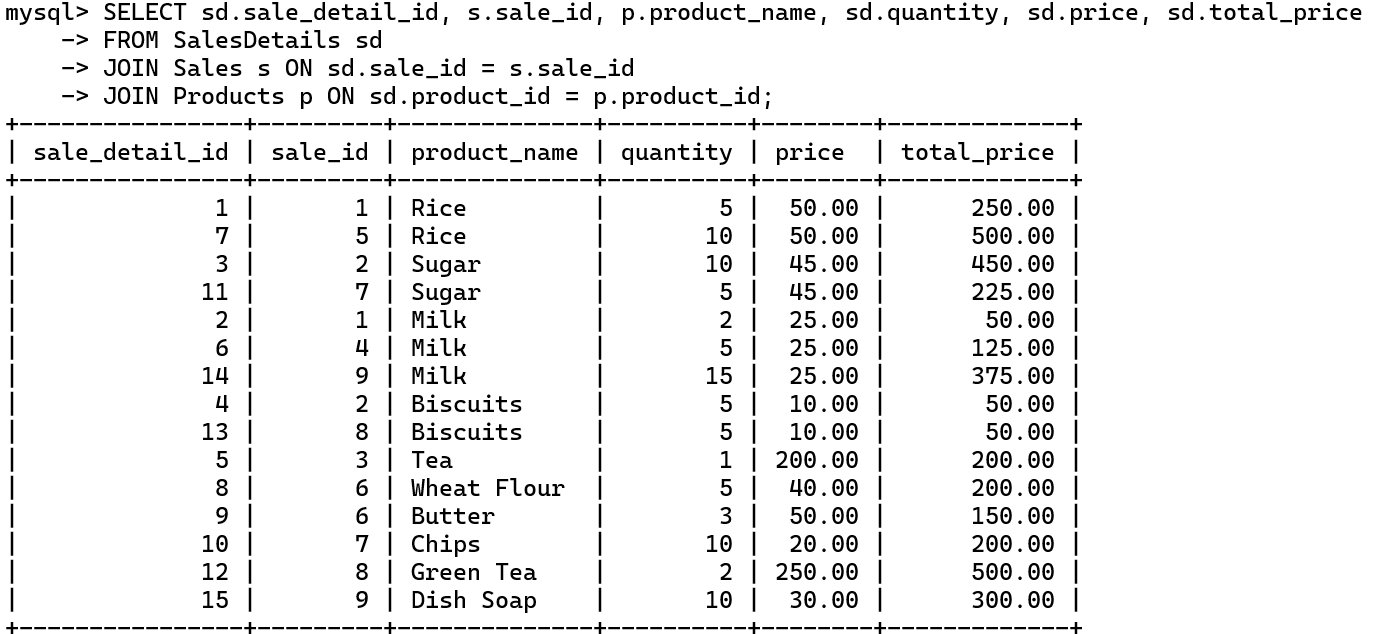
WHERE sale\_date BETWEEN '2024-10-01' AND '2024-10-31';

-- Show all sales details along with the product names

SELECT sd.sale\_detail\_id, s.sale\_id, p.product\_name, sd.quantity, sd.price, sd.total\_price

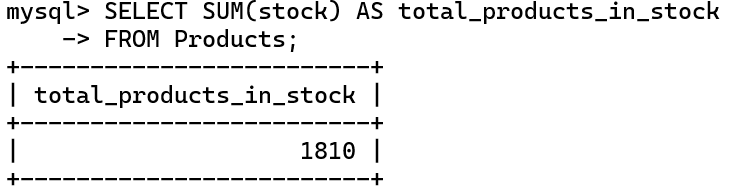
FROM SalesDetails sd

JOIN Sales s ON sd.sale\_id = s.sale\_id

JOIN Products p ON sd.product\_id = p.product\_id;

-- Count the number of products in stock

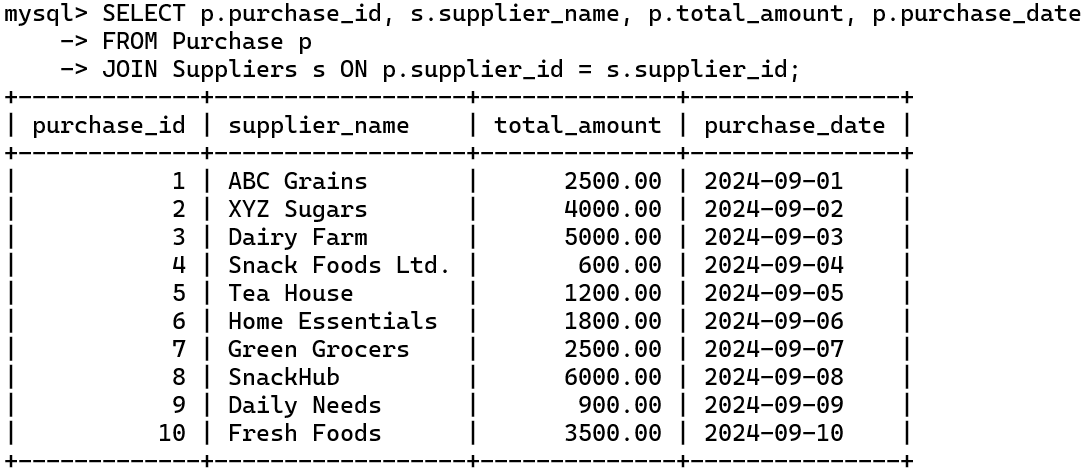
SELECT SUM(stock) AS total\_products\_in\_stock

FROM Products;

-- List all purchases along with supplier names

SELECT p.purchase\_id, s.supplier\_name, p.total\_amount, p.purchase\_date

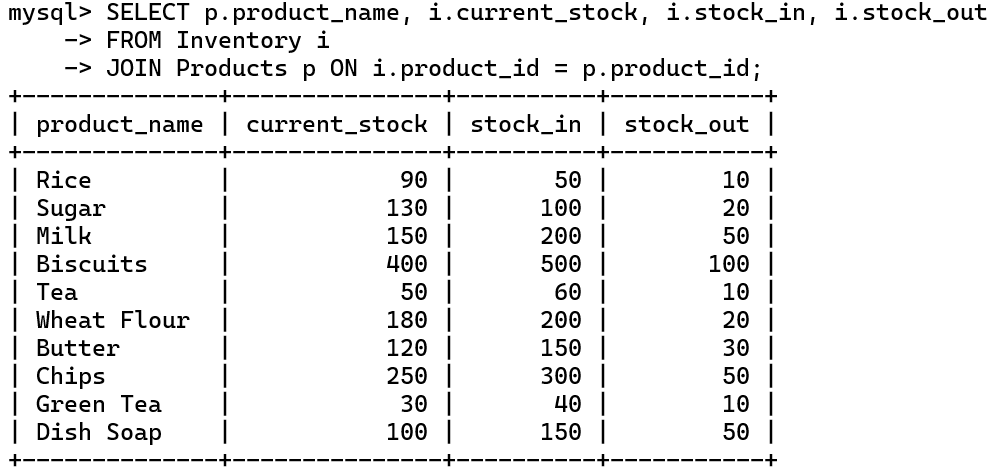
FROM Purchase p

JOIN Suppliers s ON p.supplier\_id = s.supplier\_id;

-- Retrieve inventory details for all products

SELECT p.product\_name, i.current\_stock, i.stock\_in, i.stock\_out

FROM Inventory i

JOIN Products p ON i.product\_id = p.product\_id;

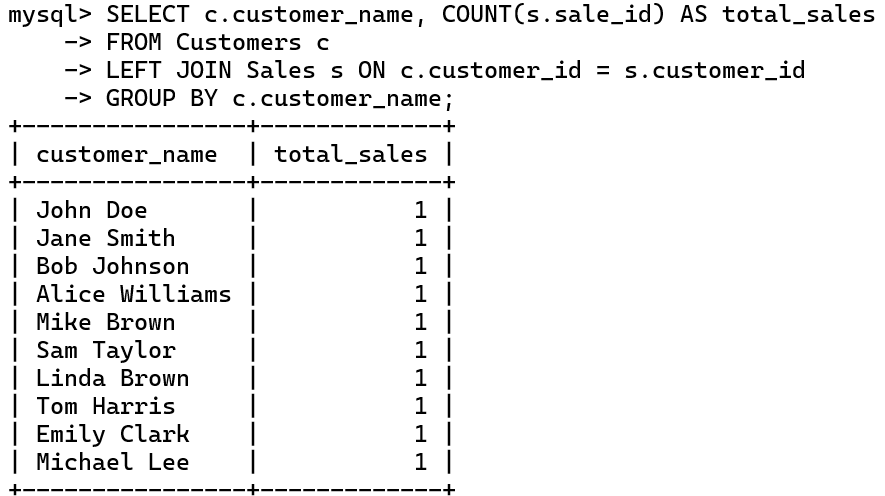
-- Get the total number of sales for each customer

SELECT c.customer\_name, COUNT(s.sale\_id) AS total\_sales

FROM Customers c

LEFT JOIN Sales s ON c.customer\_id = s.customer\_id

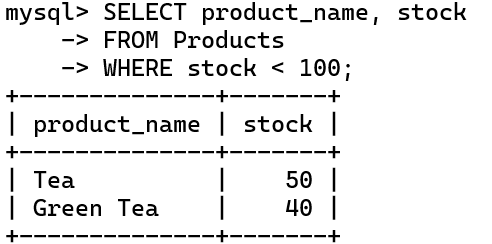
GROUP BY c.customer\_name;



-- Find products that have low stock (less than 100 units)

SELECT product\_name, stock

FROM Products

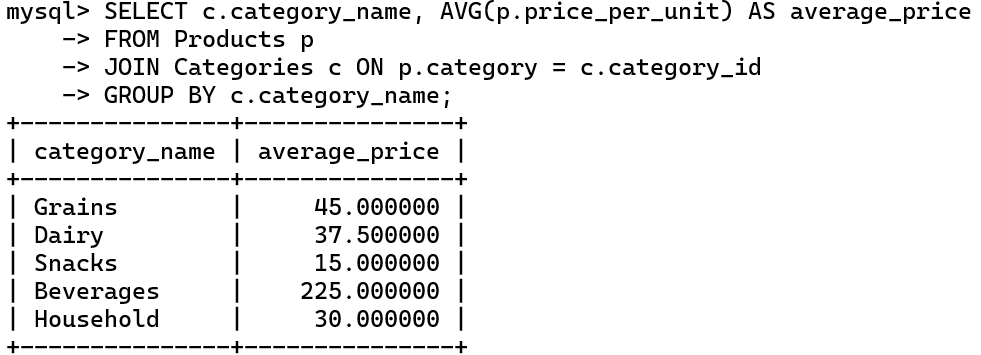
WHERE stock < 100;

-- Calculate the average price of products per category

SELECT c.category\_name, AVG(p.price\_per\_unit) AS average\_price

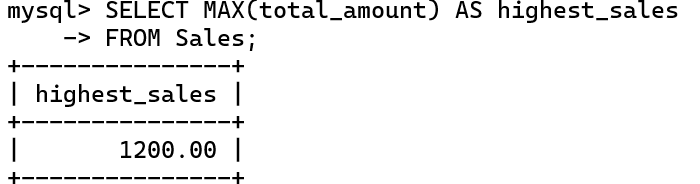
FROM Products p

JOIN Categories c ON p.category = c.category\_id

GROUP BY c.category\_name;

-- Find the highest sales amount:

SELECT MAX(total\_amount) AS highest\_sales

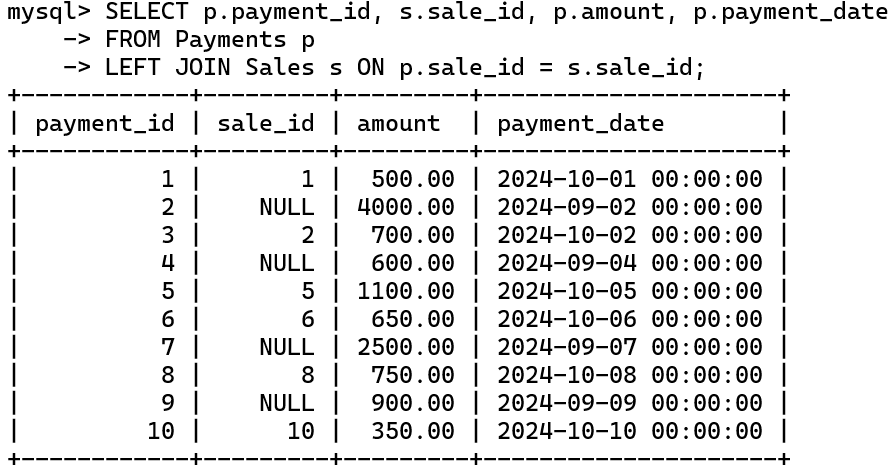
FROM Sales;

-- Show all payments made for sales

SELECT p.payment\_id, s.sale\_id, p.amount, p.payment\_date

FROM Payments p

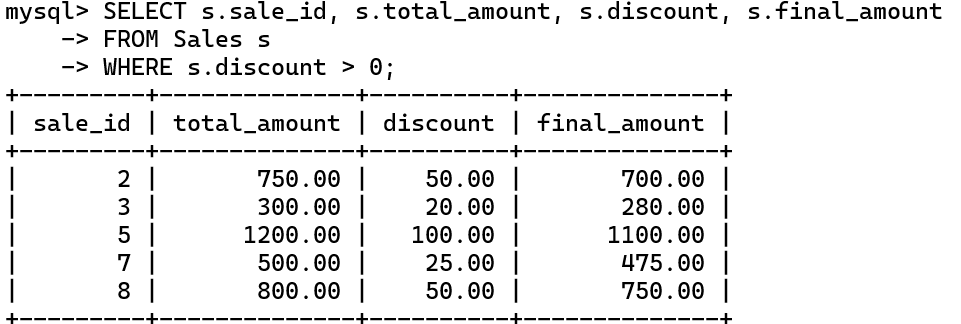
LEFT JOIN Sales s ON p.sale\_id = s.sale\_id;



-- Retrieve all sales with applied discounts

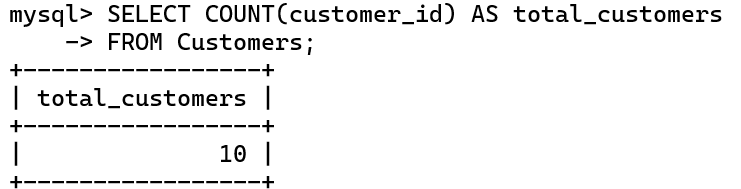
SELECT s.sale\_id, s.total\_amount, s.discount, s.final\_amount

FROM Sales s

WHERE s.discount > 0;

-- Find the total number of customers

SELECT COUNT(customer\_id) AS total\_customers

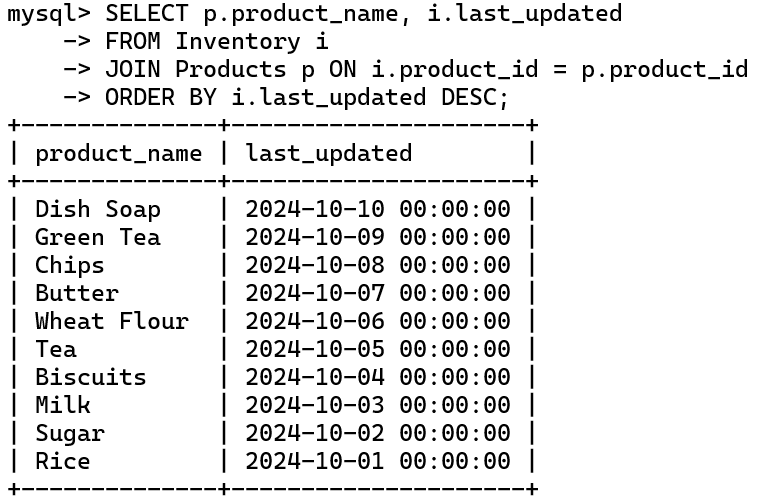
FROM Customers;

-- Get the latest inventory updates for each product

SELECT p.product\_name, i.last\_updated

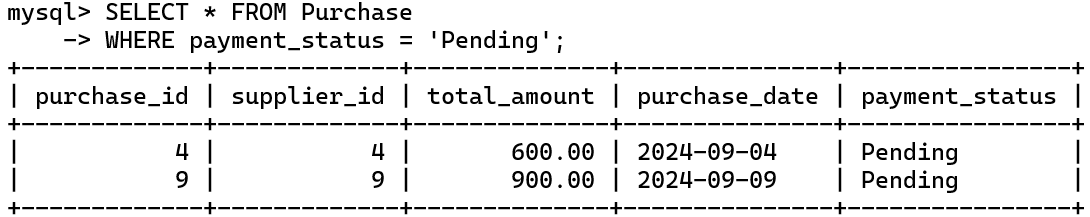
FROM Inventory i

JOIN Products p ON i.product\_id = p.product\_id

ORDER BY i.last\_updated DESC;

-- List all purchases that are pending payment:

SELECT \* FROM Purchase

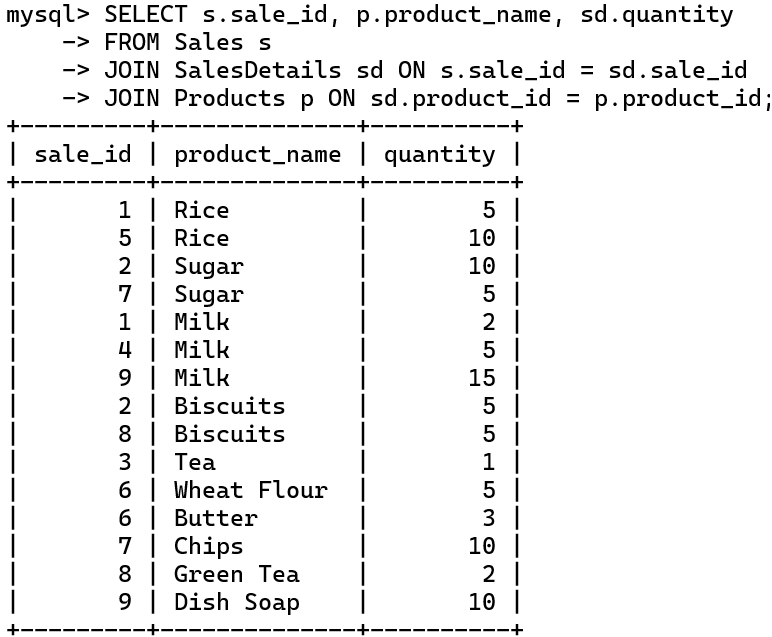
WHERE payment\_status = 'Pending';

-- Get the details of sales, including product name and quantity sold:

SELECT s.sale\_id, p.product\_name, sd.quantity

FROM Sales s

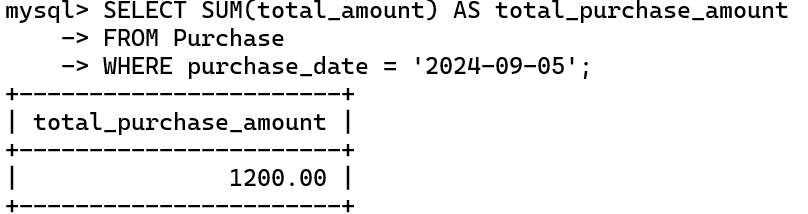
JOIN SalesDetails sd ON s.sale\_id = sd.sale\_id

JOIN Products p ON sd.product\_id = p.product\_id;

-- Calculate the total cost of purchases made on a specific date

SELECT SUM(total\_amount) AS total\_purchase\_amount

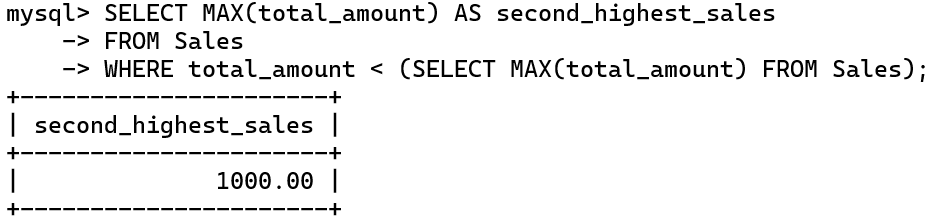
FROM Purchase

WHERE purchase\_date = '2024-09-05';

-- Find the second highest sales amount:

SELECT MAX(total\_amount) AS second\_highest\_sales

FROM Sales

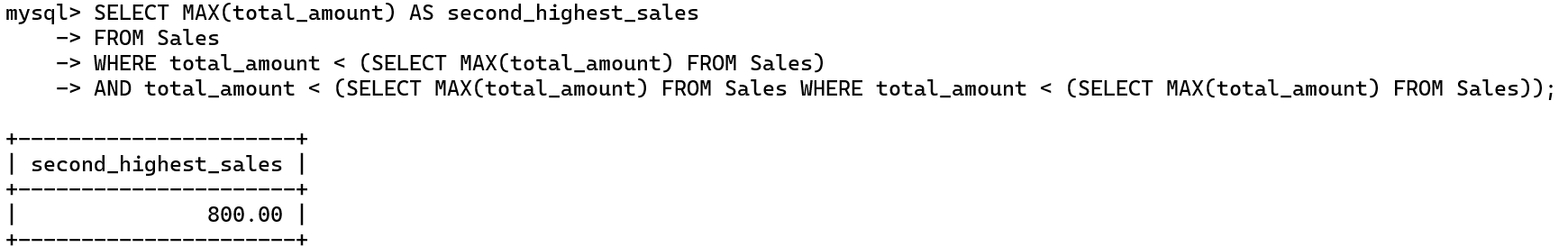
WHERE total\_amount < (SELECT MAX(total\_amount) FROM Sales);

-- Find the 3 highest sales amount:

SELECT MAX(total\_amount) AS second\_highest\_sales

FROM Sales

WHERE total\_amount < (SELECT MAX(total\_amount) FROM Sales)

AND total\_amount < (SELECT MAX(total\_amount) FROM Sales WHERE total\_amount < (SELECT MAX(total\_amount) FROM Sales));

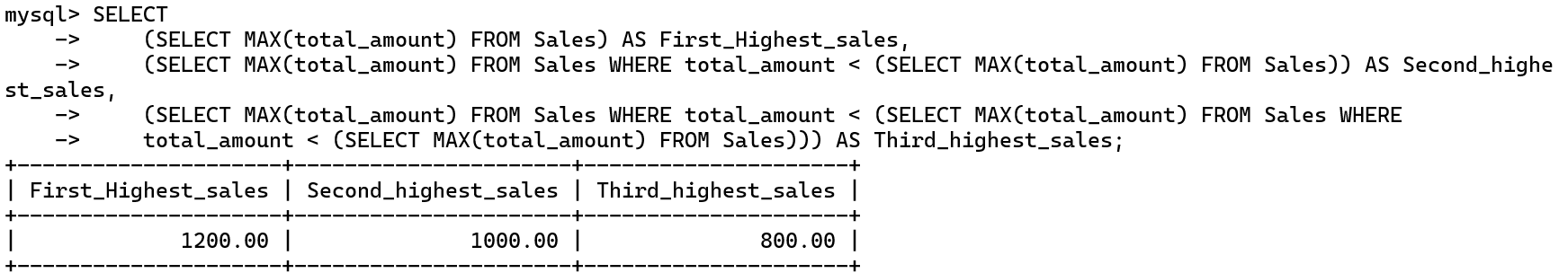
-- Find 1,2,3 highest sales

SELECT

(SELECT MAX(total\_amount) FROM Sales) AS First\_Highest\_sales,

(SELECT MAX(total\_amount) FROM Sales WHERE total\_amount < (SELECT MAX(total\_amount) FROM Sales)) AS Second\_highest\_sales,

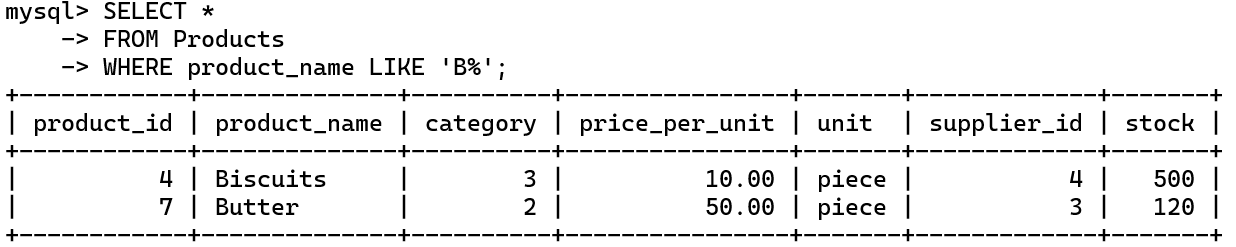
(SELECT MAX(total\_amount) FROM Sales WHERE total\_amount < (SELECT MAX(total\_amount) FROM Sales WHERE

 total\_amount < (SELECT MAX(total\_amount) FROM Sales))) AS Third\_highest\_sales;

-- find all products whose name start with 'b'

SELECT \*

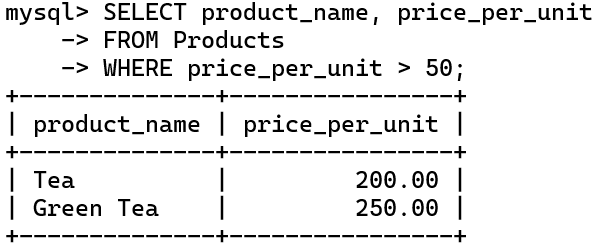
FROM Products

WHERE product\_name LIKE 'B%';

-- find all products with a price per unit greater than 50?

SELECT product\_name, price\_per\_unit

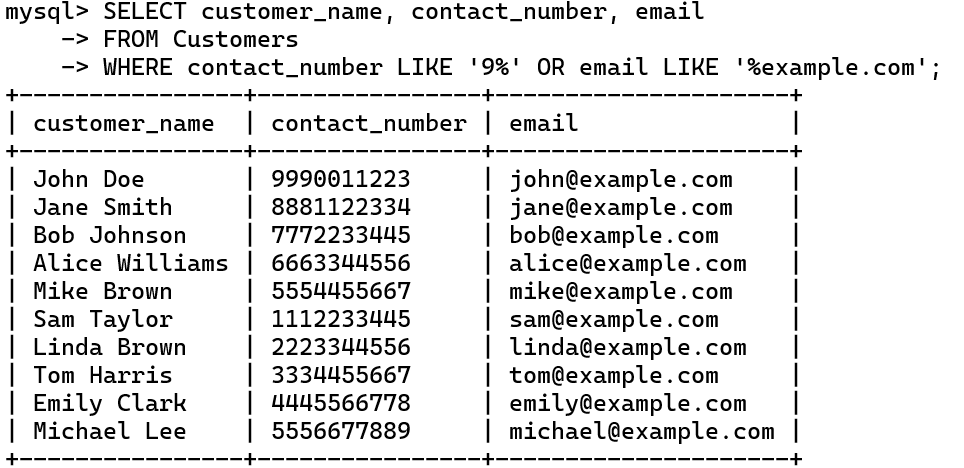
FROM Products

WHERE price\_per\_unit > 50;

-- retrieve all customers who have either a contact number starting with '9' or an email address ending with 'example.com'?

SELECT customer\_name, contact\_number, email

FROM Customers

WHERE contact\_number LIKE '9%' OR email LIKE '%example.com';

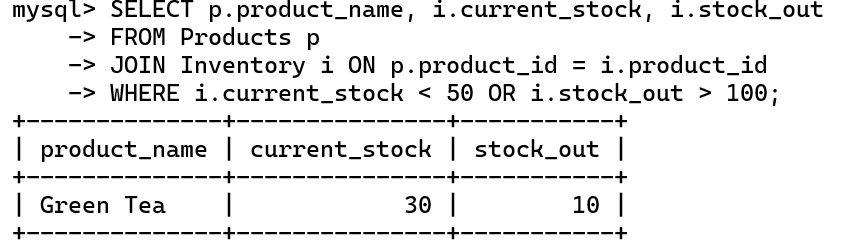
-- find products where the current stock is less than 50 or the stock out exceeds 100

SELECT p.product\_name, i.current\_stock, i.stock\_out

FROM Products p

JOIN Inventory i ON p.product\_id = i.product\_id

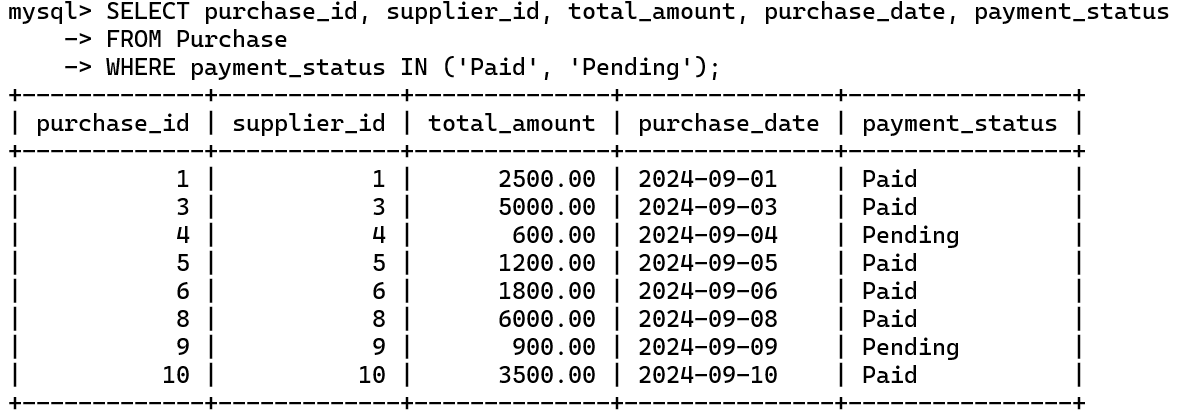
WHERE i.current\_stock < 50 OR i.stock\_out > 100;



-- retrieve all purchases that are either 'Paid' or 'Pending'?

SELECT purchase\_id, supplier\_id, total\_amount, purchase\_date, payment\_status

FROM Purchase

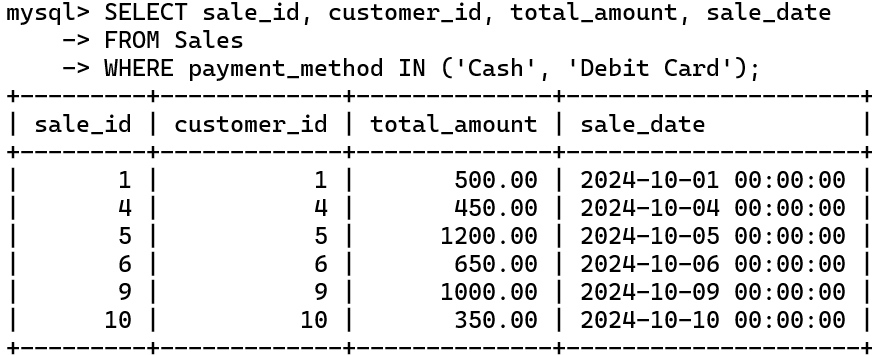
WHERE payment\_status IN ('Paid', 'Pending');

-- to find all sales made using 'Cash' or 'Debit Card'.

SELECT sale\_id, customer\_id, total\_amount, sale\_date

FROM Sales

WHERE payment\_method IN ('Cash', 'Debit Card');

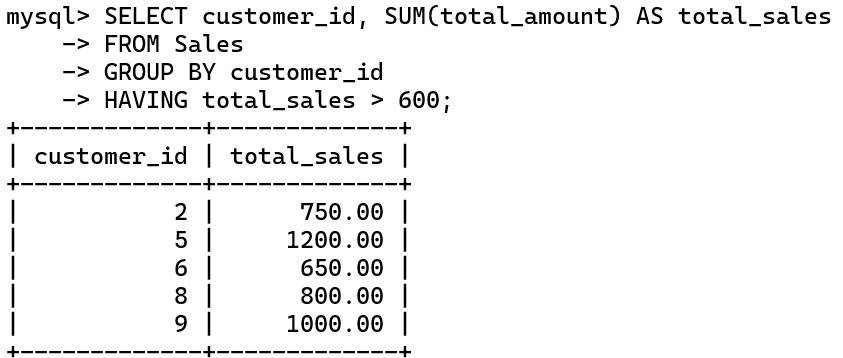


-- find out whose sales is greater than 600 ?

SELECT customer\_id, SUM(total\_amount) AS total\_sales

FROM Sales

GROUP BY customer\_id

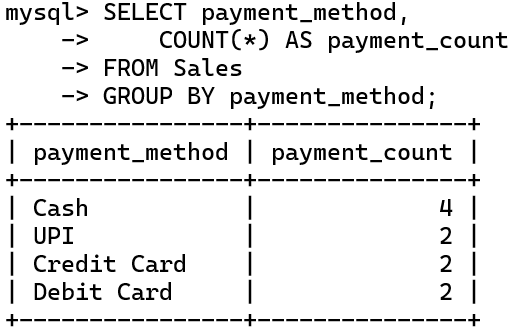
HAVING total\_sales > 600;

-- find the count of each payment method used by customers

SELECT payment\_method,

COUNT(\*) AS payment\_count

FROM Sales

GROUP BY payment\_method;

-- Which Customer has purchase Diary Product ?

SELECT DISTINCT Customers.customer\_name, Products.product\_name

FROM SalesDetails

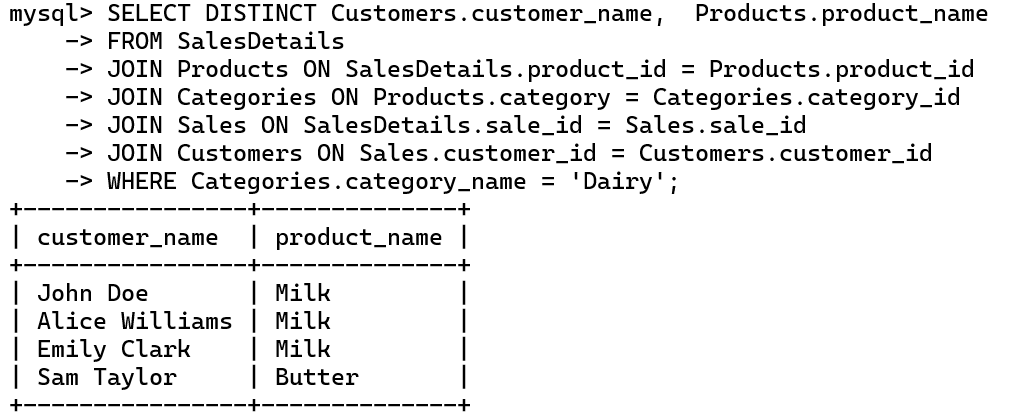
JOIN Products ON SalesDetails.product\_id = Products.product\_id

JOIN Categories ON Products.category = Categories.category\_id

JOIN Sales ON SalesDetails.sale\_id = Sales.sale\_id

JOIN Customers ON Sales.customer\_id = Customers.customer\_id

WHERE Categories.category\_name = 'Dairy';



Conclusion : -

A Kirana Store database is a vital tool for efficiently managing grocery store operations. It allows for better decision-making, helps in maintaining customer satisfaction, and ultimately contributes to the store's profitability.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**\*\*\* THE END \*\*\***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*