**DBMS CIA 3: INDIAN E-COMMERCE**

**Digvijay Purkayastha**

**3 BSC DM**

**2341311**

**DESCRIPTION:**

**Indian e-commerce has rapidly evolved into one of the world's largest and most dynamic online markets. It includes various platforms where consumers and businesses buy and sell goods and services over the Internet. With major players like Flipkart, Amazon India, and Myntra leading the space, the market encompasses various segments such as electronics, apparel, groceries, and digital services.** **E-commerce in India thrives due to the growing internet penetration, widespread use of smartphones, and the convenience of digital payments. The industry also supports large and small sellers by providing them access to a nationwide market. This project models an e-commerce database, showcasing the relationships between entities such as customers, products, orders, payments, and reviews, along with managing various business processes like order fulfillment, product listings, and customer feedback.**

**ER Models:**

**A diagram of a website

Description automatically generated with medium confidence**

**PROGRAM 1: DDL COMMANDS:**

**DDL (Data Definition Language) commands are used to define and manage the structure of a database and its objects, such as tables, indexes, and views. These commands allow users to create, modify, and delete database structures. DDL commands automatically commit the changes once executed, meaning they cannot be rolled back.**

**The main DDL commands are:**

**TRUNCATE: This command is used to remove all records from a table but keep the table structure intact. It is faster than DELETE since it doesn't generate individual row deletion logs.**

**DROP: This command is used to delete database objects such as tables or views. Be careful when using DROP as it permanently removes the object along with all data.**

**ALTER: This command is used to modify the structure of an existing table. You can add, modify, or drop columns.**

**CREATE: This command is used to create new database objects, such as tables, indexes, views, etc.**

**RENAME:** **This command in SQL is used to change the name of an existing database object, such as a table or a column.**

**EXAMPLE:**

-- Creating the Customers table

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100),

ContactNumber VARCHAR(15),

Email VARCHAR(100),

Address VARCHAR(200)

);

-- Creating the Products table

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2),

Stock INT

);

-- Creating the Orders table

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

OrderDate DATE,

CustomerID INT,

ProductID INT,

Quantity INT,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

-- Creating the Sellers table

CREATE TABLE Sellers (

SellerID INT PRIMARY KEY,

SellerName VARCHAR(100),

ContactInfo VARCHAR(100)

);

-- Creating the Payments table

CREATE TABLE Payments (

PaymentID INT PRIMARY KEY,

OrderID INT,

PaymentDate DATE,

Amount DECIMAL(10, 2),

PaymentMethod VARCHAR(50),

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)

);

-- Inserting values into the Customers table

INSERT INTO Customers (CustomerID, CustomerName, ContactNumber, Email, Address)

VALUES

(1, 'John Doe', '9876543210', 'john@example.com', 'Mumbai'),

(2, 'Jane Smith', '9988776655', 'jane@example.com', 'Delhi'),

(3, 'Ravi Kumar', '9123456789', 'ravi@example.com', 'Bangalore');

-- Inserting values into the Products table

INSERT INTO Products (ProductID, ProductName, Category, Price, Stock)

VALUES

(101, 'Smartphone', 'Electronics', 15000.00, 50),

(102, 'Laptop', 'Electronics', 45000.00, 20),

(103, 'Shoes', 'Footwear', 2000.00, 100);

-- Inserting values into the Orders table

INSERT INTO Orders (OrderID, OrderDate, CustomerID, ProductID, Quantity)

VALUES

(1001, '2024-10-15', 1, 101, 2),

(1002, '2024-10-16', 2, 102, 1),

(1003, '2024-10-17', 3, 103, 3);

-- Inserting values into the Sellers table

INSERT INTO Sellers (SellerID, SellerName, ContactInfo)

VALUES

(201, 'TechStore', 'techstore@example.com'),

(202, 'FashionHub', 'fashionhub@example.com');

-- Inserting values into the Payments table

INSERT INTO Payments (PaymentID, OrderID, PaymentDate, Amount, PaymentMethod)

VALUES

(301, 1001, '2024-10-15', 30000.00, 'Credit Card'),

(302, 1002, '2024-10-16', 45000.00, 'UPI'),

(303, 1003, '2024-10-17', 6000.00, 'Debit Card');

ALTER TABLE Customers

ADD DateOfBirth DATE;

RENAME TABLE Customers TO Clients;

DROP TABLE Sellers;

TRUNCATE TABLE Payments;

**OUTPUT:**

**A screenshot of a contact number

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a phone

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**PROGRAM 2 (DML COMMANDS):**

**DML (Data Manipulation Language) commands are used to manipulate the data stored in database tables. Unlike DDL (Data Definition Language), which focuses on the structure of the database, DML commands deal with the actual data within the tables. These commands allow users to insert, update, delete, and retrieve data from a database.**

**The DML Commands are:**

**INSERT: Adds new data (rows) to a table.**

**UPDATE: Modifies existing data in a table.**

**DELETE: Removes data from a table based on specified conditions.**

**SELECT: Retrieves data from one or more tables.**

**EXAMPLE:**

-- Creating the Customers table

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100),

ContactNumber VARCHAR(15),

Email VARCHAR(100),

Address VARCHAR(200)

);

-- Creating the Products table

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2),

Stock INT

);

-- Creating the Orders table

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

OrderDate DATE,

CustomerID INT,

ProductID INT,

Quantity INT,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

-- Inserting values into the Customers table

INSERT INTO Customers (CustomerID, CustomerName, ContactNumber, Email, Address)

VALUES

(1, 'John Doe', '9876543210', 'john@example.com', 'Mumbai'),

(2, 'Jane Smith', '9988776655', 'jane@example.com', 'Delhi'),

(3, 'Ravi Kumar', '9123456789', 'ravi@example.com', 'Bangalore');

-- Inserting values into the Products table

INSERT INTO Products (ProductID, ProductName, Category, Price, Stock)

VALUES

(101, 'Smartphone', 'Electronics', 15000.00, 50),

(102, 'Laptop', 'Electronics', 45000.00, 20),

(103, 'Shoes', 'Footwear', 2000.00, 100);

-- Inserting values into the Orders table

INSERT INTO Orders (OrderID, OrderDate, CustomerID, ProductID, Quantity)

VALUES

(1001, '2024-10-15', 1, 101, 2),

(1002, '2024-10-16', 2, 102, 1),

(1003, '2024-10-17', 3, 103, 3);

-- Update the contact number of a customer

UPDATE Customers

SET ContactNumber = '9123456780'

WHERE CustomerID = 1;

-- Update the stock of a product after an order

UPDATE Products

SET Stock = Stock - 2

WHERE ProductID = 101;

-- Delete a specific order

DELETE FROM Orders

WHERE OrderID = 1003;

-- Delete a product that is no longer available

DELETE FROM Products

WHERE ProductID = 103;

SELECT \* FROM Customers;

OUTPUT:

A screenshot of a phone number

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**PROGRAM 3(CONSTRAINTS):**

**Primary Key**: Ensures each record in a table is unique and cannot be NULL. It uniquely identifies each row.

**Foreign Key**: Establishes a relationship between two tables by referencing the primary key in another table, ensuring referential integrity.

**NULL**: Represents missing or unknown data; allows a column to have no value.

**NOT NULL**: Ensures a column cannot have a NULL value, i.e., it must always contain data.

**CHECK**: Ensures that all values in a column satisfy a specific condition or expression.

**UNIQUE**: Ensures all values in a column (or a group of columns) are distinct across the table.

**AUTO\_INCREMENT**: Automatically generates a unique number for a column, usually for the primary key, when a new record is inserted.

**DEFAULT**: Assigns a default value to a column when no explicit value is provided during data insertion.

**EXAMPLE**:

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY AUTO\_INCREMENT, -- Primary Key with Auto Increment

CustomerName VARCHAR(100) NOT NULL, -- Not Null constraint

ContactNumber VARCHAR(15) UNIQUE NOT NULL, -- Unique and Not Null constraint

Email VARCHAR(100) UNIQUE NOT NULL, -- Unique and Not Null constraint

Address VARCHAR(200),

DateOfBirth DATE CHECK (DateOfBirth <= CURDATE()) -- Check constraint to ensure DateOfBirth is not in the future

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY AUTO\_INCREMENT, -- Primary Key with Auto Increment

ProductName VARCHAR(100) NOT NULL, -- Not Null constraint

Category VARCHAR(50) NOT NULL, -- Not Null constraint

Price DECIMAL(10, 2) NOT NULL CHECK (Price >= 0), -- Check constraint to ensure price is non-negative

Stock INT NOT NULL DEFAULT 0 -- Not Null with Default value

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY AUTO\_INCREMENT, -- Primary Key with Auto Increment

OrderDate DATE NOT NULL, -- Not Null constraint

CustomerID INT NOT NULL, -- Not Null constraint

ProductID INT NOT NULL, -- Not Null constraint

Quantity INT NOT NULL CHECK (Quantity > 0), -- Check constraint to ensure quantity is positive

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID), -- Foreign Key constraint

FOREIGN KEY (ProductID) REFERENCES Products(ProductID) -- Foreign Key constraint

);

CREATE TABLE Payments (

PaymentID INT PRIMARY KEY AUTO\_INCREMENT, -- Primary Key with Auto Increment

OrderID INT NOT NULL, -- Not Null constraint

PaymentDate DATE NOT NULL, -- Not Null constraint

Amount DECIMAL(10, 2) NOT NULL CHECK (Amount > 0), -- Check constraint to ensure amount is positive

PaymentMethod VARCHAR(50) NOT NULL, -- Not Null constraint

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID) -- Foreign Key constraint

);

INSERT INTO Customers (CustomerName, ContactNumber, Email, Address, DateOfBirth)

VALUES

('John Doe', '9876543210', 'john@example.com', 'Mumbai', '1990-01-15'),

('Jane Smith', '9988776655', 'jane@example.com', 'Delhi', '1985-05-25'),

('Ravi Kumar', '9123456789', 'ravi@example.com', 'Bangalore', '1992-03-10');

INSERT INTO Products (ProductName, Category, Price, Stock)

VALUES

('Smartphone', 'Electronics', 15000.00, 50),

('Laptop', 'Electronics', 45000.00, 20),

('Shoes', 'Footwear', 2000.00, 100);

INSERT INTO Orders (OrderDate, CustomerID, ProductID, Quantity)

VALUES

('2024-10-15', 1, 1, 2),

('2024-10-16', 2, 2, 1),

('2024-10-17', 3, 3, 3);

INSERT INTO Payments (OrderID, PaymentDate, Amount, PaymentMethod)

VALUES

(1, '2024-10-15', 30000.00, 'Credit Card'),

(2, '2024-10-16', 45000.00, 'Debit Card'),

(3, '2024-10-17', 6000.00, 'Net Banking');

SELECT \* FROM Customers;

SELECT \* FROM Products;

SELECT \* FROM Orders;

SELECT \* FROM Payments;

**OUTPUT:**

A screenshot of a phone number

Description automatically generated

A screenshot of a phone

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**PROGRAM 4(CLAUSE):**

**WHERE Clause: Filters records based on specified conditions, allowing users to retrieve only the rows that meet certain criteria.**

**GROUP BY Clause: Groups of rows that have the same values in specified columns are divided into summary rows, often used with aggregate functions to perform calculations on each group.**

**HAVING Clause: Similar to the WHERE clause but is used to filter records after aggregation has taken place, applying conditions to grouped data.**

**ORDER BY Clause: Sorts the result set of a query by one or more columns, allowing users to specify the order (ascending or descending) of the retrieved rows.**

**DISTINCT Keyword: Removes duplicate values from the result set, ensuring that each returned row is unique.**

**LIMIT Keyword: Restricts the number of rows returned by a query, allowing users to specify how many records to display.**

**Aggregate Functions: Functions like COUNT, SUM, AVG, MIN, and MAX that perform calculations on a set of values and return a single value, often used in conjunction with GROUP BY.**

**Pattern Matching using LIKE: Allows users to search for a specified pattern in a column using wildcard characters, such as % (any sequence of characters) and \_ (a single character), making it useful for flexible string matching.**

**EXAMPLE:**

CREATE TABLE Customers (

CustomerID INT AUTO\_INCREMENT PRIMARY KEY,

CustomerName VARCHAR(100) NOT NULL,

ContactNumber VARCHAR(15) UNIQUE NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

Address VARCHAR(255),

DateOfBirth DATE CHECK (DateOfBirth < CURRENT\_DATE)

);

CREATE TABLE Products (

ProductID INT AUTO\_INCREMENT PRIMARY KEY,

ProductName VARCHAR(100) NOT NULL,

Category VARCHAR(50) NOT NULL,

Price DECIMAL(10, 2) CHECK (Price >= 0),

Stock INT DEFAULT 0 CHECK (Stock >= 0)

);

CREATE TABLE Orders (

OrderID INT AUTO\_INCREMENT PRIMARY KEY,

OrderDate DATE NOT NULL,

CustomerID INT,

ProductID INT,

Quantity INT CHECK (Quantity > 0),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

CREATE TABLE Payments (

PaymentID INT AUTO\_INCREMENT PRIMARY KEY,

OrderID INT,

PaymentDate DATE NOT NULL,

Amount DECIMAL(10, 2) CHECK (Amount >= 0),

PaymentMethod VARCHAR(50),

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)

);

INSERT INTO Customers (CustomerName, ContactNumber, Email, Address, DateOfBirth)

VALUES

('John Doe', '9876543210', 'john@example.com', 'Mumbai', '1990-01-15'),

('Jane Smith', '9988776655', 'jane@example.com', 'Delhi', '1985-05-25'),

('Ravi Kumar', '9123456789', 'ravi@example.com', 'Bangalore', '1992-03-10'),

('Alice Williams', '9123456780', 'alice@example.com', 'Pune', '1995-07-20');

INSERT INTO Products (ProductName, Category, Price, Stock)

VALUES

('Smartphone', 'Electronics', 15000.00, 50),

('Laptop', 'Electronics', 45000.00, 20),

('Shoes', 'Footwear', 2000.00, 100),

('Washing Machine', 'Appliances', 25000.00, 30),

('Refrigerator', 'Appliances', 30000.00, 25);

INSERT INTO Orders (OrderDate, CustomerID, ProductID, Quantity)

VALUES

('2024-10-15', 1, 1, 2),

('2024-10-16', 2, 2, 1),

('2024-10-17', 3, 3, 3),

('2024-10-17', 4, 4, 1),

('2024-10-18', 1, 5, 1);

INSERT INTO Payments (OrderID, PaymentDate, Amount, PaymentMethod)

VALUES

(1, '2024-10-15', 30000.00, 'Credit Card'),

(2, '2024-10-16', 45000.00, 'Debit Card'),

(3, '2024-10-17', 6000.00, 'Net Banking'),

(4, '2024-10-17', 25000.00, 'Credit Card'),

(5, '2024-10-18', 30000.00, 'Debit Card');

SELECT \* FROM Orders

WHERE CustomerID = (SELECT CustomerID FROM Customers WHERE Address = 'Delhi');

SELECT CustomerID, COUNT(OrderID) AS TotalOrders

FROM Orders

GROUP BY CustomerID;

SELECT CustomerID, COUNT(OrderID) AS TotalOrders

FROM Orders

GROUP BY CustomerID

HAVING COUNT(OrderID) > 1;

SELECT ProductID, ProductName, Price

FROM Products

ORDER BY Price DESC;

SELECT DISTINCT PaymentMethod

FROM Payments;

SELECT \*

FROM Products

ORDER BY Price DESC

LIMIT 3;

SELECT SUM(Amount) AS TotalRevenue

FROM Payments;

SELECT \*

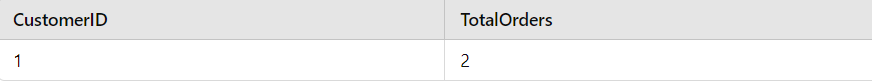
FROM Customers

WHERE CustomerName LIKE 'J%';

**OUTPUT:**

**A screenshot of a number

Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**

**A screenshot of a phone

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

****

**A screenshot of a phone

Description automatically generated**

**PROGRAM 5 (JOINS):**

**Joins in SQL are used to combine rows from two or more tables based on a related column between them. They allow you to retrieve data from multiple tables in a single query.**

**The various kinds of join are:**

**INNER JOIN: Returns records that have matching values in both tables**

**LEFT JOIN (or LEFT OUTER JOIN): Returns all records from the left table and the matched records from the right table. If there is no match, NULL values are returned for columns from the right table.**

**RIGHT JOIN (or RIGHT OUTER JOIN): Returns all records from the right table and the matched records from the left table. If there is no match, NULL values are returned for columns from the left table.**

**FULL OUTER JOIN: Returns all records when a match is in the left or right table. If there is no match, NULL values are returned for non-matching rows from both tables.**

**EXAMPLE:**

CREATE TABLE Customers (

CustomerID INT AUTO\_INCREMENT PRIMARY KEY,

CustomerName VARCHAR(100) NOT NULL,

ContactNumber VARCHAR(15) UNIQUE NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL

);

CREATE TABLE Products (

ProductID INT AUTO\_INCREMENT PRIMARY KEY,

ProductName VARCHAR(100) NOT NULL,

Price DECIMAL(10, 2) CHECK (Price >= 0)

);

CREATE TABLE Orders (

OrderID INT AUTO\_INCREMENT PRIMARY KEY,

OrderDate DATE NOT NULL,

CustomerID INT,

ProductID INT,

Quantity INT CHECK (Quantity > 0),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

INSERT INTO Customers (CustomerName, ContactNumber, Email)

VALUES

('John Doe', '9876543210', 'john@example.com'),

('Jane Smith', '9988776655', 'jane@example.com'),

('Ravi Kumar', '9123456789', 'ravi@example.com');

INSERT INTO Products (ProductName, Price)

VALUES

('Smartphone', 15000.00),

('Laptop', 45000.00),

('Shoes', 2000.00);

INSERT INTO Orders (OrderDate, CustomerID, ProductID, Quantity)

VALUES

('2024-10-15', 1, 1, 2), -- John Doe ordered 2 Smartphones

('2024-10-16', 2, 2, 1), -- Jane Smith ordered 1 Laptop

('2024-10-17', 1, 3, 1); -- John Doe ordered 1 pair of Shoes

SELECT

o.OrderID,

c.CustomerName,

p.ProductName,

o.Quantity

FROM

Orders o

INNER JOIN

Customers c ON o.CustomerID = c.CustomerID

INNER JOIN

Products p ON o.ProductID = p.ProductID;

SELECT

c.CustomerName,

o.OrderID,

p.ProductName,

o.Quantity

FROM

Customers c

LEFT JOIN

Orders o ON c.CustomerID = o.CustomerID

LEFT JOIN

Products p ON o.ProductID = p.ProductID;

SELECT

o.OrderID,

c.CustomerName,

p.ProductName,

o.Quantity

FROM

Orders o

RIGHT JOIN

Customers c ON o.CustomerID = c.CustomerID

RIGHT JOIN

Products p ON o.ProductID = p.ProductID;

SELECT

c.CustomerName,

o.OrderID,

p.ProductName,

o.Quantity

FROM

Customers c

LEFT JOIN

Orders o ON c.CustomerID = o.CustomerID

LEFT JOIN

Products p ON o.ProductID = p.ProductID

UNION

SELECT

c.CustomerName,

o.OrderID,

p.ProductName,

o.Quantity

FROM

Orders o

RIGHT JOIN

Customers c ON o.CustomerID = c.CustomerID

RIGHT JOIN

Products p ON o.ProductID = p.ProductID;

**OUTPUT:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**PROGRAM 6(FUNCTIONS):**

**String Functions:**

**String functions operate on text data (i.e., strings). These functions are useful for manipulating and extracting information from strings.**

**CONCAT(): Combines two or more strings into one.**

**LENGTH(): Returns the number of characters in a string.**

**UPPER() / LOWER(): Converts a string to uppercase or lowercase.**

**Numeric Functions**

**Numeric functions are used to perform operations on numeric data, such as calculations or rounding**

**CEIL() / FLOOR(): Rounds a number up to the next integer (CEIL) or down to the previous integer (FLOOR).**

**POWER(): Returns a number raised to a specified power.**

**SQRT(): Returns the square root of a number.**

**ROUND(): Rounds a number to a specified number of decimal places.**

**Date-Time Functions**

**Date-time functions help you work with dates and times. We can extract information, manipulate dates, or format them as needed.**

**DATEDIFF (): Returns the difference between two dates (in days).**

**DAY () / MONTH() / YEAR(): Extracts the day, month, or year from a date.**

**NOW(): Returns the current date and time.**

**EXAMPLE:**

CREATE TABLE Customers (

CustomerID INT AUTO\_INCREMENT PRIMARY KEY,

CustomerName VARCHAR(100),

ContactNumber VARCHAR(15),

Email VARCHAR(100)

);

CREATE TABLE Products (

ProductID INT AUTO\_INCREMENT PRIMARY KEY,

ProductName VARCHAR(100),

Price DECIMAL(10, 2)

);

CREATE TABLE Orders (

OrderID INT AUTO\_INCREMENT PRIMARY KEY,

OrderDate DATE,

CustomerID INT,

ProductID INT,

Quantity INT

);

INSERT INTO Customers (CustomerName, ContactNumber, Email)

VALUES ('John Doe', '9876543210', 'john@example.com'),

('Jane Smith', '9988776655', 'jane@example.com'),

('Ravi Kumar', '9123456789', 'ravi@example.com');

INSERT INTO Products (ProductName, Price)

VALUES ('Smartphone', 15000.00),

('Laptop', 45000.00),

('Shoes', 2000.00);

INSERT INTO Orders (OrderDate, CustomerID, ProductID, Quantity)

VALUES ('2024-10-16', 1, 1, 2),

('2024-10-16', 2, 2, 1),

('2024-10-17', 3, 3, 3);

SELECT CONCAT(CustomerName, ' <', Email, '>') AS ContactInfo

FROM Customers;

SELECT ProductName, Price, CEIL(Price) AS RoundedUp, FLOOR(Price) AS RoundedDown

FROM Products;

SELECT ProductName, LENGTH(ProductName) AS NameLength

FROM Products;

SELECT CustomerName, UPPER(CustomerName) AS UpperCaseName, LOWER(CustomerName) AS LowerCaseName

FROM Customers;

SELECT OrderID, DATEDIFF(NOW(), OrderDate) AS DaysSinceOrder

FROM Orders;

SELECT NOW() AS CurrentDateTime;

SELECT OrderID, Quantity, SQRT(Quantity) AS QuantityRoot

FROM Orders;

SELECT ProductName, ROUND(Price, -3) AS RoundedPrice

FROM Products;

**OUTPUT:**

A white rectangular object with a gray background

Description automatically generated

A screenshot of a computer

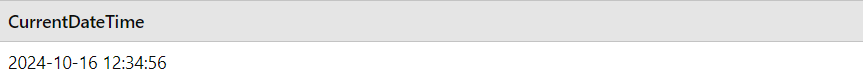
Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a phone

Description automatically generated



A screenshot of a computer

Description automatically generated



**PROGRAM 7(Nested Queries/Subqueries):**

**Self-Query:**

**A query where a table references itself, often using joins to compare rows within the same table. It is commonly used for hierarchical or recursive relationships (e.g., employees and managers).**

**Subquery:**

**A query embedded inside another SQL query is used to retrieve data for the outer query. Subqueries can be placed in SELECT, WHERE, or FROM clauses.**

**Relational Operator:**

**Operators like =,=, <, >, <=, >= are used to compare two values in SQL, returning results based on the comparison (e.g., Price > 100).**

**Comparison Operator:**

**A set of operators used to compare two expressions or values (e.g., = for equality,= for inequality) to filter or match data.**

**Membership Operator (ANY and ALL):**

**ANY compares a value to any value in a subquery's result set, while ALL compares a value to all values in a subquery.**

**SET OPERATION:**

**INTERSECT :**

**The INTERSECT operation returns the common rows between two queries. Instead of using INTERSECT, we can use a self-join to achieve the same result.**

**UNION:**

**The UNION operation returns distinct rows from two queries.**

**EXAMPLE:**

CREATE TABLE Employees (

EmployeeID INT AUTO\_INCREMENT PRIMARY KEY,

EmployeeName VARCHAR(100),

ManagerID INT

);

INSERT INTO Employees (EmployeeName, ManagerID)

VALUES ('John Doe', NULL), -- John is the CEO with no manager

('Jane Smith', 1), -- Jane reports to John

('Ravi Kumar', 1), -- Ravi reports to John

('Sara Lee', 2); -- Sara reports to Jane

CREATE TABLE Customers (

CustomerID INT AUTO\_INCREMENT PRIMARY KEY,

CustomerName VARCHAR(100) NOT NULL

);

INSERT INTO Customers (CustomerName)

VALUES ('John Doe'),

('Jane Smith'),

('Ravi Kumar');

CREATE TABLE Products (

ProductID INT AUTO\_INCREMENT PRIMARY KEY,

ProductName VARCHAR(100) NOT NULL,

Price DECIMAL(10, 2) NOT NULL

);

INSERT INTO Products (ProductName, Price)

VALUES ('Smartphone', 15000),

('Laptop', 45000),

('Shoes', 5000);

CREATE TABLE Orders (

OrderID INT AUTO\_INCREMENT PRIMARY KEY,

CustomerID INT NOT NULL,

ProductID INT NOT NULL,

OrderDate DATE,

Quantity INT,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

INSERT INTO Orders (CustomerID, ProductID, OrderDate, Quantity)

VALUES (1, 1, '2024-10-15', 2), -- John Doe ordered 2 Smartphones

(2, 2, '2024-10-15', 1), -- Jane Smith ordered 1 Laptop

(1, 2, '2024-10-16', 1), -- John Doe ordered 1 Laptop

(3, 3, '2024-10-16', 3); -- Ravi Kumar ordered 3 pairs of Shoes

SELECT E1.EmployeeName AS Employee, E2.EmployeeName AS Manager

FROM Employees E1

JOIN Employees E2

ON E1.ManagerID = E2.EmployeeID

WHERE E2.EmployeeName = 'Jane Smith';

SELECT CustomerName

FROM Customers

WHERE CustomerID IN (

SELECT O.CustomerID

FROM Orders O

JOIN Products P

ON O.ProductID = P.ProductID

WHERE P.Price > 30000

);

SELECT CustomerName

FROM Customers

WHERE CustomerID IN (

SELECT CustomerID

FROM Orders

GROUP BY CustomerID

HAVING COUNT(OrderID) > 1

);

SELECT ProductName

FROM Products

WHERE Price < 20000

AND ProductID IN (

SELECT ProductID

FROM Orders

GROUP BY ProductID

HAVING COUNT(OrderID) > 1

);

SELECT ProductName

FROM Products

WHERE Price > ANY (

SELECT Price

FROM Products

WHERE Price < 10000

);

SELECT ProductName

FROM Products

WHERE Price > ALL (

SELECT Price

FROM Products

WHERE Price < 10000

);

SELECT ProductID, ProductName, Price

FROM Products

WHERE Price > 10000

UNION

SELECT ProductID, ProductName, Price

FROM Products

WHERE Price < 5000;

SELECT P1.ProductID, P1.ProductName, P1.Price

FROM Products P1

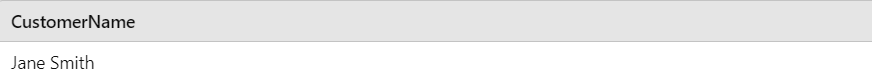
JOIN Products P2

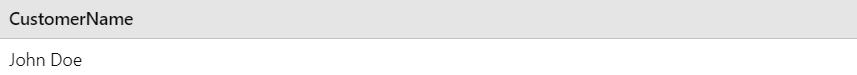
ON P1.ProductID = P2.ProductID

WHERE P1.Price > 10000 AND P2.Price < 50000;

OUTPUT:









A white rectangular object with a black border

Description automatically generated with medium confidence

A white rectangular object with a white stripe

Description automatically generated

A screenshot of a phone

Description automatically generated

A screenshot of a phone

Description automatically generated

**PROGRAM 8 (CARTESIAN PRODUCT, DIVISION AND RENAME):**

**The Cartesian product in SQL occurs when you perform a JOIN without specifying a condition (e.g., using CROSS JOIN), resulting in each row from the first table being paired with every row from the second table. This produces a result set containing all possible combinations of rows from both tables.**

**Division in SQL is used to find rows in one table that are associated with all rows in another table, typically using a combination of JOIN, GROUP BY, and HAVING.**

**Rename in SQL is the process of assigning new names to tables or columns using the AS keyword to make queries more readable or to work with temporary aliases.**

**EXAMPLE:**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100) NOT NULL

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100) NOT NULL,

Price DECIMAL(10, 2)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

ProductID INT,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

INSERT INTO Customers (CustomerID, CustomerName)

VALUES (1, 'John Doe'),

(2, 'Jane Smith');

INSERT INTO Products (ProductID, ProductName, Price)

VALUES (1, 'Smartphone', 15000),

(2, 'Laptop', 45000);

INSERT INTO Orders (OrderID, CustomerID, ProductID)

VALUES (1, 1, 1), -- John Doe ordered a Smartphone

(2, 1, 2), -- John Doe ordered a Laptop

(3, 2, 1); -- Jane Smith ordered a Smartphone

SELECT Customers.CustomerName, Products.ProductName, Products.Price

FROM Customers

CROSS JOIN Products;

SELECT C.CustomerName

FROM Customers C

JOIN Orders O ON C.CustomerID = O.CustomerID

GROUP BY C.CustomerID

HAVING COUNT(DISTINCT O.ProductID) = (SELECT COUNT(\*) FROM Products);

SELECT CustomerName AS Buyer, ProductName AS Item

FROM Customers

JOIN Orders ON Customers.CustomerID = Orders.CustomerID

JOIN Products ON Orders.ProductID = Products.ProductID;

**OUTPUT:**

**A screenshot of a phone

Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**

**PROGRAM 9 (TCL COMMANDS) :**

**TCL (Transaction Control Language) commands in SQL are used to manage transactions in a database. Transactions are logical units of work that either succeed entirely or fail completely, ensuring data integrity. TCL commands help manage these transactions.**

**The TCL commands are:**

**START TRANSACTION:**

Begin a new transaction.

Until the transaction is committed, all changes remain temporary.

**AUTO COMMIT (ON/OFF):**

Controls whether each SQL statement is automatically committed after execution.

ON: Each statement is committed automatically (default behavior).

OFF: Transactions must be explicitly committed using the COMMIT command.

**COMMIT:**

Permanently saves all changes made during the current transaction to the database.

Once committed, the changes cannot be rolled back.

**SAVEPOINT:**

Creates a savepoint within a transaction, allowing partial rollbacks to a specific point without rolling back the entire transaction.

**ROLLBACK**

Reverts the database back to its state at the last COMMIT or to a specific SAVEPOINT, undoing any changes made after that point.

**EXAMPLE:**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100) NOT NULL

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100) NOT NULL,

Price DECIMAL(10, 2)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

ProductID INT,

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

INSERT INTO Customers (CustomerID, CustomerName)

VALUES (1, 'John Doe'),

(2, 'Jane Smith');

INSERT INTO Products (ProductID, ProductName, Price)

VALUES (1, 'Smartphone', 15000),

(2, 'Laptop', 45000);

INSERT INTO Orders (OrderID, CustomerID, ProductID, OrderDate)

VALUES (101, 1, 1, '2024-10-15'),

(102, 2, 2, '2024-10-16');

START TRANSACTION;

SET AUTOCOMMIT = 0;

UPDATE Products

SET Price = 14000

WHERE ProductID = 1;

SELECT \* FROM Products;

SAVEPOINT sp1;

UPDATE Products

SET Price = 40000

WHERE ProductID = 2;

SAVEPOINT sp2;

SELECT \* FROM Products;

ROLLBACK TO sp1;

SELECT \* FROM Products;

COMMIT;

SET AUTOCOMMIT = 1;

**OUTPUT:**

**A screenshot of a phone

Description automatically generated**

**A screenshot of a phone

Description automatically generated**

**A screenshot of a phone

Description automatically generated**

**PROGRAM 10 ( VDL COMMANDS):**

**VDL (View Definition Language) commands in SQL are used to define, manage, and manipulate views in a database. A view is a virtual table that presents data from one or more tables in a simplified or filtered manner without storing the actual data itself. VDL commands allow the creation, updating, and deletion of these views.**

**The VDL Commands are:**

**CREATE VIEW:**

**Used to define a new view in the database.**

**A view can present data from multiple tables and include conditions for filtering data.**

**ALTER VIEW:**

**Used to modify an existing view without dropping it.**

**This command is often used to update the view’s query or structure.**

**DROP VIEW:**

**Used to remove an existing view from the database.**

**After executing this command, the view is deleted, and queries using that view will no longer work.**

**EXAMPLE:**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100) NOT NULL,

Price DECIMAL(10, 2) NOT NULL CHECK (Price > 0)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

ProductID INT,

OrderDate DATE,

Quantity INT CHECK (Quantity > 0),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

INSERT INTO Customers (CustomerID, CustomerName, Email)

VALUES (1, 'John Doe', 'john@example.com'),

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

(3, 'Amit Sharma', 'amit@example.com');

INSERT INTO Products (ProductID, ProductName, Price)

VALUES (1, 'Smartphone', 15000),

(2, 'Laptop', 45000),

(3, 'Tablet', 25000);

INSERT INTO Orders (OrderID, CustomerID, ProductID, OrderDate, Quantity)

VALUES (101, 1, 1, '2024-10-15', 2),

(102, 2, 2, '2024-10-16', 1),

(103, 3, 3, '2024-10-17', 3);

CREATE VIEW AllCustomers AS

SELECT \* FROM Customers;

SELECT \* FROM AllCustomers;

CREATE VIEW ActiveCustomers AS

SELECT CustomerName, Email

FROM Customers

WHERE CustomerID <= 2; -- Assuming customers with ID <= 2 are 'active'

SELECT \* FROM ActiveCustomers;

CREATE VIEW CustomerOrders AS

SELECT c.CustomerName, o.OrderDate, p.ProductName

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

JOIN Products p ON o.ProductID = p.ProductID

WHERE o.Quantity > 1; -- Only include orders with quantity greater than 1

SELECT \* FROM CustomerOrders;

CREATE VIEW ProductOrders AS

SELECT p.ProductName, o.Quantity, o.OrderDate

FROM Products p

JOIN Orders o ON p.ProductID = o.ProductID;

SELECT \* FROM ProductOrders;

UPDATE Orders

SET Quantity = 1

WHERE OrderID = 101; -- Changing John's order quantity from 2 to 1

SELECT \* FROM Orders;

SELECT \* FROM CustomerOrders;

DELETE FROM Orders

WHERE OrderID IN (102, 103); -- Deleting Jane's and Amit's orders

SELECT \* FROM Orders;

SELECT \* FROM ProductOrders;

INSERT INTO ActiveCustomers (CustomerName, Email)

VALUES ('Ravi Kumar', 'ravi@example.com'),

('Priya Singh', 'priya@example.com'),

('Sita Verma', 'sita@example.com'); -- This will violate the UNIQUE constraint on Email

SELECT \* FROM Customers;

SELECT \* FROM ActiveCustomers;

UPDATE Customers

SET Email = CASE

WHEN CustomerName = 'John Doe' THEN 'john\_doe@example.com'

ELSE Email

END;

SELECT \* FROM Customers;

SELECT \* FROM ActiveCustomers;

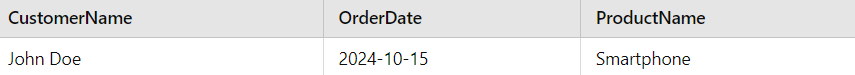
**OUTPUT:**

**A screenshot of a phone

Description automatically generated**

**A screenshot of a email

Description automatically generated**

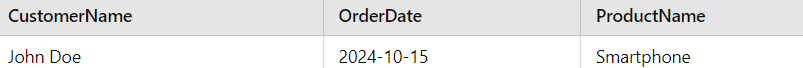
****

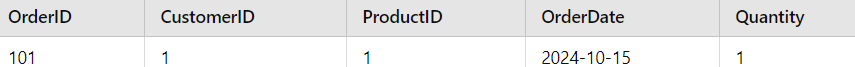
**A screenshot of a computer

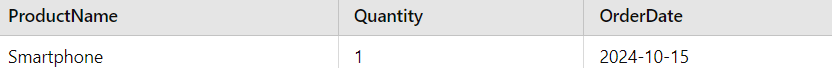
Description automatically generated**

**A screenshot of a computer

Description automatically generated**

****

****

****

**A screenshot of a computer

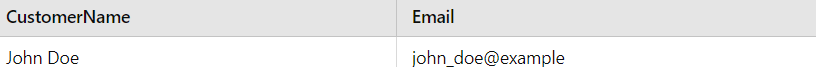
Description automatically generated**

**A screenshot of a email

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

****