**E-commerce management system**

Scope:

The scope of an ecommerce management system project is to create a platform that enables businesses to sell their products or services online. The platform should allow customers to browse through the products, add them to their shopping carts, and make payments online. The project should include website design and development, product management, order management, payment gateway integration, shipping and delivery management, customer relationship management, analytics and reporting, security and compliance, and mobile application development. The goal is to create a robust, user-friendly, and secure platform that can help businesses grow their online sales and revenue.

Motivation:

* Increase in Online Sales: As more and more customers prefer to shop online, businesses need to have a strong online presence to increase their sales. An ecommerce management system can provide businesses with a platform to sell their products or services online, thereby increasing their revenue.
* Better Customer Experience: Online shoppers expect a seamless and hassle-free shopping experience. An ecommerce management system can provide businesses with the tools to create an intuitive and user-friendly platform that meets the needs and expectations of customers.
* Cost Savings: Traditional brick-and-mortar stores require significant investments in rent, inventory management, and staffing. An ecommerce management system can help businesses reduce these costs and improve their bottom line.

Module Description

* Customer Management: This feature enables businesses to manage customer data, track their order history, and send them promotional offers and discounts.
* Customer Management: This feature enables businesses to manage customer data, track their order history, and send them promotional offers and discounts.
* Customer Management: This feature enables businesses to manage customer data, track their order history, and send them promotional offers and discounts.
* Payment Gateway Integration: This feature facilitates secure online payment transactions using various payment methods like credit/debit cards, net banking, e-wallets, etc.

Relation Algebra:

The relation algebra for the e-commerce management system will include the following relations:

* Product (product\_id, name, description, price, image, category\_id)
* Category (category\_id, category\_name, description)
* Inventory (product\_id, quantity)
* Order (order\_id, customer\_id, order\_date, status)
* Order\_Detail (order\_id, product\_id, quantity, price)
* Payment (payment\_id, order\_id, payment\_date, payment\_method, amount)
* Customer (customer\_id, name, email, phone, address)

Selection

Category\_NAME="HealthCare"

Projection

∏ NAME, Email (CUSTOMER)

Union

∏ NAME (Customer) ∪ ∏ Order\_id (Order)

E-R diagram

Diagram, schematic

Description automatically generated

Architecture Diagram

Diagram

Description automatically generated

Implementation

MY SQL QUERIES

CREATE DATABASE ECOMMERCE;

use ECOMMERCE;

CREATE TABLE UserIDandPasswords

(

UserID VARCHAR(6) NOT NULL PRIMARY KEY,

UserPassword VARCHAR(10) NOT NULL,

TypeOfUser VARCHAR(10) NOT NULL

);

CREATE TABLE Customer

(

CustomerID VARCHAR(6) NOT NULL PRIMARY KEY,

CustomerPassword VARCHAR(10) NOT NULL,

Name VARCHAR(30) NOT NULL,

Address VARCHAR(50) NOT NULL,

Pincode VARCHAR(10) NOT NULL,

PhoneNumber VARCHAR(10) NOT NULL,

Email VARCHAR(255)

);

CREATE TABLE Seller

(

SellerID VARCHAR(6) NOT NULL PRIMARY KEY,

SellerPassword VARCHAR(10) NOT NULL,

Name VARCHAR(20) NOT NULL,

Address VARCHAR(50) NOT NULL,

PhoneNumber VARCHAR(10) NOT NULL

);

CREATE TABLE Payment

(

OrderID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

PaymentType VARCHAR(20) NOT NULL,

TotalAmount INT

)AUTO\_INCREMENT=1000001;

CREATE TABLE Product

(

ProductID VARCHAR(7) NOT NULL PRIMARY KEY,

ProductType VARCHAR(20) NOT NULL,

ProductName VARCHAR(20) NOT NULL,

Cost INT NOT NULL,

Quantity INT NOT NULL,

SellerID VARCHAR(6),

FOREIGN KEY (SellerID) REFERENCES Seller(SellerID)

ON DELETE SET NULL

);

CREATE TABLE CartItem

(

CustomerID VARCHAR(7) NOT NULL,

ProductID VARCHAR(7) NOT NULL,

Quantity INT NOT NULL,

TotalAmount INT NOT NULL,

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Product(ProductID),

PRIMARY KEY(CustomerID,ProductID)

);

CREATE TABLE Orders

(

OrderID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

CustomerID VARCHAR(7) NOT NULL,

ProductID VARCHAR(7) NOT NULL,

OrderDate DATE NOT NULL,

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Product(ProductID),

FOREIGN KEY (OrderID) REFERENCES Payment(OrderID)

)AUTO\_INCREMENT=1000001;

INSERT INTO UserIDandPasswords VALUES

(10001,'abvpwd123','Customer'),

(10002,'axxcvpd3','Customer'),

(10003,'bbvxwd123','Customer'),

(10004,'abbcvp245','Customer'),

(10005,'abvpwd123','Customer'),

(10006,'abvdgd1675','Customer'),

(10007,'lskvpds79','Customer'),

(10008,'bbvpwd320','Customer'),

(10009,'ab1234psh','Customer'),

(10010,'xxv123pqr','Customer'),

('sid101','hwevbf','Seller'),

('sid102','fwgihi','Seller'),

('sid103','bbweuhww','Seller'),

('sid104','wbdfbh','Seller'),

('sid105','uiewbibi','Seller'),

('sid106','ebfiwbef','Seller'),

('sid107','bbwewu','Seller'),

('sid108','feydfuwu','Seller'),

('sid109','iueouow','Seller');

INSERT INTO Customer VALUES

(10001,'abvpwd123','Rajesh','East Phase New Delhi',661023,'8456712345','rajesh@gmail.com'),

(10002,'axxcvpd3','Ajit','Juhu Mumbai',681023,'8456652345','ajit@gmail.com'),

(10003,'bbvxwd123','Rahul','Lions Road Bangalore',661323,'8126712345','rahul@gmail.com'),

(10004,'abbcvp245','Arjun','New Delhi',661023,'8456712345','arjun@gmail.com'),

(10005,'abvpwd123','Raheem','Old City Lucknow',781023,'8456719845','raheem@gmail.com'),

(10006,'abvdgd1675','Charu','Kings Road Chandigarh',661723,'8106712345','charu@gmail.com'),

(10007,'lskvpds79','Sonam','New Delhi',661023,'8456719785','sonam@gmail.com'),

(10008,'bbvpwd320','Priya','Rajeshwari Kolkata',721023,'8896714345','priya@gmail.com'),

(10009,'ab1234psh','Anil','Colaba Mumbai',661093,'8456719845','anil@gmail.com'),

(10010,'xxv123pqr','Riya','Rajouri NewDelhi',760823,'8419812345','riya@gmail.com');

INSERT INTO Seller VALUES

('sid100','bfhbfj','Aman','At post Delhi','9923657432'),

('sid101','hwevbf','Zaad','At post Mumbai','9923657433'),

('sid102','fwgihi','Nazir','At post Bhiwandi','9923657434'),

('sid103','bbweuhww','Aadil','At post Pune','9923657435'),

('sid104','wbdfbh','Anil','At post Bhopal','9923657436'),

('sid105','uiewbibi','Yogesh','At post Mumbai','9923657437'),

('sid106','ebfiwbef','Kiran','At post Delhi','9923657438'),

('sid107','bbwewu','Himanshu','At post Bhiwandi','9923657439'),

('sid108','feydfuwu','Mohan','At post Chennai','9923657440'),

('sid109','iueouow','Bhanupriya','At post Banglore','9923657430');

INSERT INTO Product VALUES

('pid1001','Book','Harry Potter',600,10,'sid101'),

('pid1002','Fashion','Makeup',800,18,'sid105'),

('pid1003','Electronics','iPhone X',100000,40,'sid109'),

('pid1004','Eatables','Ice Cream',30,15,'sid104'),

('pid1005','Book','Rich Dad Poor Dad',200,15,'sid101'),

('pid1006','Electronics','Sony Bravia',75000,7,'sid106'),

('pid1007','Footwear','Adidas Predator',5000,20,'sid108'),

('pid1008','Vehicle','TVS Jupiter',45000,5,'sid102'),

('pid1009','Eatables','Coke',40,50,'sid101'),

('pid1010','Eatables','Bread',20,100,'sid103'),

('pid1011','Eatables','Bisleri',10,100,'sid103'),

('pid1012','Book','Tinkle',120,18,'sid101');

INSERT INTO CartItem VALUES

(10001,'pid1001',2,1200),

(10002,'pid1004',3,90),

(10007,'pid1005',1,200),

(10004,'pid1009',5,200),

(10008,'pid1011',10,100),

(10009,'pid1003',1,100000),

(10008,'pid1009',3,120),

(10002,'pid1005',2,400),

(10006,'pid1002',1,800),

(10004,'pid1006',1,75000);

INSERT INTO Payment(PaymentType,TotalAmount) VALUES

('Net Banking',800),

('Credit/Debit',100000),

('Net Banking',75000),

('UPI',600),

('COD',45000),

('COD',600),

('UPI',200),

('Credit/Debit',40),

('COD',5000),

('UPI',120),

('UPI',10),

('COD',10),

('Net Banking',200),

('COD',30),

('UPI',30);

INSERT INTO Orders(CustomerID,ProductID,OrderDate) VALUES

(10003,'pid1002','2020-10-12'),

(10001,'pid1003','2019-10-23'),

(10005,'pid1006','2018-10-15'),

(10003,'pid1001','2019-10-05'),

(10006,'pid1008','2020-10-06'),

(10007,'pid1001','2019-10-08'),

(10005,'pid1005','2020-10-12'),

(10003,'pid1009','2019-10-02'),

(10001,'pid1007','2019-10-14'),

(10009,'pid1012','2018-10-02'),

(10004,'pid1011','2020-10-13'),

(10007,'pid1011','2018-10-02'),

(10005,'pid1005','2018-10-17'),

(10001,'pid1004','2018-10-18'),

(10003,'pid1004','2020-10-02');

/\*

SELECT CustomerID,Name,PhoneNumber,Email

FROM Customer

WHERE Address LIKE '%Mumbai%';

SELECT \* FROM Product;

UPDATE Product SET cost=0.9\*cost

WHERE SellerID='sid101';

SELECT \* FROM Product;

SELECT DISTINCT PaymentType AS DifferentTypesOfPaymentsAccepted

FROM Payment;

\*/

SET sql\_mode = '';

/\*

SELECT P.ProductID,P.ProductName,S.SellerID,S.Name,P.Cost

FROM Product P,Seller S

WHERE P.SellerID=S.SellerID

GROUP BY P.SellerID

HAVING Cost=MAX(Cost);

SELECT DISTINCT Address

FROM Seller;

SELECT C.CustomerID, C.Name, COUNT(\*) AS NumberOfItems

FROM Customer C,CartItem CI

WHERE C.CustomerID=CI.CustomerID

GROUP BY C.CustomerID

ORDER BY NumberOfItems desc;

SELECT \* FROM Orders

WHERE OrderDate > '2019-12-31';

SELECT ProductType, COUNT(\*) AS NumberOfProducts

FROM Product

GROUP BY ProductType;

SELECT ProductType,AVG(COST) AS AverageCost

FROM Product

GROUP BY ProductType;

SELECT C.Name,O.OrderID,O.ProductID,O.OrderDate

FROM Customer as C, Orders as O

WHERE C.Name='Rajesh' and C.CustomerID=O.CustomerID;

SELECT \* FROM Customer as C

WHERE C.CustomerID IN(SELECT O.CustomerID

From Orders AS O, CartItem as CI

WHERE O.CustomerID=C.CustomerID AND C.CustomerID=CI.CustomerID);

SELECT P.PaymentType,COUNT(\*) AS NumberOfPayments

FROM Payment P, Orders O

WHERE P.OrderID=O.OrderID AND

O.CustomerID IN(SELECT CustomerID

FROM Customer

WHERE CustomerID=O.CustomerID AND

Address LIKE '%Mumbai%' OR Address LIKE '%Delhi%')

GROUP BY P.PaymentType;

SELECT C.CustomerID, C.Name, O.OrderID, P.TotalAmount

FROM Customer AS C, Orders AS O, Payment AS P

WHERE O.CustomerID=C.CustomerID AND P.OrderID=O.OrderID

AND P.TotalAmount=(SELECT MAX(P1.TotalAmount)

FROM Payment AS P1, Orders AS O1

WHERE P1.OrderID=O1.OrderID AND O1.CustomerID=C.CustomerID);

SELECT C.CustomerID, C.Name,COUNT(\*) AS NoOfOrders FROM Customer C

INNER JOIN Orders O

ON O.CustomerID=C.CustomerID

GROUP BY C.CustomerID

HAVING NoOfOrders>=ALL(SELECT COUNT(\*) FROM

Orders

GROUP BY CustomerID);

SELECT DISTINCT C.CustomerID, C.Name FROM Customer C,Orders O, Product P,Seller S

WHERE O.CustomerID=C.CustomerID AND O.ProductID=P.ProductID AND P.SellerID=S.SellerID AND

P.SellerID IN(SELECT SellerID

FROM Product

GROUP BY SellerID

HAVING COUNT(\*)>1);

SELECT O.CustomerID, O.OrderID, O.ProductID, O.OrderDate FROM Orders AS O

WHERE O.CustomerID = (SELECT CustomerID

FROM Orders

GROUP BY CustomerID

ORDER BY COUNT(CustomerID) DESC LIMIT 1);

SELECT \* FROM Customer

WHERE CustomerID!=ALL(SELECT CustomerID

FROM Orders);

\*/

DROP VIEW OrderDet;

CREATE VIEW OrderDet AS

SELECT O.OrderID, C.Name AS CustomerName, S.Name AS SellerName

FROM Orders AS O, Customer AS C, Product AS P, Seller AS S

WHERE O.CustomerID=C.CustomerID AND O.ProductID=P.ProductID AND P.SellerID=S.SellerID;

/\*SELECT \* FROM OrderDet;\*/

DROP VIEW ProductsWithPriceLessThanRs1000;

CREATE VIEW ProductsWithPriceLessThanRs1000 AS

SELECT \* FROM Product

WHERE Cost<1000

ORDER BY Cost;

/\*SELECT \* FROM ProductsWithPriceLessThanRs1000;\*/

DROP VIEW HighestPurchaseOfDay;

CREATE VIEW HighestPurchaseOfDay AS

SELECT C.CustomerID, C.Name, MAX(P.Cost) as PurchaseAmount, O.OrderDate

FROM Customer C, Orders O, Product P

WHERE O.CustomerID=C.CustomerID AND O.ProductID=P.ProductID

GROUP BY O.OrderDate;

SET sql\_mode = '';

/\*SELECT \* FROM HighestPurchaseOfDay;\*/

DROP VIEW CheapestProductOfSeller;

CREATE VIEW CheapestProductOfSeller AS

SELECT S.SellerID, P.ProductID, P.ProductName, P.Cost

FROM Product AS P, Seller AS S

WHERE P.SellerID=S.SellerID AND P.ProductID = (SELECT ProductID

FROM Product

WHERE SellerID=S.SellerID

ORDER BY COST LIMIT 1);

/\*SELECT \* FROM CheapestProductOfSeller;\*/

DROP VIEW HighestPayType;

CREATE VIEW HighestPayType AS

SELECT PaymentType,Max(TotalAmount)

FROM Payment

GROUP BY PaymentType ;

/\*SELECT \* FROM HighestPayType;\*/

DROP PROCEDURE IF EXISTS AddToCart;

DELIMITER //

CREATE PROCEDURE AddToCart(IN ProductID VARCHAR(7), IN CustomerID VARCHAR(7), IN Quantity INT)

BEGIN

DECLARE TotalAmount int;

DECLARE Stock int;

DECLARE ErrorMessage VARCHAR(255);

SET ErrorMessage = 'Not Enough Stock Available';

SELECT P.Quantity INTO Stock FROM Product AS P WHERE P.ProductID=ProductID;

IF Quantity<=Stock THEN

SELECT P.Cost\*Quantity INTO TotalAmount From Product as P WHERE P.ProductID=ProductID;

INSERT INTO CartItem VALUES(CustomerID,ProductID,Quantity,TotalAmount);

IF Quantity=Stock THEN

DELETE FROM Product P WHERE ProductID=P.ProductID;

ELSE

UPDATE Product P SET P.Quantity=Stock-Quantity WHERE ProductID=P.ProductID;

END IF;

ELSE

SIGNAL SQLSTATE '45000' SET message\_text = ErrorMessage;

END IF;

END//

DELIMITER ;

/\*

SELECT \* FROM CartItem;

CALL AddToCart('pid1011','10001',20);

SELECT \* FROM CartItem;

\*/

DROP PROCEDURE IF EXISTS RemoveFromCart;

DELIMITER //

CREATE PROCEDURE RemoveFromCart(IN ProductID VARCHAR(7),IN CustomerID VARCHAR(7),IN NoOfItems INT)

BEGIN

DECLARE Quan INT;

DECLARE QLeft INT;

SELECT C.Quantity INTO Quan

FROM CartItem C

WHERE C.CustomerID=CustomerID AND C.ProductID=ProductID;

IF NoOfItems<Quan THEN SET QLeft=Quan-NoOfItems;

ELSE SET QLeft=0;

END IF;

UPDATE CartItem C

SET C.Quantity=QLeft

WHERE C.ProductID=ProductID AND C.CustomerID=CustomerID;

END //

DELIMITER;

/\*

SELECT \* FROM CartItem;

CALL RemoveFromCart('pid1011','10001',5);

SELECT \* FROM CartItem;

\*/

DROP PROCEDURE IF EXISTS GetSellerProducts;

DELIMITER //

CREATE PROCEDURE GetSellerProducts(IN SellerID VARCHAR(6))

BEGIN

SELECT \* FROM Product AS P

WHERE P.SellerID=SellerID;

END//

DELIMITER ;

/\*

CALL GetSellerProducts('sid101');

\*/

DROP PROCEDURE IF EXISTS GetCustomersCartItems;

DELIMITER //

CREATE PROCEDURE GetCustomersCartItems(IN CustomerID VARCHAR(6))

BEGIN

SELECT \* FROM CartItem AS CI

WHERE CI.CustomerID=CustomerID;

END//

DELIMITER ;

/\*

CALL GetCustomersCartItems('10001');

\*/

DROP PROCEDURE IF EXISTS GetCustomersOrders;

DELIMITER //

CREATE PROCEDURE GetCustomersOrders(IN CustomerID VARCHAR(6))

BEGIN

SELECT OrderID,ProductID,OrderDate FROM Orders AS O

WHERE O.CustomerID=CustomerID;

END//

DELIMITER ;

/\*

CALL GetCustomersOrders('10003');

\*/

DROP FUNCTION IF EXISTS GetCartCost;

DELIMITER //

CREATE FUNCTION GetCartCost(customerID VARCHAR(7))

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE TotalCost INT DEFAULT 0;

SELECT SUM(TotalAmount) INTO TotalCost

FROM CartItem AS CI

WHERE CI.CustomerID=customerID;

RETURN TotalCost;

END //

DELIMITER ;

/\*

SELECT GetCartCost('10002') AS TotalAmountInCart;

SELECT GetCartCost('10008') AS TotalAmountInCart;

\*/

DROP FUNCTION IF EXISTS GetProductLevel;

DELIMITER //

CREATE FUNCTION GetProductLevel(PName VARCHAR(20))

RETURNS VARCHAR(15)

DETERMINISTIC

BEGIN

DECLARE val INT DEFAULT 0;

DECLARE level varchar(15);

SELECT Cost INTO val FROM Product WHERE ProductName=PName;

IF val<10000 THEN

SET level='AFFORDABLE';

ELSEIF val<=70000 THEN

SET level='MID-RANGE';

ELSE

SET level='EXPENSIVE';

END IF;

RETURN (level);

END //

DELIMITER ;

/\*

SELECT GetProductLevel('Sony Bravia') AS ProductLevel;

SELECT GetProductLevel('Bisleri') AS ProductLevel;

\*/

DROP FUNCTION IF EXISTS GetExpensiveProduct;

DELIMITER //

CREATE FUNCTION GetExpensiveProduct()

RETURNS varchar(20)

DETERMINISTIC

BEGIN

DECLARE val varchar(20);

SELECT ProductName INTO val FROM Product WHERE Cost=(SELECT MAX(Cost) FROM Product P) ;

RETURN val;

END //

DELIMITER ;

SELECT GetExpensiveProduct() AS ExpensiveProduct;

DROP FUNCTION IF EXISTS GetStock;

DELIMITER //

CREATE FUNCTION GetStock(ProductID VARCHAR(7))

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE Stock INT;

SELECT Quantity INTO Stock FROM Product P

WHERE P.ProductID=ProductID;

RETURN Stock;

END //

DELIMITER ;

/\*

SELECT GetStock('pid1003') AS Stock;

SELECT GetStock('pid1006') AS Stock;

\*/

DROP FUNCTION IF EXISTS GetContactDetailOfSeller;

DELIMITER //

CREATE FUNCTION GetContactDetailOfSeller(ProductID VARCHAR(7))

RETURNS VARCHAR(10)

DETERMINISTIC

BEGIN

DECLARE ContactNo VARCHAR(10);

SELECT S.PhoneNumber INTO ContactNo FROM Product P, Seller S

WHERE P.ProductID=ProductID AND S.SellerID=P.SellerID;

RETURN ContactNo;

END //

DELIMITER ;

/\*

SELECT GetContactDetailOfSeller('pid1005') AS ContactNo;

SELECT GetContactDetailOfSeller('pid1011') AS ContactNo;

\*/

DROP TRIGGER IF EXISTS BuyProductTrigger;

DELIMITER //

CREATE TRIGGER BuyProductTrigger

AFTER DELETE ON CartItem

FOR EACH ROW

BEGIN

DECLARE Quantity INT;

DECLARE Total INT;

DECLARE Price INT;

DECLARE PaymentID VARCHAR(7);

SELECT Cost INTO Price FROM Product WHERE ProductID=old.ProductID;

SET Total=Price\*old.Quantity;

INSERT INTO Payment(PaymentType,TotalAmount) VALUES ('COD',Total);

SELECT OrderID INTO PaymentID FROM Payment ORDER BY OrderID DESC LIMIT 1;

INSERT INTO Orders VALUES (PaymentID,old.CustomerID,old.ProductID,CURDATE());

UPDATE Product P SET P.Quantity=P.Quantity-old.Quantity

WHERE P.ProductID=old.ProductID;

END //

DELIMITER ;

DROP PROCEDURE IF EXISTS BuyProduct;

DELIMITER //

CREATE PROCEDURE BuyProduct(IN CustomerID VARCHAR(7),IN ProductID VARCHAR(7),IN PaymentType VARCHAR(7))

BEGIN

DECLARE PaymentID VARCHAR(7);

DELETE FROM CartItem C

WHERE C.CustomerID=CustomerID AND C.ProductID=ProductID;

SELECT OrderID INTO PaymentID FROM Payment ORDER BY OrderID DESC LIMIT 1;

UPDATE Payment P SET P.PaymentType=PaymentType WHERE P.OrderID=PaymentID;

END //

DELIMITER ;

/\*

SELECT \* FROM CartItem;

SELECT \* FROM Product;

SELECT \* FROM Orders;

SELECT \* FROM Payment;

CALL BuyProduct(10008,'pid1009','Net Banking');

SELECT \* FROM CartItem;

SELECT \* FROM Product;

SELECT \* FROM Orders;

SELECT \* FROM Payment;

\*/

DROP TRIGGER IF EXISTS ApplyDiscount;

DELIMITER //

CREATE TRIGGER ApplyDiscount

BEFORE INSERT ON Payment

FOR EACH ROW

BEGIN

IF new.TotalAmount>=75000 THEN

SET new.TotalAmount=0.9\*new.TotalAmount;

END IF;

END//

DELIMITER ;

/\*

SELECT \* FROM CartItem;

SELECT \* FROM Product;

CALL BuyProduct(10009,'pid1003','COD');

SELECT \* FROM Payment;

\*/

DROP TRIGGER IF EXISTS Refund;

DELIMITER //

CREATE TRIGGER Refund

AFTER DELETE ON Orders

FOR EACH ROW

BEGIN

DELETE FROM Payment

WHERE OrderID=old.OrderID;

END//

DELIMITER ;

DROP PROCEDURE IF EXISTS ReturnOrder;

DELIMITER //

CREATE PROCEDURE ReturnOrder(IN OrderID VARCHAR(7))

BEGIN

DELETE FROM Orders AS O

WHERE O.OrderID = OrderID;

END//

DELIMITER ;

/\*

SELECT \* FROM Orders;

SELECT \* FROM Payment;

CALL ReturnOrder('1000004');

SELECT \* FROM Orders;

SELECT \* FROM Payment;

\*/

DELIMITER //

CREATE TRIGGER insert\_payment

AFTER INSERT ON Orders

FOR EACH ROW

BEGIN

INSERT INTO Payment (OrderID, PaymentType, TotalAmount)

VALUES (NEW.OrderID, 'Unpaid', 0);

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER update\_payment\_totalamount

AFTER INSERT ON CartItem

FOR EACH ROW

BEGIN

UPDATE Payment

SET TotalAmount = TotalAmount + NEW.TotalAmount

WHERE OrderID = (SELECT MAX(OrderID) FROM Orders WHERE CustomerID = NEW.CustomerID);

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER update\_product\_quantity

AFTER UPDATE ON CartItem

FOR EACH ROW

BEGIN

UPDATE Product

SET Quantity = Quantity - (NEW.Quantity - OLD.Quantity)

WHERE ProductID = NEW.ProductID;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER update\_cartitem\_totalamount

AFTER UPDATE ON Product

FOR EACH ROW

BEGIN

UPDATE CartItem

SET TotalAmount = Quantity \* Cost

WHERE ProductID = NEW.ProductID;

END //

DELIMITER ;

PL-SQL

SQL> SET SERVEROUTPUT ON;

SQL> DECLARE

var1 INTEGER;

var2 REAL;

var3 varchar2(20) ;

BEGIN

null;

END;

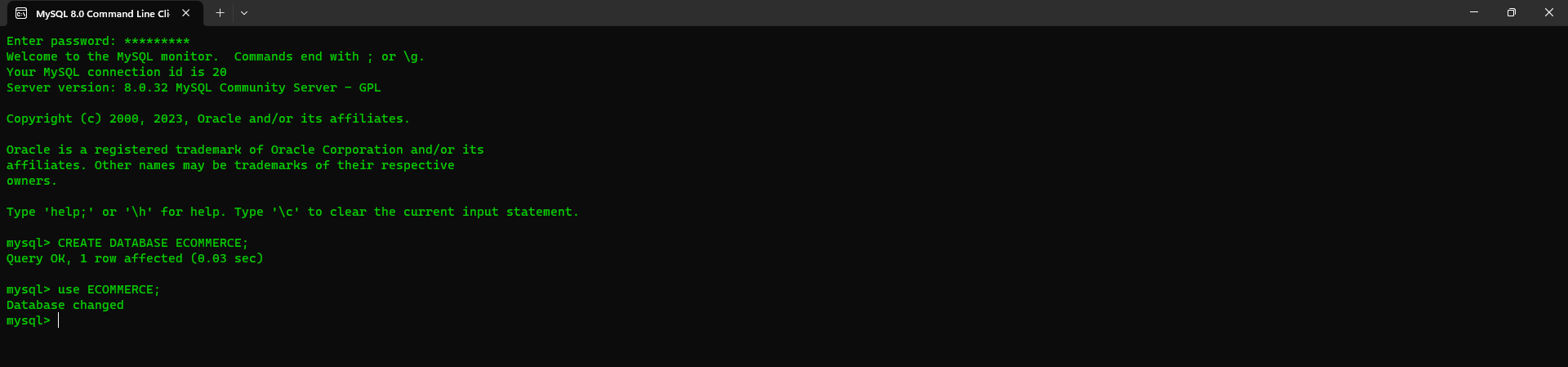
/

Output:

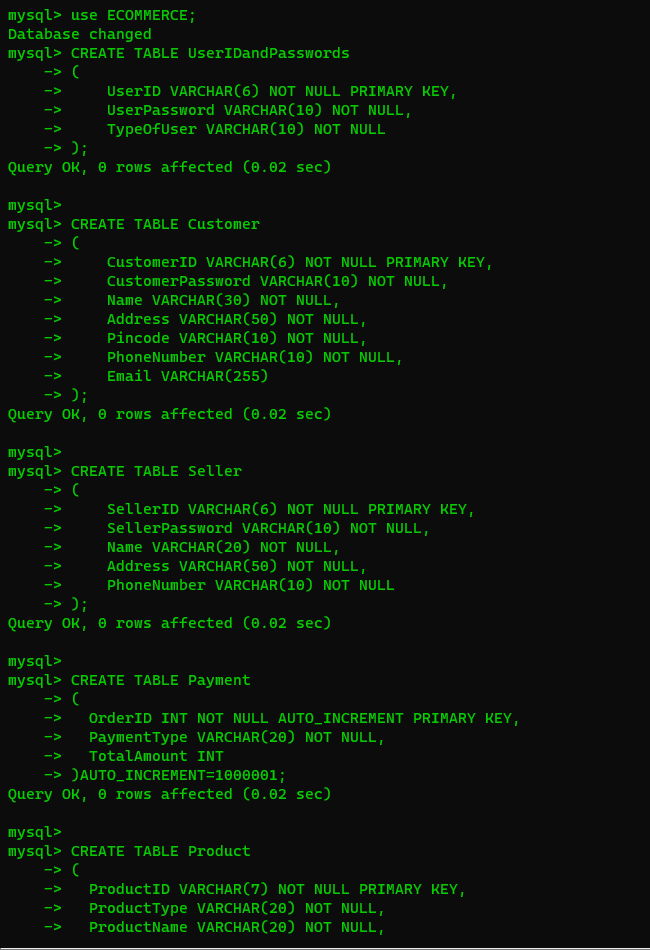
PL/SQL procedure successfully completed

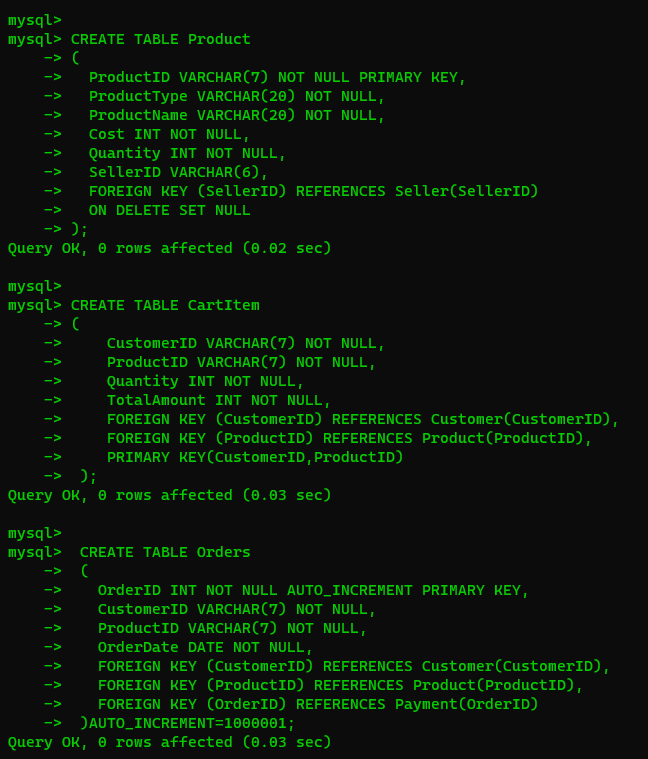
OUTPUT

Creating Database

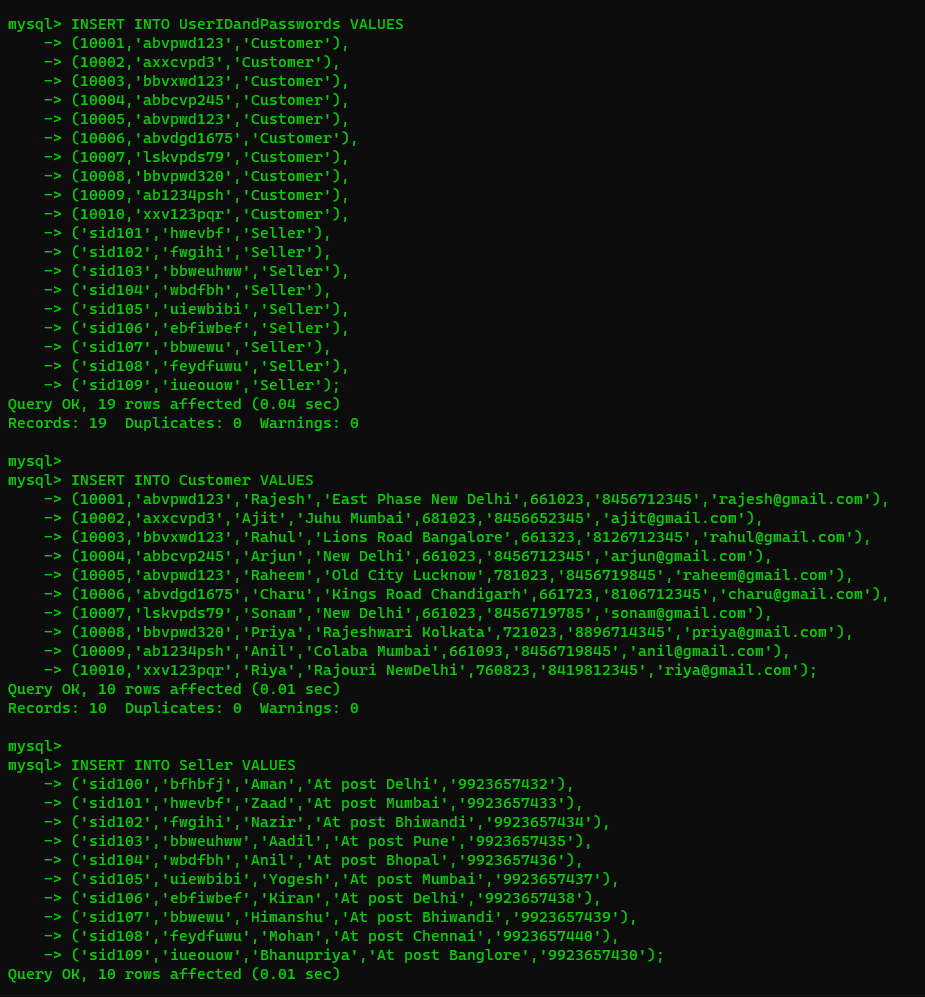


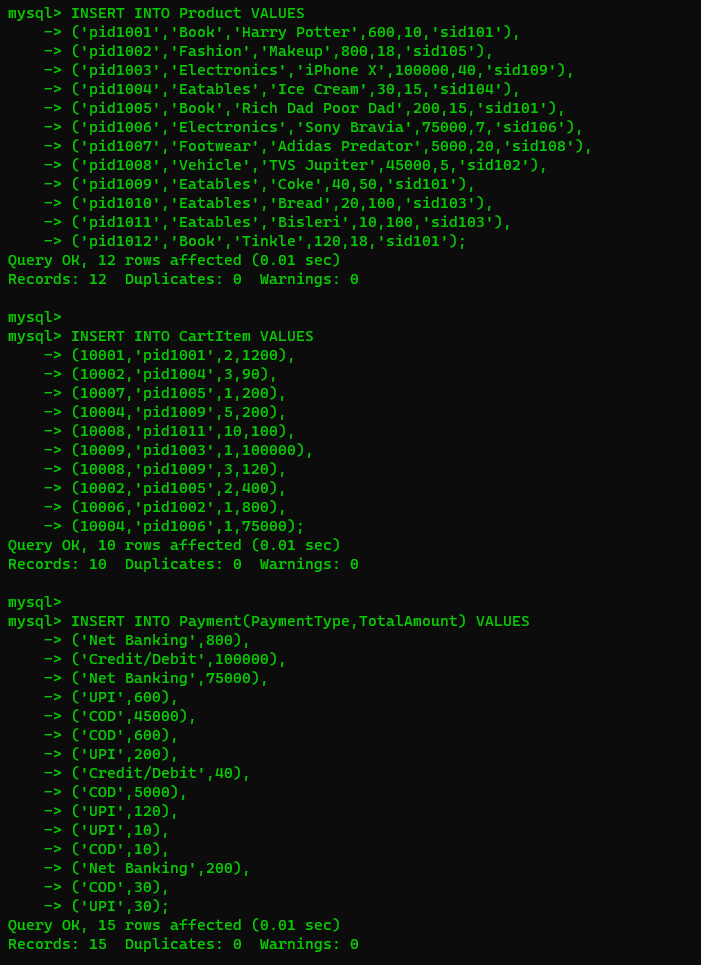
Creating Tables

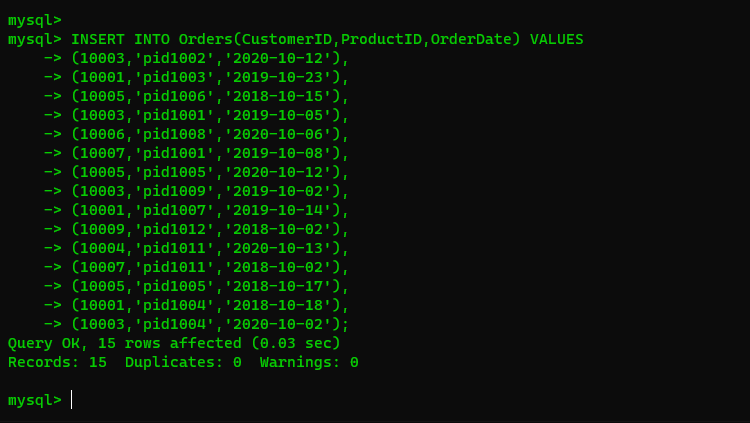




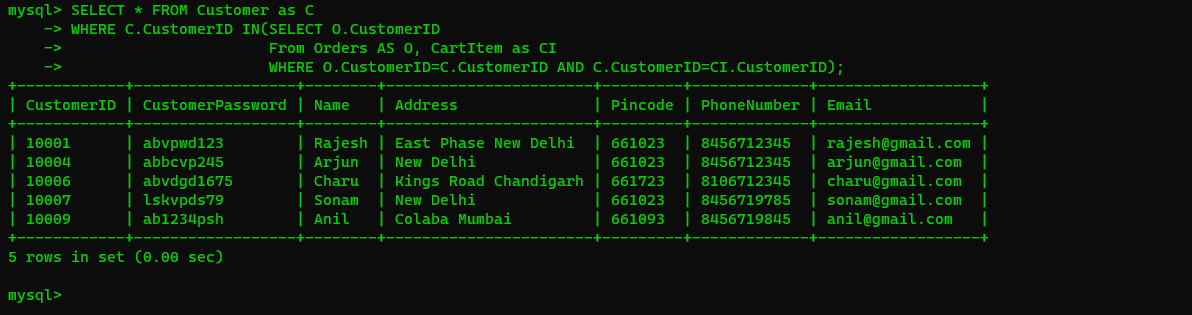
Inserting Values

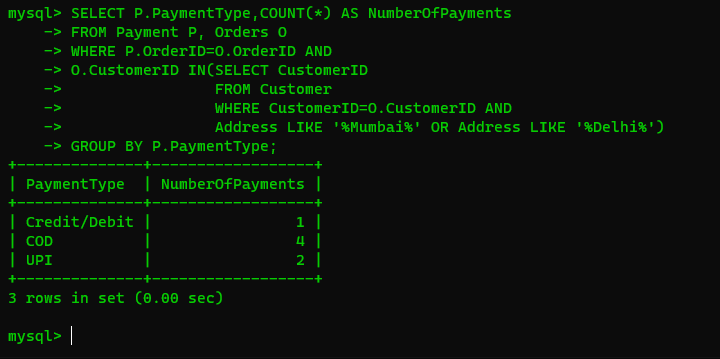




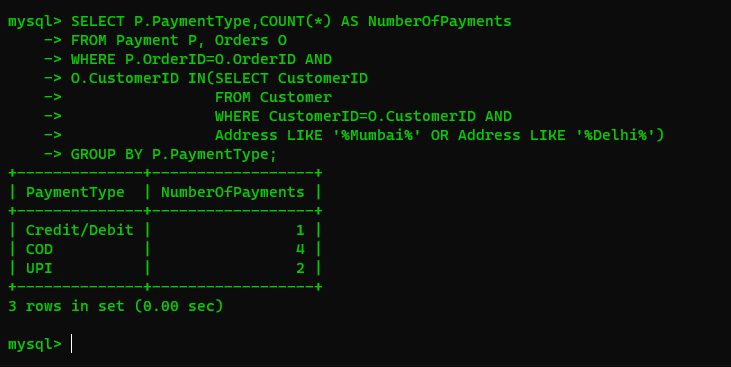


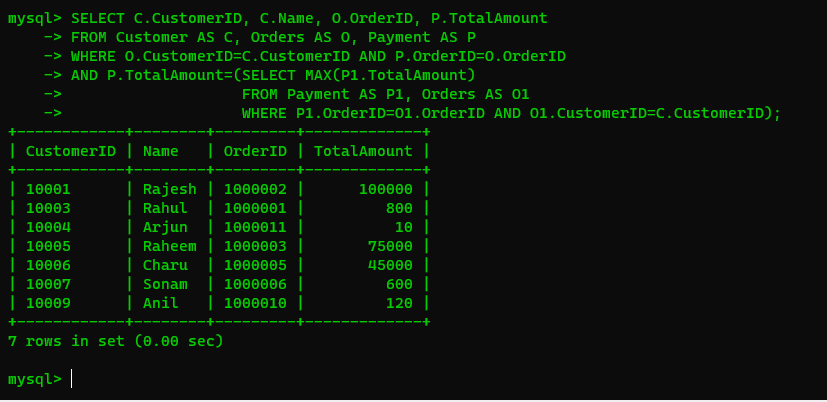
SELECT QUERIES

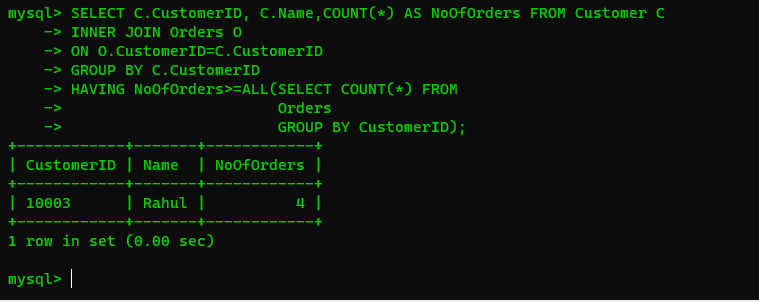


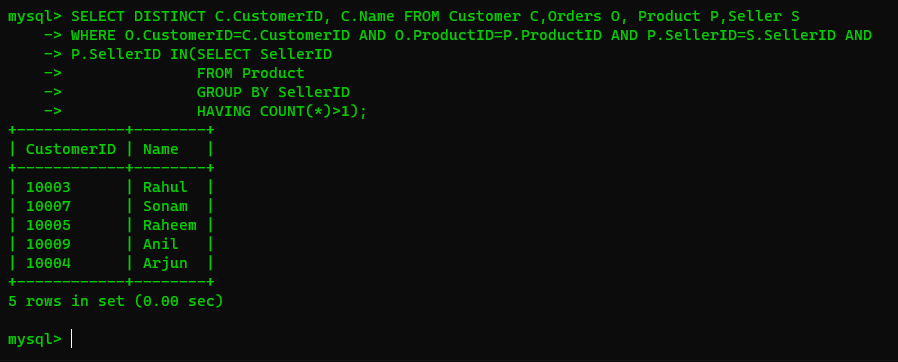


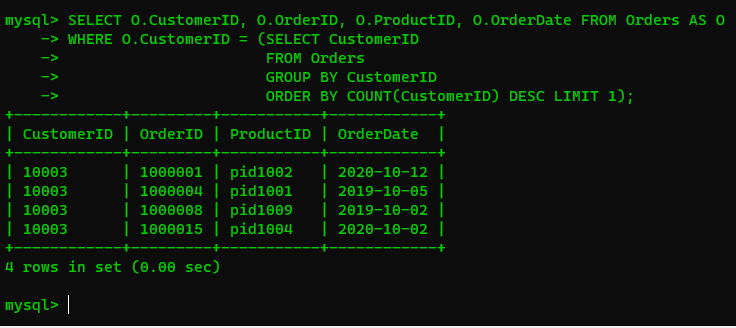
Join Queries

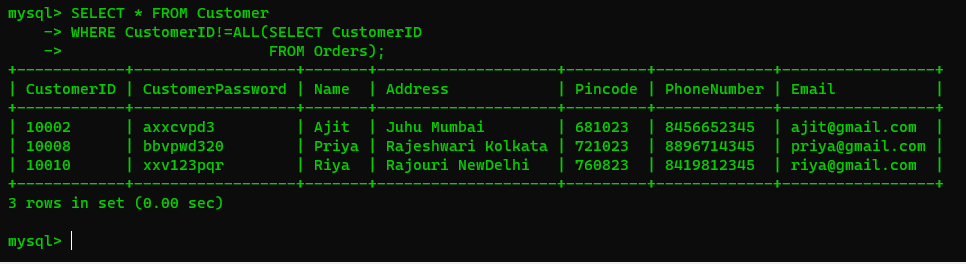


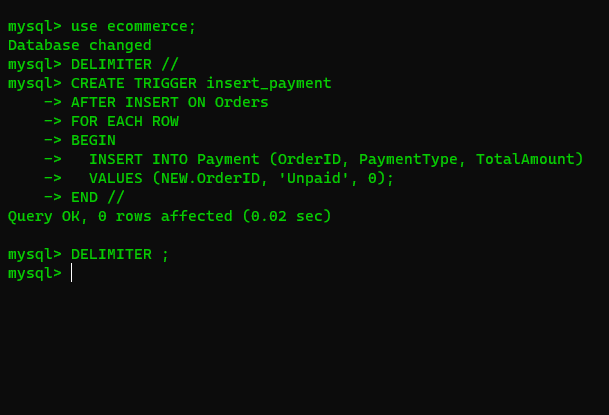


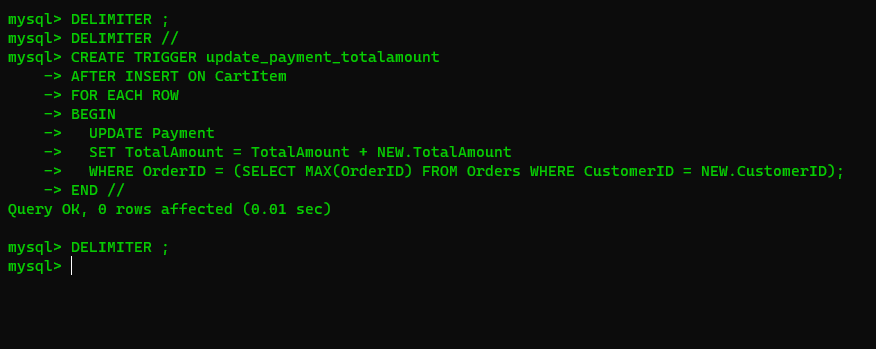


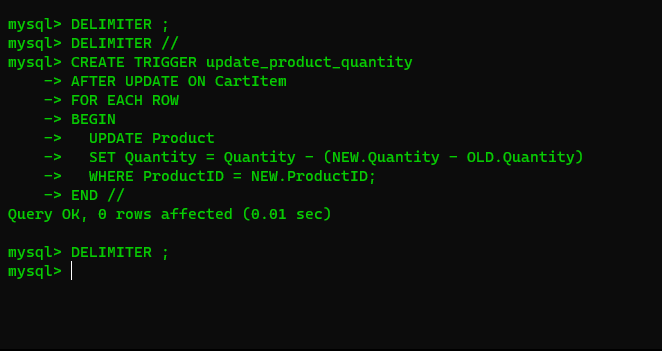


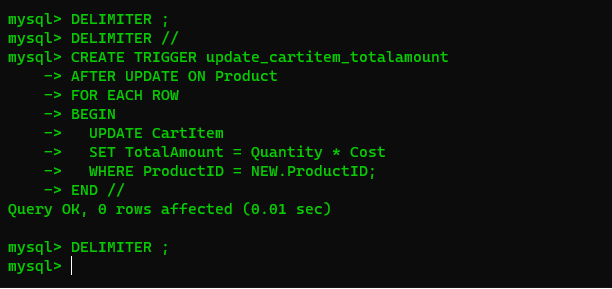












Normalization

Normalization is a process of organizing data in a database to reduce redundancy and dependency. There are different levels of normalization, known as normal forms, and each normal form has its own set of rules. The goal of normalization is to minimize data duplication and improve data integrity.

Let's take a sample table to demonstrate the normalization concepts. Suppose we have a table called "Customer Orders" with the following columns:

First, let's start with a denormalized customer table:

| **Customer ID** | **Name** | **Phone Number** | **Email** | **Address** | **City** | **State** | **Zip** | **Credit Card Number** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 001 | John Smith | 555-1234 | john.smith@example.com | 123 Main St. | Anytown | NY | 12345 | 1234-5678-9012-3456 |
| 002 | Jane Doe | 555-5678 | jane.doe@example.com | 456 Maple Ave. | Anytown | NY | 12345 | 2345-6789-0123-4567 |
| 003 | Bob Johnson | 555-9012 | bob.johnson@example.com | 789 Oak Rd. | Othertown | CA | 67890 | 3456-7890-1234-5678 |

1. First Normal Form (1NF):

To achieve 1NF, we need to make sure that each column contains only one value and that we have a primary key.

Next, let's apply the first normal form (1NF) by ensuring that each column contains atomic values:

Customers table (1NF):

| **Customer ID** | **Name** | **Phone Number** | **Email** | **Address** | **City** | **State** | **Zip** | **Credit Card Number** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 001 | John Smith | 555-1234 | john.smith@example.com | 123 Main St. | Anytown | NY | 12345 | 1234-5678-9012-3456 |
| 002 | Jane Doe | 555-5678 | jane.doe@example.com | 456 Maple Ave. | Anytown | NY | 12345 | 2345-6789-0123-4567 |
| 003 | Bob Johnson | 555-9012 | bob.johnson@example.com | 789 Oak Rd. | Othertown | CA | 67890 | 3456-7890-1234-5678 |

2.Second normal form (2NF):

The table should be in 1NF, and each non-key column should be fully dependent on the primary key. To achieve 2NF, we need to ensure that each non-key column is fully dependent on the primary key.

Next, let's apply the second normal form (2NF) by ensuring that each column in a table depends on the primary key:

Customers table (2NF):

| Customer ID | Name | Phone Number | Email | Address | City | State | Zip |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 001 | John Smith | 555-1234 | john.smith@example.com | 123 Main St. | Anytown | NY | 12345 |
| 002 | Jane Doe | 555-5678 | jane.doe@example.com | 456 Maple Ave. | Anytown | NY | 12345 |
| 003 | Bob Johnson | 555-9012 | bob.johnson@example.com | 789 Oak Rd. | Othertown | CA | 67890 |

Credit Cards table (2NF):

| **Customer ID** | **Credit Card Number** |
| --- | --- |
| 001 | 1234-5678-9012-3456 |
| 002 | 2345-6789-0123-4567 |
| 003 | 3456-7890-1234-5678 |

3.Third normal form (3NF):

The table should be in 2NF, and there should be no transitive dependencies. To achieve 3NF, we need to eliminate any transitive dependencies. A transitive dependency is when a non-key column depends on another non-key column.

Finally, let's apply the third normal form (3NF) by ensuring that there are no transitive dependencies:

Customers table (3NF):

| **Customer ID** | **Name** | **Phone Number** | **Email** | **Address** | **City** | **State** | **Zip** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 001 | John Smith | 555-1234 | john.smith@example.com | 123 Main St. | Anytown | NY | 12345 |
| 002 | Jane Doe | 555-5678 | jane.doe@example.com | 456 Maple Ave. | Anytown | NY | 12345 |
| 003 | Bob Johnson | 555-9012 | bob.johnson@example.com | 789 Oak Rd. | Othertown | CA | 67890 |

Credit Cards table (3NF):

| **Customer ID** | **Credit Card ID** |
| --- | --- |
| 001 | 001 |
| 002 | 002 |
| 003 | 003 |

Credit Cards Info table (3NF):

| **Credit Card ID** | **Credit Card Number** |
| --- | --- |
| 001 | 1234-5678-9012-3456 |
| 002 | 2345-6789-0123-4567 |
| 003 | 3456-7890-1234-5678 |

In this example, we first applied the first normal form by ensuring that each column contains atomic values. Then, we applied the second normal form by ensuring that each column in a table depends on the primary key. Finally, we applied the third normal form by ensuring that there are no transitive dependencies.

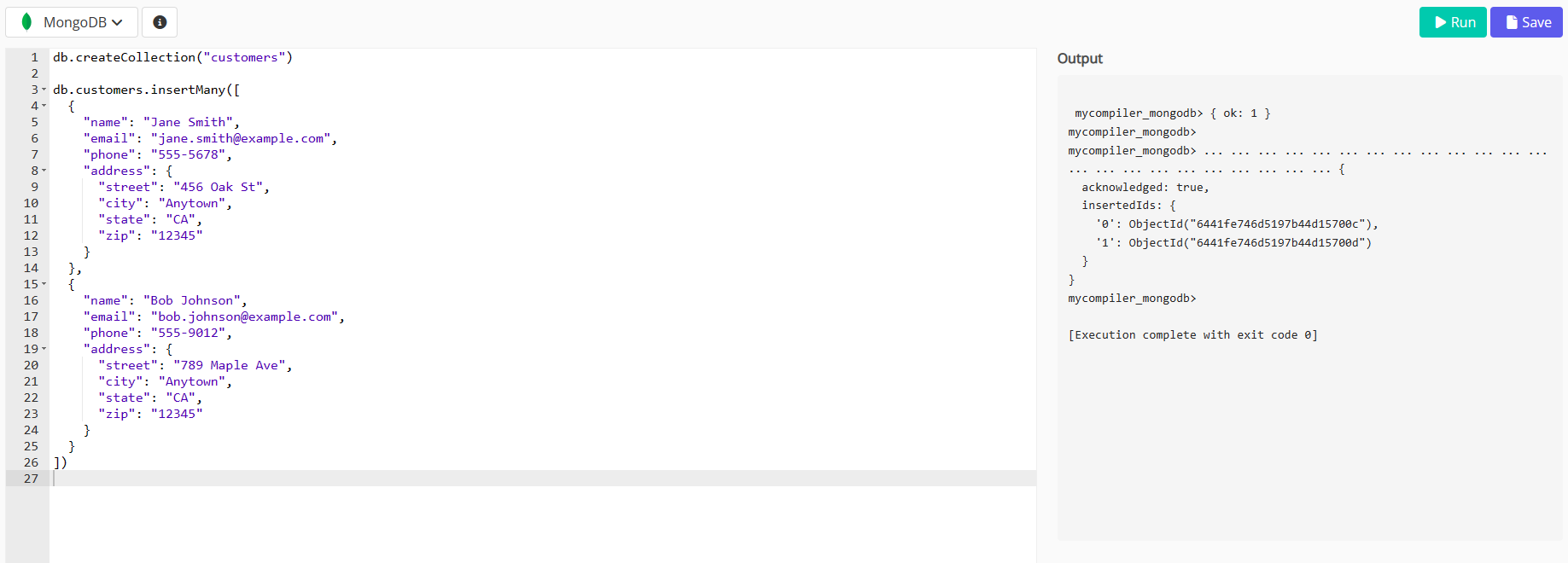
This normalization process has resulted in three separate tables that eliminate redundancy and ensure data consistency. The first table contains information about customers, the second table contains information about credit cards associated with each customer, and the third table contains information about each credit card number.

NO SQL

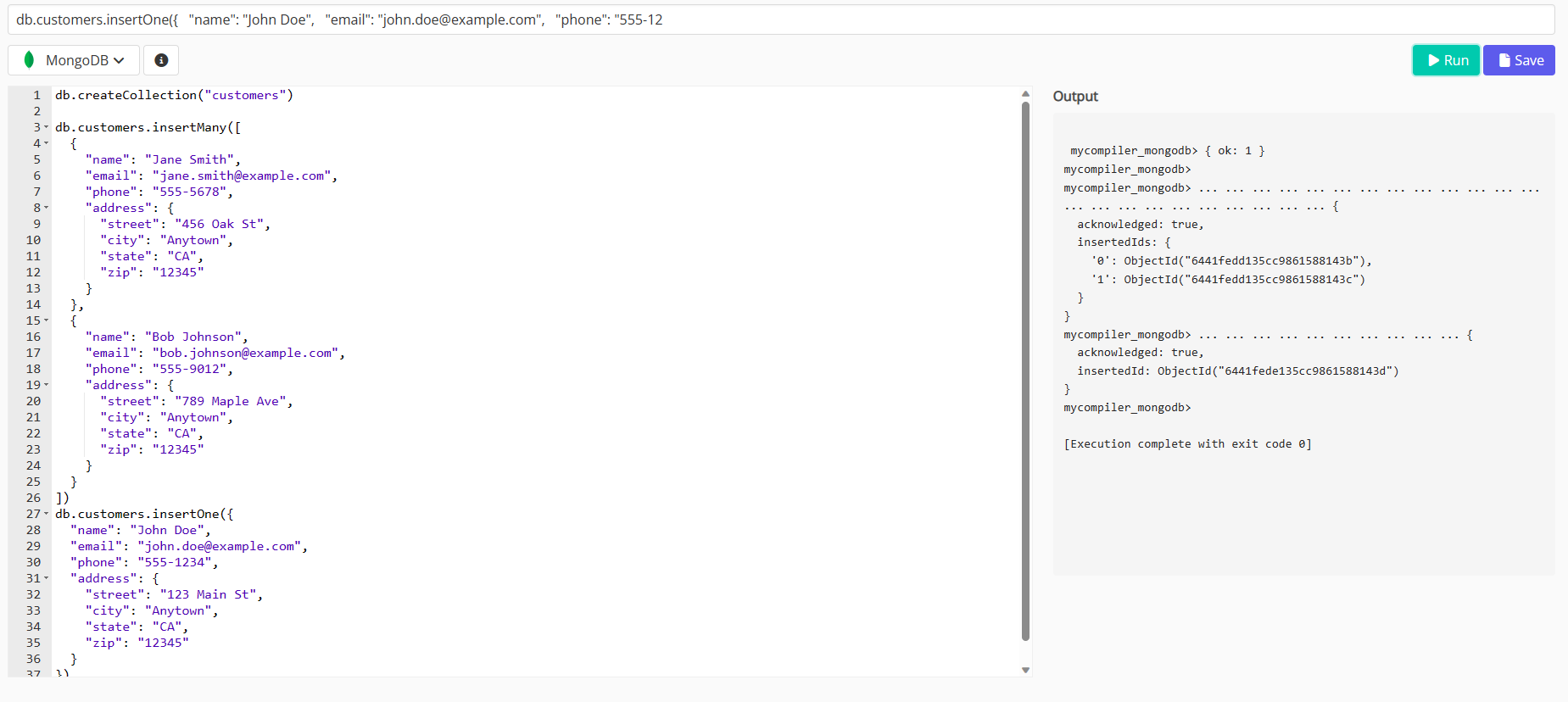
NoSQL is termed non-SQL. We won’t use SQL to write queries in No SQL. It is not relational but it gives us an organized way of storing data. The data is stored in the form of documents rather than tabular form. The best example for NoSql is Mongo DB. In SQL we will use the term key-value pairs but in Mongo DB we will use field-value pairs. Documents are stored and the group of documents is called “Collection”. The document will be in JSON format. The data is called a “Document” and the collection of documents is called a “Collection”.

QUERIES:

CREATING:



INSERTION:



SELECTION:

