BANKING MANAGEMENT SYSTEM USING SQL

PROJECT REPORT

Submitted by

PREM LOHIA (RA2211029010007)

AADIT VINAYAK (RA2211029010012)

CHIRANJEEV KUMAR (RA2211029010019)

Under the guidance of

Dr P. Mahalakshmi

Assistant Professor, Department of Networking and Communications

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

with specialization in Computer Networking

SRM

INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University u/s 3 of UGC Act, 1956)

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603 203

MAY 2024

SRM

INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University u/s 3 of UGC Act, 1956)

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603 203

BONAFIDE CERTIFICATE

Certified that this Project Report titled "**BANKING MANAGEMENT SYSTEM USING SQL**" is the bonafide work done by:

PREM LOHIA (RA2211029010007)

AADIT VINAYAK (RA2211029010012)

CHIRANJEEV KUMAR (RA2211029010019)

who completed the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE

Dr P. Mahalakshmi

DBMS-Course Faculty

Assistant Professor

Department of Networking and Communications

SRMIST

SIGNATURE

Dr Annapurani Panaiyappan

Head of the Department

Department of Networking and Communications

SRMIST

TABLE OF CONTENTS

ABSTRACT

The Banking Management System (BMS) revolutionizes banking operations with its integrated approach, combining advanced technologies and robust functionalities. Featuring a user-friendly interface accessible to both customers and bank personnel, secure authentication protocols facilitate a range of transactions including fund transfers, account inquiries, and loan applications, all processed in real-time to ensure accuracy and speed, thus fostering customer satisfaction and loyalty. For bank administrators, the BMS offers a centralized platform encompassing modules for customer relationship management, account oversight, risk assessment, compliance monitoring, and reporting, with automation reducing manual errors and optimizing resource allocation. A standout feature is its advanced analytics capabilities, harnessing big data and machine learning algorithms to provide invaluable insights into customer behavior, market dynamics, and risk profiles, empowering banks to make informed, data-driven decisions, personalize services, and effectively mitigate risks. Moreover, the BMS is designed for scalability and adaptability, whether deployed on-premises or in the cloud, ensuring it can readily meet the evolving demands and technological advancements within the banking industry.

Chapter 1

INTRODUCTION

In the fast-paced realm of banking, the efficient management of operations stands as a cornerstone for success. With the advent of technological advancements and the ever- increasing expectations of customers, banking institutions are compelled to embrace innovative solutions that streamline processes, enhance security, and elevate customer experiences. Enter the Banking Management System (BMS), a comprehensive solution designed to revolutionize the way banks manage their operations.

The introduction of the BMS marks a significant milestone in the evolution of banking systems, offering a holistic approach to address the multifaceted challenges faced by modern financial institutions. By amalgamating cutting-edge technologies, sophisticated analytics, and robust security protocols, the BMS emerges as a transformative tool poised to reshape the banking landscape.

Moreover, the introduction highlights the overarching objectives of the BMS, including but not limited to enhancing accessibility, optimizing resource utilization, ensuring regulatory compliance, and fostering innovation. By providing a comprehensive overview, this introduction sets the stage for a deeper exploration into the intricate workings and profound impact of the Banking Management System on the entire banking industry.

Chapter 2

LITERATURE SURVEY

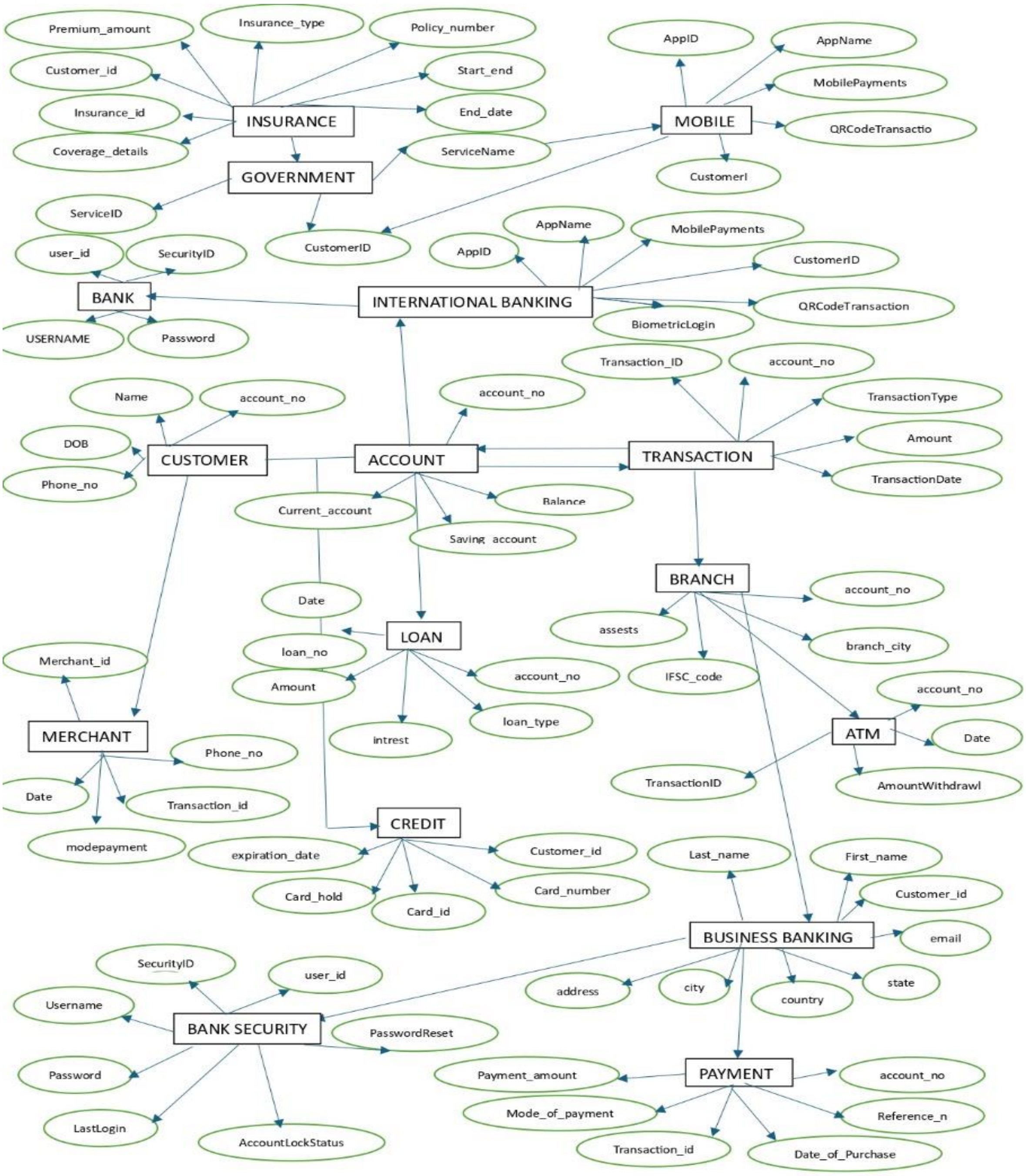
1. **“Fundamentals of Database Systems”**
   Authors: Ramez Elmasri, Shamkant B. Navathe
   Edition: Sixth
   Publication Year: 2020
   Publisher: Pearson Education
   We referred to this book to understand the basic concept of Database management system Normalization and how to apply them in our project to make and banking management system and use them to our advantage.

2. **Database System Concepts**
   Authors: Abraham Silberschatz, Henry F. Korth, and S. Sudarshan
   Edition: Seventh
   Publication Year: 2019
   Publisher: Tata McGraw Hill
   We referred to this book to understand the basic concept of the Database management system “Concurrency Control." and how to apply them in our project to make and banking management system and also use them to our advantage.

3. **An Introduction to Database Systems**
   Authors: CJ Date, A Kannan, S Swaminathan
   Publication Year: 2006
   Publisher: Addison-Wesley
   We referred to this book to understand the basic concept of the Database management system “Database Normalization." and how to apply them in our project to make and banking management system and also use them in our advantage.

Chapter 3

ENTITY-RELATIONSHIP DIAGRAM



ENTITIES AND THEIR ATTRIBUTES

- Customer:
  - Attributes: account_no, Name, phone_no, Email_id, DOB, address, user_id

- Account:
    - Attributes: account_no, Saving_account, Current_account, Balance
    - Linked to Customer via foreign key (account_no)
- Branch:
    - Attributes: account_no, branch_city, IFSC_code, assests
    - Linked to Customer via foreign key (account_no)
- Payment:
    - Attributes: Reference_no, Payment_amount, Mode_of_payment, Transaction_id, Account_no, Date_of_purchase
    - Linked to Account via foreign key (Account_no)
- Transactions:
    - Attributes: TransactionID, Account_no, TransactionType, Amount, TransactionDate
    - Linked to Account via foreign key (Account_no)
- Merchant:
    - Attributes: merchant_id, user_id, transaction_id, phone_no, datetime, modepayment, amount
    - Linked to Customer and Payment via foreign keys (user_id, transaction_id)
- Insurance:
    - Attributes: insurance_id, customer_id, insurance_type,

    policy_number, start_date, end_date, premium_amount, coverage_details

    - Linked to Customer via foreign key (customer_id)

- BankToBank:
    - Attributes: bank_name, branch, amount, transaction_date, account_no
    - Linked to Account via foreign key (account_no)
- CreditCard:
    - Attributes: card_id, customer_id, card_number, expiration_date, card_holder_name, billing_address
    - Linked to Customer via foreign key (customer_id)
- DebitCard:
    - Attributes: customer_id, card_number, expiration_date, card_holder_name, amount
    - Linked to Customer via foreign key (customer_id)
- ATM_CARD:
    - Attributes: Account_no, AmountWithdrawal, Date_t, TransactionID
    - Linked to Transactions via foreign key (TransactionID)
- Loan:
    - Attributes: account_no, loan_type, loan_no, amount, interest
    - Linked to Customer via foreign key (account_no)
- BankSecurity:

    - Attributes: SecurityID, user_id, Username, Password, LastLoginTimestamp, PasswordResetToken, AccountLockStatus
    - Linked to Customer via foreign key (user_id)
- MobileBankingApps:
    - Attributes: AppID, AppName, MobilePayments, BiometricLogin, QRCodeTransactions, CustomerID
    - Linked to Customer via foreign key (CustomerID)
- GovernmentServices:
    - Attributes: ServiceID, ServiceName, ServiceDescription, CustomerID
- Linked to Customer via foreign key (CustomerID)

UNDERSTANDING THE ENTITY-RELATIONSHIP DIAGRAM

In a banking management system project, an Entity-Relationship Diagram (ERD) serves as a foundational blueprint for structuring and understanding the data model. The ERD visually depicts the various entities within the system, such as customers, accounts, transactions, branches, and employees, along with their relationships and attributes. By illustrating how these entities interact and relate to each other, the ERD provides a clear and concise representation of the system's data architecture. For instance, it elucidates that a customer can have multiple accounts, a branch can manage several accounts, and transactions are tied to specific accounts. Furthermore, the ERD outlines the attributes associated with each entity, aiding in the identification of key data elements and the design of database tables.

Moreover, the ERD plays a pivotal role in database design, ensuring data integrity and efficient data management. By defining the relationships between entities, such as one-to-one, one-to-many, or many-to-many, the ERD helps establish the rules for data manipulation and retrieval. For example, it specifies that an employee is assigned to a single branch while a branch can have multiple employees, guiding the implementation of foreign key constraints and referential integrity. Additionally, the ERD assists in identifying potential normalization issues and optimizing the database schema for performance and scalability. It enables developers and stakeholders to collaboratively visualize the data structure and make informed decisions regarding system functionality and user requirements. The ERD serves as a communication tool for project stakeholders, facilitating discussions and clarifications regarding the system's data model.

Chapter 4

SYSTEM REQUIREMENTS

1. **OPERATING SYSTEM**
   The BMS can be used on various operating systems, including Windows, macOS, and Linux. We can choose the one that we are most comfortable with. We recommend using Windows 10 or Windows 11.

2. **DEVELOPMENT ENVIRONMENT**
   We can use a variety of databases for creating a BMS, like MySQL, Oracle Database, etc. For our project we have chosen MySQL.

3. **HARDWARE**
   We don't need a high-end computer for this BMS. A basic desktop or laptop with at least 4GB of RAM and a modern multi-core processor should suffice.

4. **GRAPHICS**
   The BMS is not a very graphics-demanding system, so we don't need a powerful graphics card. Integrated graphics on most modern computers will be more than enough.

5. **STORAGE**
   We don't need much storage space for code and assets. A few gigabytes should be sufficient.

6. **REPORTS**
   To create reports and store data, we can use Microsoft Excel spreadsheets (.xlsx) and CSV files (Comma Separated Values, .csv).

Chapter 5

USE OF DESIGN THINKING APPROACH

1. **DESIGN THE PROBLEM and EMPATHISE**

   This represents a customer-centric approach that employs empathy, creativity, and rationality to solve complex problems. As banks face growing demands for personalised experiences and streamlined services, Design Thinking provides a methodology that fosters innovation

2. **RESEARCH, IDEATION and DEFINE**

   When comparing similar businesses, some of which use BMS, and others which don't, the businesses using BMS are "infrastructurally" better, efficient, and faster for both the owner and customers.

3. **PROTOTYPING**

   To make an SMS for this problem, we need to identify the stakeholders first. Stakeholders include the owner/manager, customers, and the staff. It brings the customer perspective into developing new services and improving experiences. It helps banks empathize with diverse customers, brainstorm creative ideas to meet their needs, and continuously iterate based on feedback.

4. **USER FEEDBACK**

   Design thinking in banking sector is a creative problem-solving approach that focuses on the customer's needs. In the context of banking, design thinking can be used to create financial products, services, and experiences that are more user-friendly and effective. In the long run, it can be seen how the business has been positively affected. Based on the owner's feedback, the BMS can be simplified and improved to better fit the owner's capabilities.

5. **IMPLEMENTATION**

   We must identify the best approach to make the BMS and its databases. Again, using ER Diagrams and Databases schemas can help. We have picked MySQL for our project.

6. **TESTING**

   Once the project is made, it must be tested in all possible cases and scenarios for debugging and improvements. Getting preliminary beta feedback for users and building on that is also helpful.

7. **DOCUMENTATION**

   Creating meaningful Reports, Presentations and README files to help users understand the BMS is crucial. Without understanding how something works, a user cannot obviously use the system properly.

8. **REFLECTION and ITERATION**

   Once again, gather feedback and iterate through possible cases to identify areas of improvement or errors. Adjust the system accordingly.

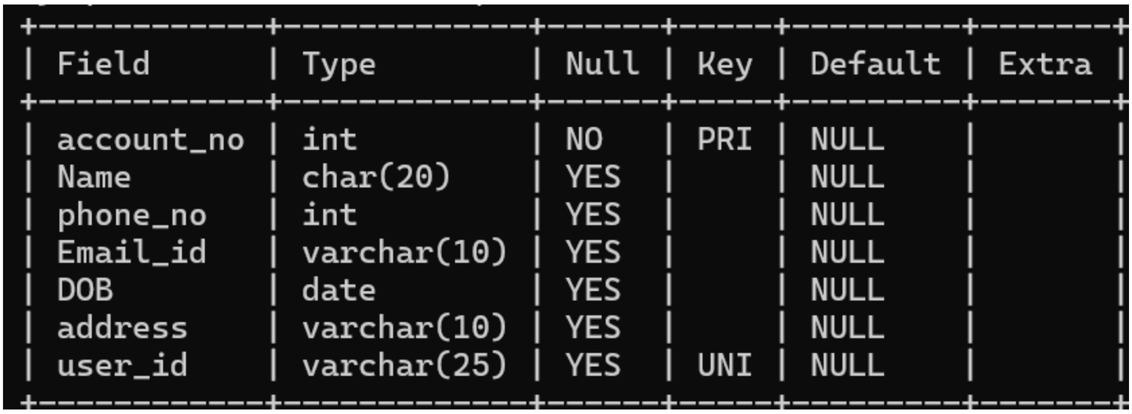
9. **FINAL PRESENTATION**

   To make this, we must reflect on every step that has come before this. We must highlight key features and designs in our final PPT.

Chapter 6

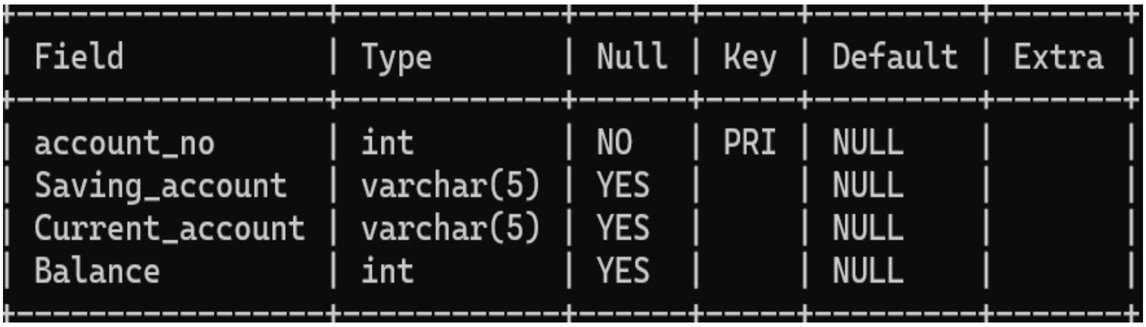
RELATIONAL TABLES AND SCHEMA

1. TABLE CUSTOMER



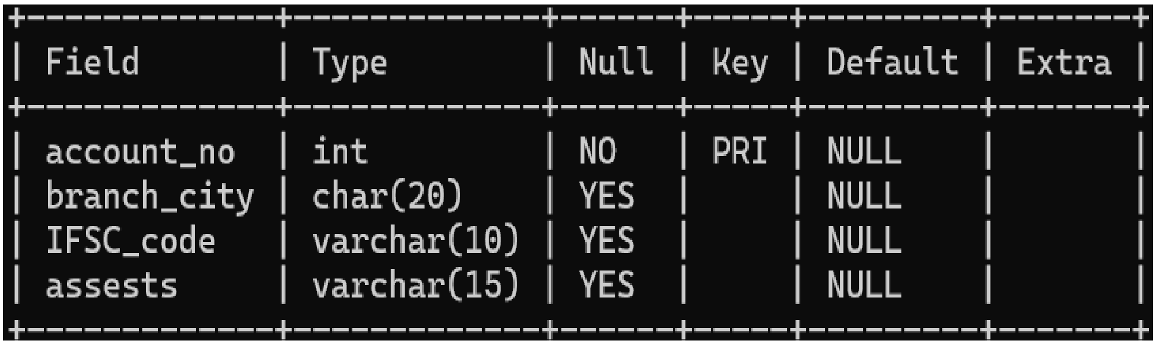
```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| account_no | int         | NO   | PRI | NULL    |       |
| Name       | char(20)    | YES  |     | NULL    |       |
| phone_no   | int         | YES  |     | NULL    |       |
| Email_id   | varchar(10) | YES  |     | NULL    |       |
| DOB        | date        | YES  |     | NULL    |       |
| address    | varchar(10) | YES  |     | NULL    |       |
| user_id    | varchar(25) | YES  | UNI | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

2. TABLE ACCOUNT



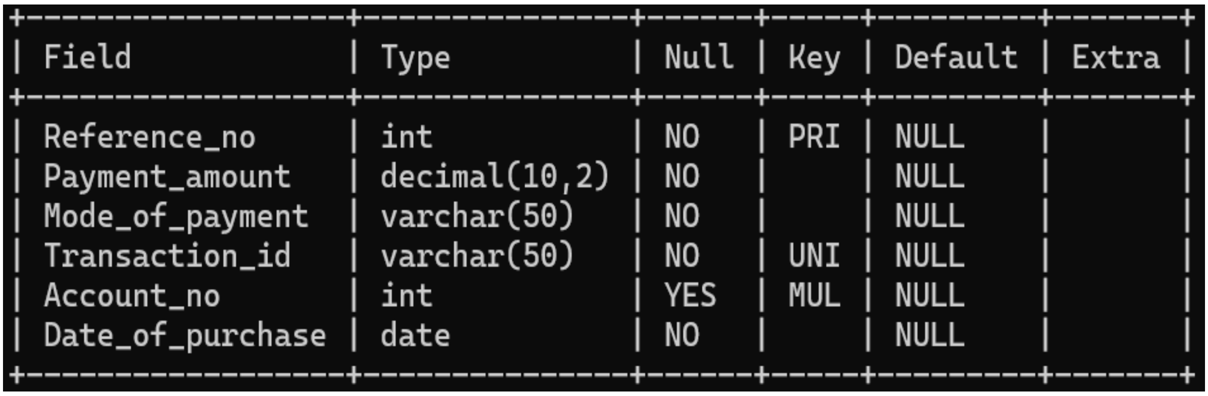
```
+-----------------+------------+------+-----+---------+-------+
| Field           | Type       | Null | Key | Default | Extra |
+-----------------+------------+------+-----+---------+-------+
| account_no      | int        | NO   | PRI | NULL    |       |
| Saving_account  | varchar(5) | YES  |     | NULL    |       |
| Current_account | varchar(5) | YES  |     | NULL    |       |
| Balance         | int        | YES  |     | NULL    |       |
+-----------------+------------+------+-----+---------+-------+
```

3. TABLE BRANCH



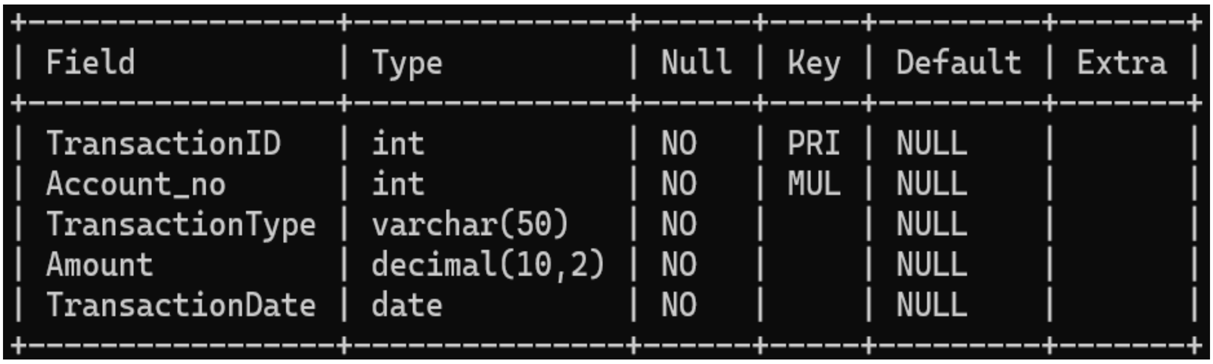
```
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| account_no  | int         | NO   | PRI | NULL    |       |
| branch_city | char(20)    | YES  |     | NULL    |       |
| IFSC_code   | varchar(10) | YES  |     | NULL    |       |
| assests     | varchar(15) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

4. TABLE PAYMENT



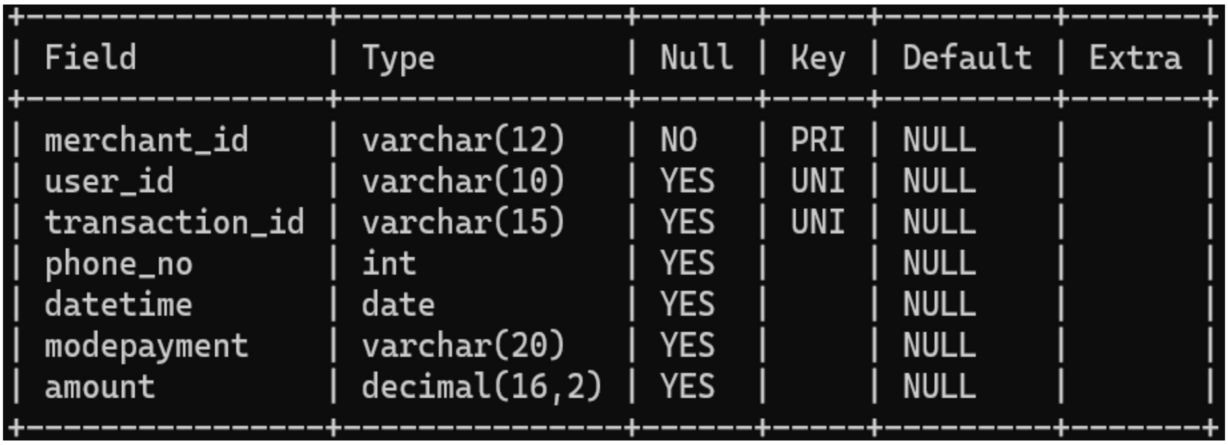
```
+-----------------+---------------+------+-----+---------+-------+
| Field           | Type          | Null | Key | Default | Extra |
+-----------------+---------------+------+-----+---------+-------+
| Reference_no    | int           | NO   | PRI | NULL    |       |
| Payment_amount  | decimal(10,2) | NO   |     | NULL    |       |
| Mode_of_payment | varchar(50)   | NO   |     | NULL    |       |
| Transaction_id  | varchar(50)   | NO   | UNI | NULL    |       |
| Account_no      | int           | YES  | MUL | NULL    |       |
| Date_of_purchase| date          | NO   |     | NULL    |       |
+-----------------+---------------+------+-----+---------+-------+
```

5. TABLE TRANSACTIONS



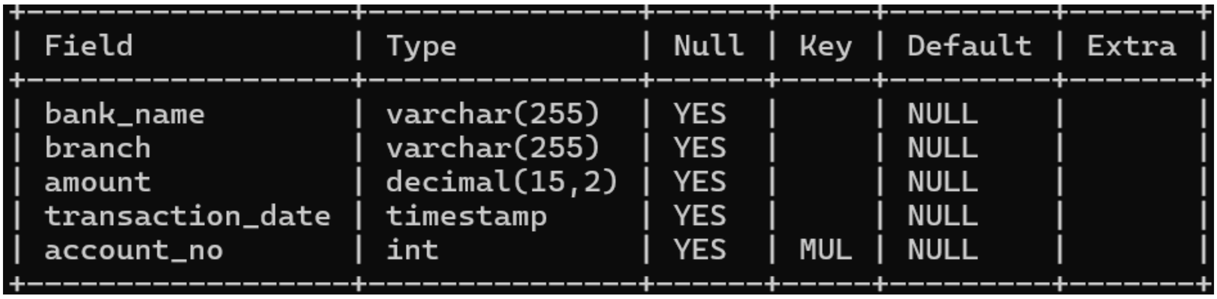
```
+-----------------+---------------+------+-----+---------+-------+
| Field           | Type          | Null | Key | Default | Extra |
+-----------------+---------------+------+-----+---------+-------+
| TransactionID   | int           | NO   | PRI | NULL    |       |
| Account_no      | int           | NO   | MUL | NULL    |       |
| TransactionType | varchar(50)   | NO   |     | NULL    |       |
| Amount          | decimal(10,2) | NO   |     | NULL    |       |
| TransactionDate | date          | NO   |     | NULL    |       |
+-----------------+---------------+------+-----+---------+-------+
```

6. TABLE MERCHANT



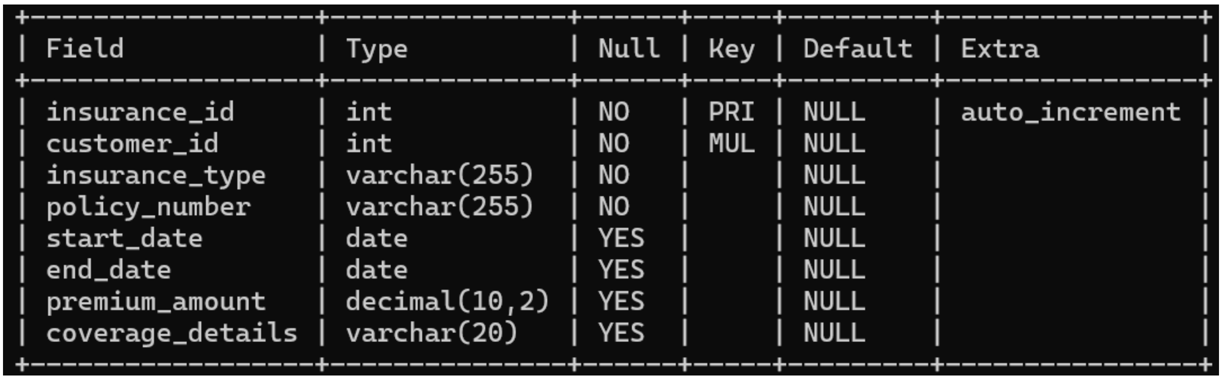
```
+----------------+---------------+------+-----+---------+-------+
| Field          | Type          | Null | Key | Default | Extra |
+----------------+---------------+------+-----+---------+-------+
| merchant_id    | varchar(12)   | NO   | PRI | NULL    |       |
| user_id        | varchar(10)   | YES  | UNI | NULL    |       |
| transaction_id | varchar(15)   | YES  | UNI | NULL    |       |
| phone_no       | int           | YES  |     | NULL    |       |
| datetime       | date          | YES  |     | NULL    |       |
| modepayment    | varchar(20)   | YES  |     | NULL    |       |
| amount         | decimal(16,2) | YES  |     | NULL    |       |
+----------------+---------------+------+-----+---------+-------+
```

7. TABLE BANKTOBANK



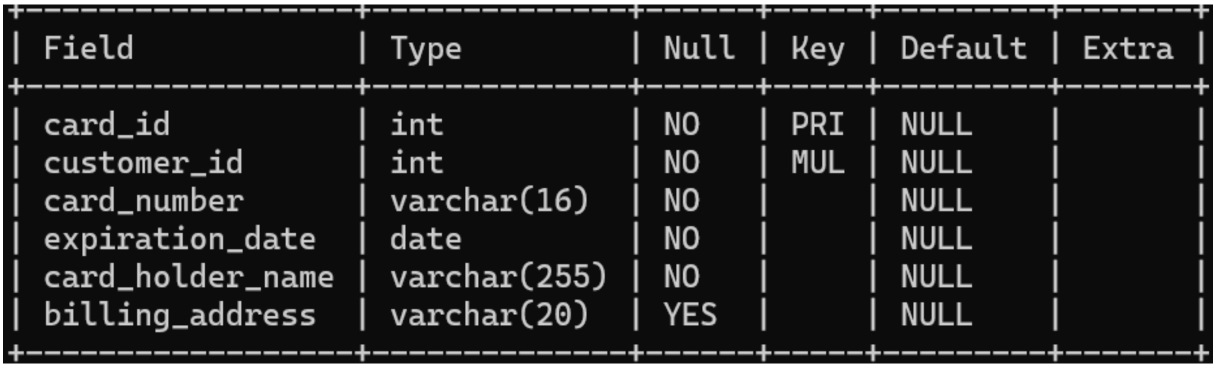
```
+------------------+--------------+------+-----+---------+-------+
| Field            | Type         | Null | Key | Default | Extra |
+------------------+--------------+------+-----+---------+-------+
| bank_name        | varchar(255) | YES  |     | NULL    |       |
| branch           | varchar(255) | YES  |     | NULL    |       |
| amount           | decimal(15,2)| YES  |     | NULL    |       |
| transaction_date | timestamp    | YES  |     | NULL    |       |
| account_no       | int          | YES  | MUL | NULL    |       |
+------------------+--------------+------+-----+---------+-------+
```

8. TABLE INSURANCE



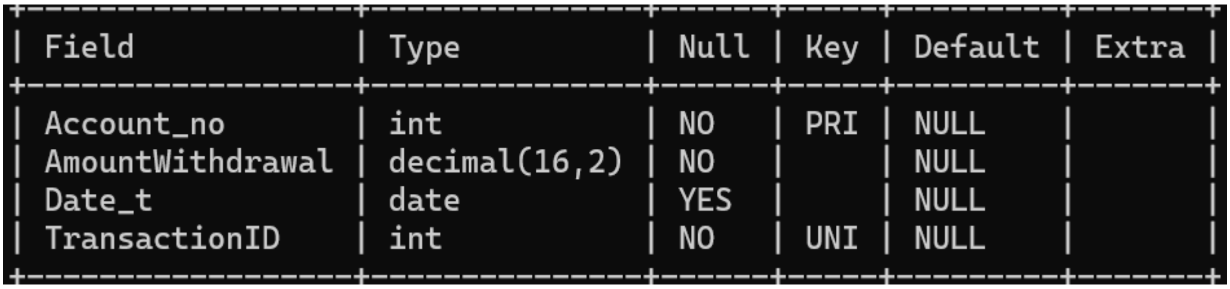
```
+------------------+---------------+------+-----+---------+----------------+
| Field            | Type          | Null | Key | Default | Extra          |
+------------------+---------------+------+-----+---------+----------------+
| insurance_id     | int           | NO   | PRI | NULL    | auto_increment |
| customer_id      | int           | NO   | MUL | NULL    |                |
| insurance_type   | varchar(255)  | NO   |     | NULL    |                |
| policy_number    | varchar(255)  | NO   |     | NULL    |                |
| start_date       | date          | YES  |     | NULL    |                |
| end_date         | date          | YES  |     | NULL    |                |
| premium_amount   | decimal(10,2) | YES  |     | NULL    |                |
| coverage_details | varchar(20)   | YES  |     | NULL    |                |
+------------------+---------------+------+-----+---------+----------------+
```

9. TABLE CREDITCARD



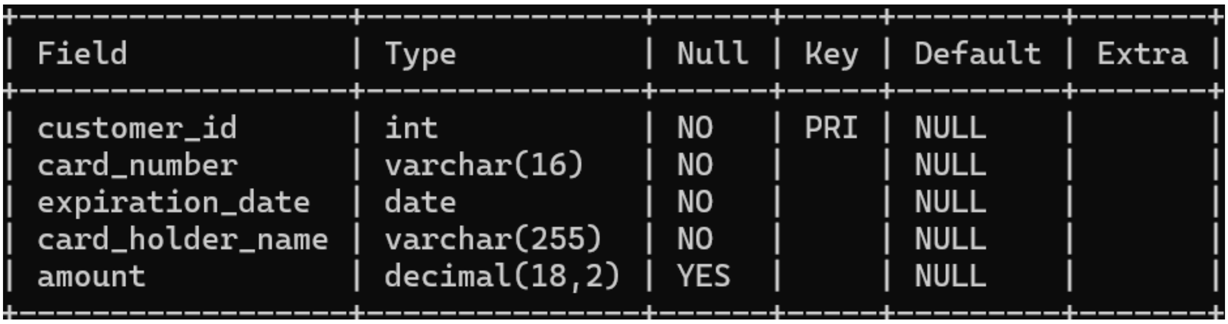
```
+------------------+--------------+------+-----+---------+-------+
| Field            | Type         | Null | Key | Default | Extra |
+------------------+--------------+------+-----+---------+-------+
| card_id          | int          | NO   | PRI | NULL    |       |
| customer_id      | int          | NO   | MUL | NULL    |       |
| card_number      | varchar(16)  | NO   |     | NULL    |       |
| expiration_date  | date         | NO   |     | NULL    |       |
| card_holder_name | varchar(255) | NO   |     | NULL    |       |
| billing_address  | varchar(20)  | YES  |     | NULL    |       |
+------------------+--------------+------+-----+---------+-------+
```

10.TABLE ATM_CARD



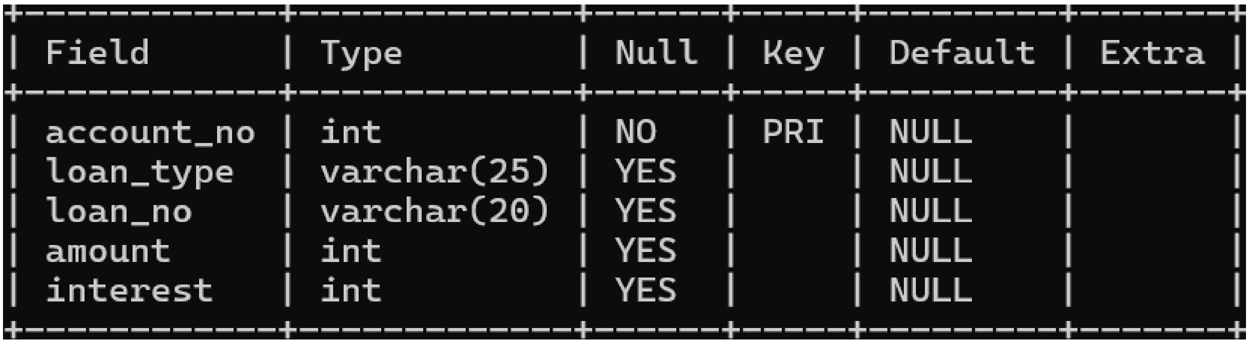
```
+------------------+---------------+------+-----+---------+-------+
| Field            | Type          | Null | Key | Default | Extra |
+------------------+---------------+------+-----+---------+-------+
| Account_no       | int           | NO   | PRI | NULL    |       |
| AmountWithdrawal | decimal(16,2) | NO   |     | NULL    |       |
| Date_t           | date          | YES  |     | NULL    |       |
| TransactionID    | int           | NO   | UNI | NULL    |       |
+------------------+---------------+------+-----+---------+-------+
```

11.TABLE DEBITCARD



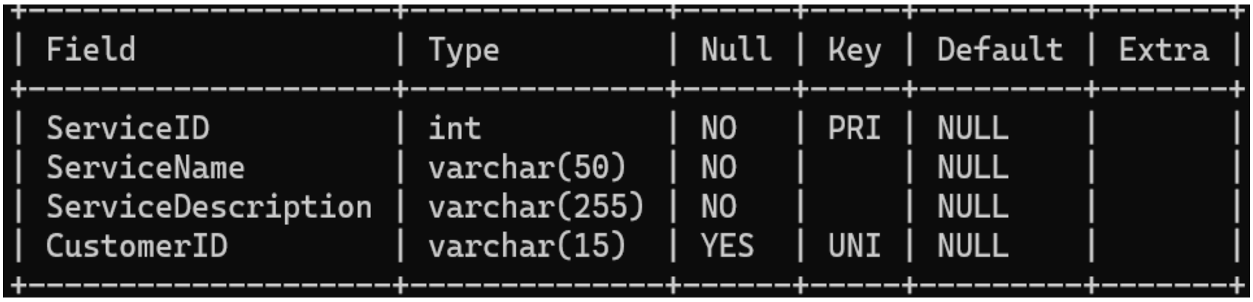
```
+------------------+---------------+------+-----+---------+-------+
| Field            | Type          | Null | Key | Default | Extra |
+------------------+---------------+------+-----+---------+-------+
| customer_id      | int           | NO   | PRI | NULL    |       |
| card_number      | varchar(16)   | NO   |     | NULL    |       |
| expiration_date  | date          | NO   |     | NULL    |       |
| card_holder_name | varchar(255)  | NO   |     | NULL    |       |
| amount           | decimal(18,2) | YES  |     | NULL    |       |
+------------------+---------------+------+-----+---------+-------+
```

12.TABLE LOAN



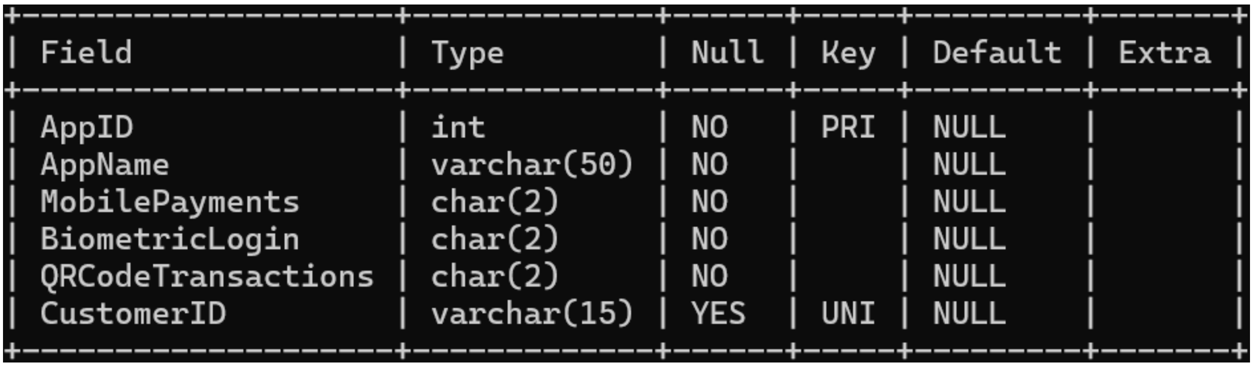
```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| account_no | int         | NO   | PRI | NULL    |       |
| loan_type  | varchar(25) | YES  |     | NULL    |       |
| loan_no    | varchar(20) | YES  |     | NULL    |       |
| amount     | int         | YES  |     | NULL    |       |
| interest   | int         | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

13.TABLE GOVERNMENTSERVICE



```
+--------------------+--------------+------+-----+---------+-------+
| Field              | Type         | Null | Key | Default | Extra |
+--------------------+--------------+------+-----+---------+-------+
| ServiceID          | int          | NO   | PRI | NULL    |       |
| ServiceName        | varchar(50)  | NO   |     | NULL    |       |
| ServiceDescription | varchar(255) | NO   |     | NULL    |       |
| CustomerID         | varchar(15)  | YES  | UNI | NULL    |       |
+--------------------+--------------+------+-----+---------+-------+
```

14.TABLE MOBILEBANKINGAPP



```
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| AppID              | int         | NO   | PRI | NULL    |       |
| AppName            | varchar(50) | NO   |     | NULL    |       |
| MobilePayments     | char(2)     | NO   |     | NULL    |       |
| BiometricLogin     | char(2)     | NO   |     | NULL    |       |
| QRCodeTransactions | char(2)     | NO   |     | NULL    |       |
| CustomerID         | varchar(15) | YES  | UNI | NULL    |       |
+--------------------+-------------+------+-----+---------+-------+
```

TABLE QUERY

TABLE CUSTOMER :

```
create table customer(
account_no int primary key,
Name char(20),
phone_no int,Email_id varchar(10)
,DOB date,
address varchar(10),
user_id varchar(25) unique);
```

TABLE ACCOUNT :

```
CREATE TABLE account(
account_no INT,
Saving_account VARCHAR(5),
Current_account VARCHAR(5),
Balance INT,
PRIMARY KEY (account_no),
FOREIGN KEY (account_no) REFERENCES customer(account_no));
```

TABLE BRANCH :

```
CREATE TABLE branch (
account_no INT,
branch_city CHAR(20),

IFSC_code VARCHAR(10),

assests VARCHAR(15),

PRIMARY KEY (account_no),

FOREIGN KEY (account_no) REFERENCES customer(account_no)

);
```

TABLE PAYMENT :

```
CREATE TABLE Payment (

Reference_no int PRIMARY KEY,

Payment_amount DECIMAL(10, 2) NOT NULL,

Mode_of_payment VARCHAR(50) NOT NULL,

Transaction_id VARCHAR(50) NOT NULL UNIQUE,

Account_no int,

Date_of_purchase DATE NOT NULL,

FOREIGN KEY (Account_no) REFERENCES Account(Account_no)

);
```

TABLE TRANSACTIONS :

```
CREATE TABLE Transactions (

TransactionID int PRIMARY KEY,

Account_no int NOT NULL,
TransactionType VARCHAR(50) NOT NULL,
Amount DECIMAL(10, 2) NOT NULL,
TransactionDate DATE NOT NULL,
FOREIGN KEY (Account_no) REFERENCES Account(Account_no)
);
```

TABLE MERCHANT :

```
create table merchant(
merchant_id varchar(12) primary key,
user_id varchar(10) unique,
transaction_id varchar(15) unique,
phone_no int(10),
datetime date,
modepayment varchar(20),
amount decimal(16,2),
FOREIGN KEY (user_id) REFERENCES Customer(user_id),
FOREIGN KEY (transaction_id) REFERENCES payment(transaction_id));
```

TABLE INSURANCE :

```
CREATE TABLE Insurance (
insurance_id INT PRIMARY KEY AUTO_INCREMENT,
customer_id INT NOT NULL,
insurance_type VARCHAR(255) NOT NULL,
policy_number VARCHAR(255) NOT NULL,
start_date DATE,
end_date DATE,
premium_amount DECIMAL(10,2),
coverage_details varchar(20),
FOREIGN KEY (customer_id) REFERENCES customer(account_no)
);
```

TABLE BANKTOBANK :

```
create table banktobank(
bank_name varchar(255),
branch varchar(255),
amount decimal(15,2),
transaction_date timestamp,
account_no int,
FOREIGN KEY (account_no) REFERENCES account(account_no));
```

TABLE CREDITCARD :

```
CREATE TABLE CreditCard (
card_id INT PRIMARY KEY,
customer_id INT NOT NULL,
card_number VARCHAR(16) NOT NULL,
expiration_date DATE NOT NULL
card_holder_name VARCHAR(255) NOT NULL,
billing_address varchar(20)
FOREIGN KEY (customer_id) REFERENCES customer(account_no)
);
```

TABLE DEBITCARD :

```
CREATE TABLE DebitCard (
customer_id INT primary key,
card_number VARCHAR(16) NOT NULL,
expiration_date DATE NOT NULL,
card_holder_name VARCHAR(255) NOT NULL,
amount decimal(18,2),
FOREIGN KEY (customer_id) REFERENCES customer(account_no)
);
```

TABLE ATM_CARD:

```
CREATE TABLE ATM_CARD (
Account_no int PRIMARY KEY,
AmountWithdrawal DECIMAL(16, 2) NOT NULL,
Date_t DATE ,
TransactionID int UNIQUE NOT NULL,

FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID)

);
```

TABLE LOAN:

```
CREATE TABLE loan(

account_no int,

loan_type int,

loan_no varchar(20),

amount int,

interest int,

PRIMARY KEY (account_no),

FOREIGN KEY (account_no) REFERENCES customer(account_no));
```

TABLE BANK SECURITY :

```
CREATE TABLE BankSecurity (

SecurityID int PRIMARY KEY,

user_id varchar(15) UNIQUE NOT NULL,

Username VARCHAR(50) NOT NULL,

Password VARCHAR(255) NOT NULL,

LastLoginTimestamp TIMESTAMP NOT NULL,

PasswordResetToken VARCHAR(255),
AccountLockStatus VARCHAR(50) NOT NULL,

FOREIGN KEY (user_id) REFERENCES Customer1(user_id)

);
```

TABLE MobileBankingApps :

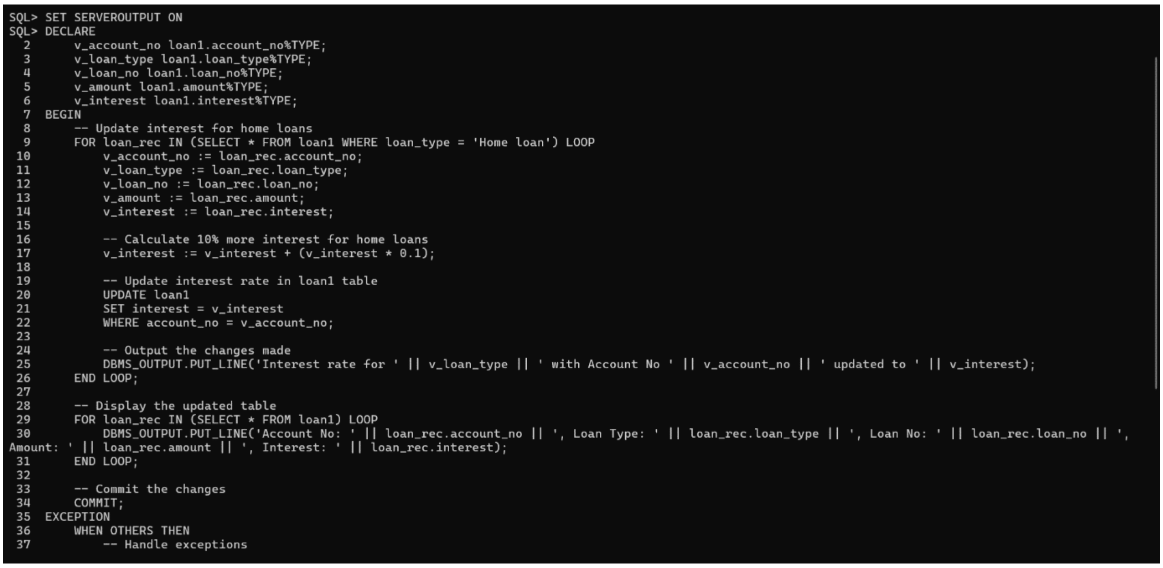
```
CREATE TABLE MobileBankingApps (

AppID int PRIMARY KEY,

AppName VARCHAR(50) NOT NULL,

MobilePayments char(2) NOT NULL,

BiometricLogin char(2) NOT NULL,

QRCodeTransactions char(2) NOT NULL,

CustomerID varchar(15) unique,

FOREIGN KEY (CustomerID) REFERENCES Customer(user_id)

);
```

Chapter 7

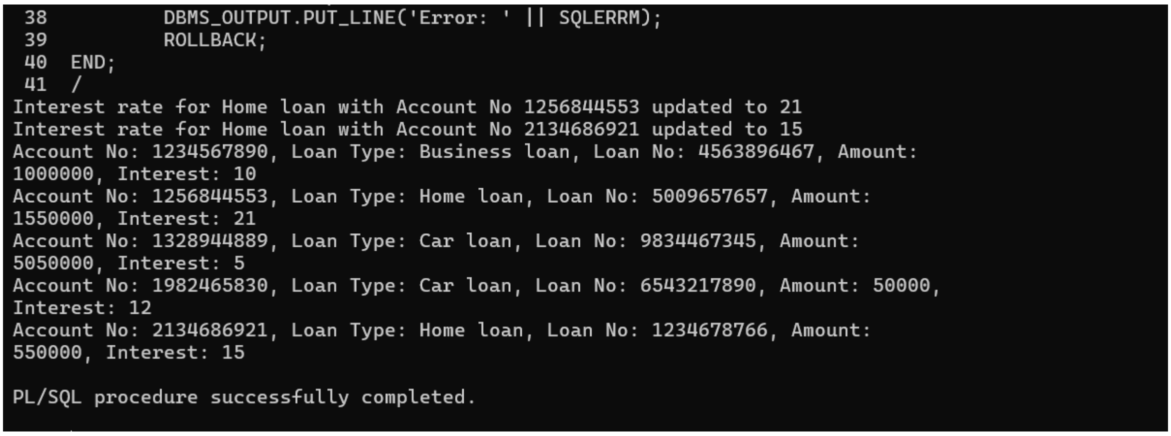
COMPLEX QUERIES

Q1) Write a PL/SQL block to add 10% more interest on loan type = home loan

ANSWER:



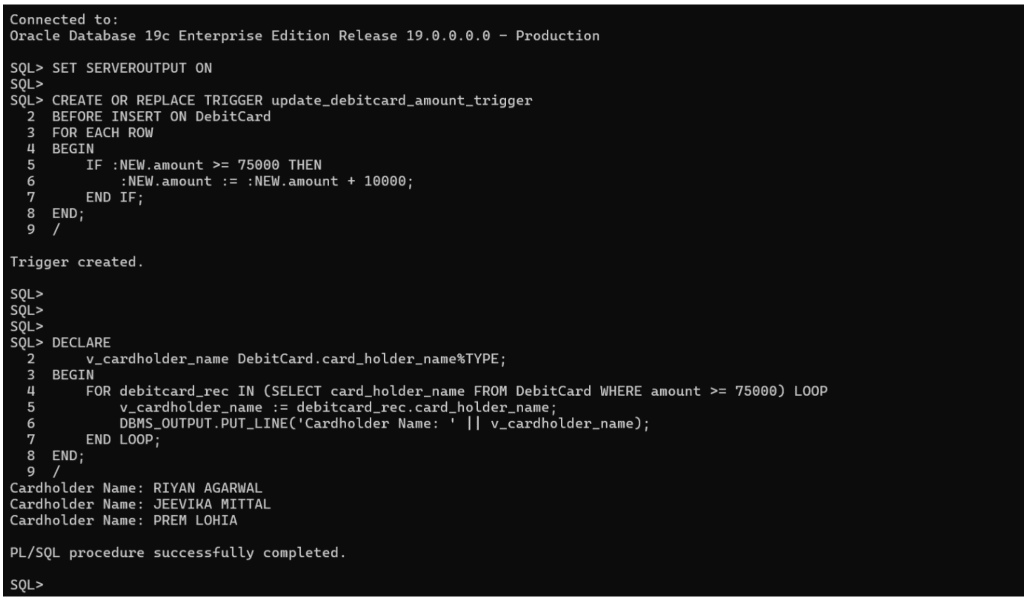
```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
         v_account_no loan1.account_no%TYPE;
         v_loan_type loan1.loan_type%TYPE;
         v_loan_no loan1.loan_no%TYPE;
         v_amount loan1.amount%TYPE;
         v_interest loan1.interest%TYPE;
     BEGIN
         -- Update interest for home loans
         FOR loan_rec IN (SELECT * FROM loan1 WHERE loan_type = 'Home loan') LOOP
             v_account_no := loan_rec.account_no;
             v_loan_type := loan_rec.loan_type;
             v_loan_no := loan_rec.loan_no;
             v_amount := loan_rec.amount;
             v_interest := loan_rec.interest;

             -- Calculate 10% more interest for home loans
             v_interest := v_interest + (v_interest * 0.1);

             -- Update interest rate in loan1 table
             UPDATE loan1
             SET interest = v_interest
             WHERE account_no = v_account_no;

             -- Output the changes made
             DBMS_OUTPUT.PUT_LINE('Interest rate for ' || v_loan_type || ' with Account No ' || v_account_no || ' updated to ' || v_interest);
         END LOOP;

         -- Display the updated table
         FOR loan_rec IN (SELECT * FROM loan1) LOOP
             DBMS_OUTPUT.PUT_LINE('Account No: ' || loan_rec.account_no || ', Loan Type: ' || loan_rec.loan_type || ', Loan No: ' || loan_rec.loan_no || ',
Amount: ' || loan_rec.amount || ', Interest: ' || loan_rec.interest);
         END LOOP;

         -- Commit the changes
         COMMIT;
     EXCEPTION
         WHEN OTHERS THEN
             -- Handle exceptions
```



```
             DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
             ROLLBACK;
     END;
     /
Interest rate for Home loan with Account No 1256844553 updated to 21
Interest rate for Home loan with Account No 2134686921 updated to 15
Account No: 1234567890, Loan Type: Business loan, Loan No: 4563896467, Amount:
1000000, Interest: 10
Account No: 1256844553, Loan Type: Home loan, Loan No: 5009657657, Amount:
1550000, Interest: 21
Account No: 1328944889, Loan Type: Car loan, Loan No: 9834467345, Amount:
5050000, Interest: 5
Account No: 1982465830, Loan Type: Car loan, Loan No: 6543217890, Amount: 50000,
Interest: 12
Account No: 2134686921, Loan Type: Home loan, Loan No: 1234678766, Amount:
550000, Interest: 15

PL/SQL procedure successfully completed.
```

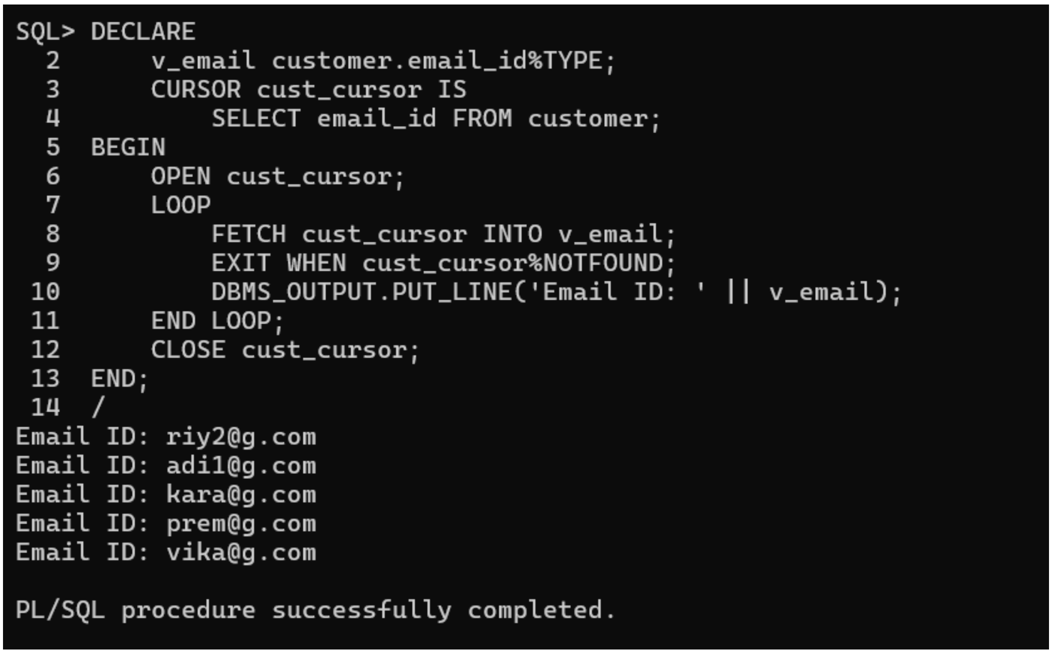
Q2) Write a before trigger to add 10000 ruppee to customer table debit card whose amount is more than 75000

ANSWER: BEFORE INSERT trigger



```
Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production

SQL> SET SERVEROUTPUT ON
SQL>
SQL> CREATE OR REPLACE TRIGGER update_debitcard_amount_trigger
  2  BEFORE INSERT ON DebitCard
  3  FOR EACH ROW
  4  BEGIN
  5      IF :NEW.amount >= 75000 THEN
  6          :NEW.amount := :NEW.amount + 10000;
  7      END IF;
  8  END;
  9  /

Trigger created.

SQL>
SQL>
SQL>
SQL> DECLARE
  2      v_cardholder_name DebitCard.card_holder_name%TYPE;
  3  BEGIN
  4      FOR debitcard_rec IN (SELECT card_holder_name FROM DebitCard WHERE amount >= 75000) LOOP
  5          v_cardholder_name := debitcard_rec.card_holder_name;
  6          DBMS_OUTPUT.PUT_LINE('Cardholder Name: ' || v_cardholder_name);
  7      END LOOP;
  8  END;
  9  /
Cardholder Name: RIYAN AGARWAL
Cardholder Name: JEEVIKA MITTAL
Cardholder Name: PREM LOHIA

PL/SQL procedure successfully completed.

SQL>
```

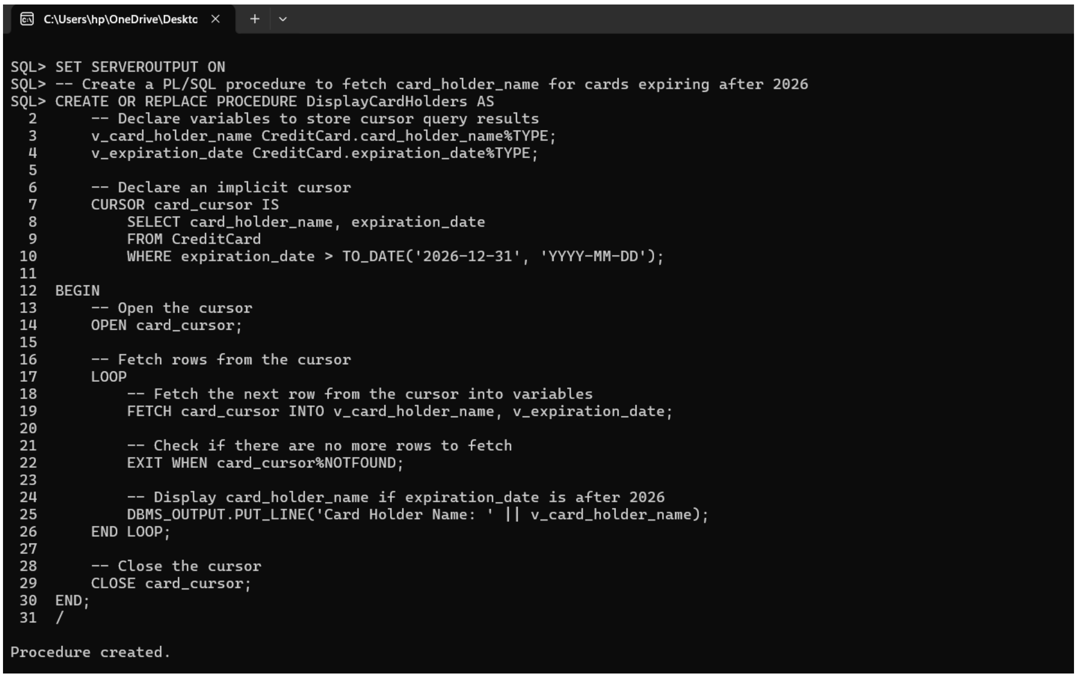
Q3) Write a PL/SQL code of explicit cursor to display all the email_id af all customer name from table customer



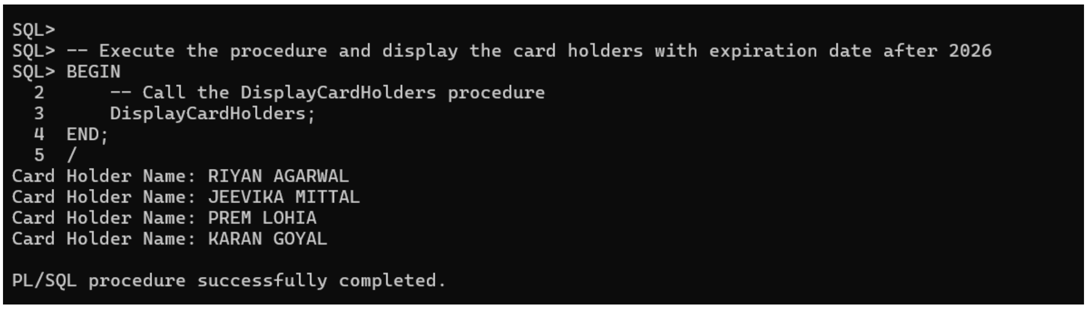
```
SQL> DECLARE
  2      v_email customer.email_id%TYPE;
  3      CURSOR cust_cursor IS
  4          SELECT email_id FROM customer;
  5  BEGIN
  6      OPEN cust_cursor;
  7      LOOP
  8          FETCH cust_cursor INTO v_email;
  9          EXIT WHEN cust_cursor%NOTFOUND;
 10          DBMS_OUTPUT.PUT_LINE('Email ID: ' || v_email);
 11      END LOOP;
 12      CLOSE cust_cursor;
 13  END;
 14  /
Email ID: riy2@g.com
Email ID: adi1@g.com
Email ID: kara@g.com
Email ID: prem@g.com
Email ID: vika@g.com

PL/SQL procedure successfully completed.
```

Q4) Write a PL/SQL code of implicit cursor to display the card_holder_name of those people who have expiration_date after year 2026 and also display result as an output

ANSWER:



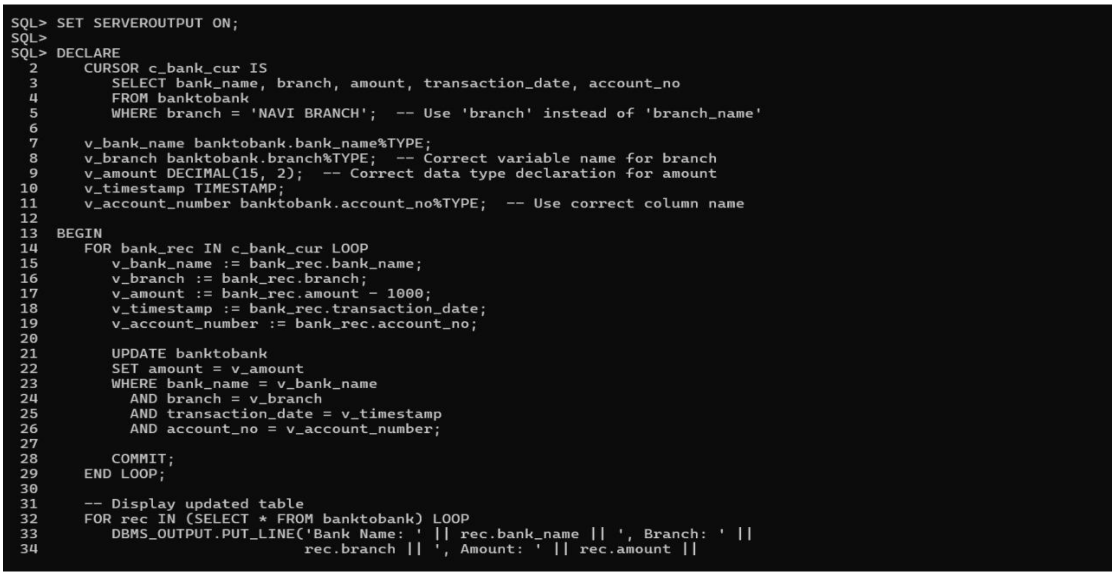
```
SQL> SET SERVEROUTPUT ON
SQL> -- Create a PL/SQL procedure to fetch card_holder_name for cards expiring after 2026
SQL> CREATE OR REPLACE PROCEDURE DisplayCardHolders AS
         -- Declare variables to store cursor query results
         v_card_holder_name CreditCard.card_holder_name%TYPE;
         v_expiration_date CreditCard.expiration_date%TYPE;

         -- Declare an implicit cursor
         CURSOR card_cursor IS
             SELECT card_holder_name, expiration_date
             FROM CreditCard
             WHERE expiration_date > TO_DATE('2026-12-31', 'YYYY-MM-DD');

     BEGIN
         -- Open the cursor
         OPEN card_cursor;

         -- Fetch rows from the cursor
         LOOP
             -- Fetch the next row from the cursor into variables
             FETCH card_cursor INTO v_card_holder_name, v_expiration_date;

             -- Check if there are no more rows to fetch
             EXIT WHEN card_cursor%NOTFOUND;

             -- Display card_holder_name if expiration_date is after 2026
             DBMS_OUTPUT.PUT_LINE('Card Holder Name: ' || v_card_holder_name);
         END LOOP;

         -- Close the cursor
         CLOSE card_cursor;
     END;
     /

Procedure created.
```



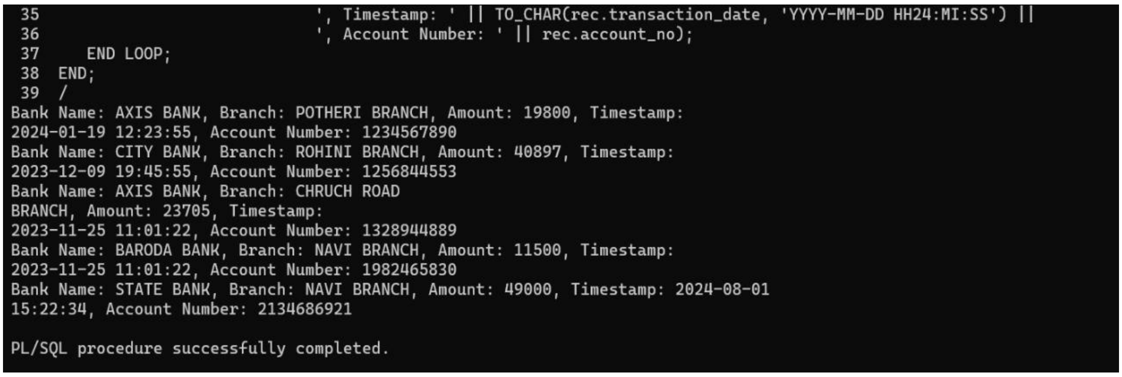
```
SQL>
SQL> -- Execute the procedure and display the card holders with expiration date after 2026
SQL> BEGIN
         -- Call the DisplayCardHolders procedure
         DisplayCardHolders;
     END;
     /
Card Holder Name: RIYAN AGARWAL
Card Holder Name: JEEVIKA MITTAL
Card Holder Name: PREM LOHIA
Card Holder Name: KARAN GOYAL

PL/SQL procedure successfully completed.
```

Q1) Write a normal PL/SQL block to change the amount from branch name = navi branch from BanktoBank x table and decrease amount to 1000 from all columns of the table using cursor

Answer:



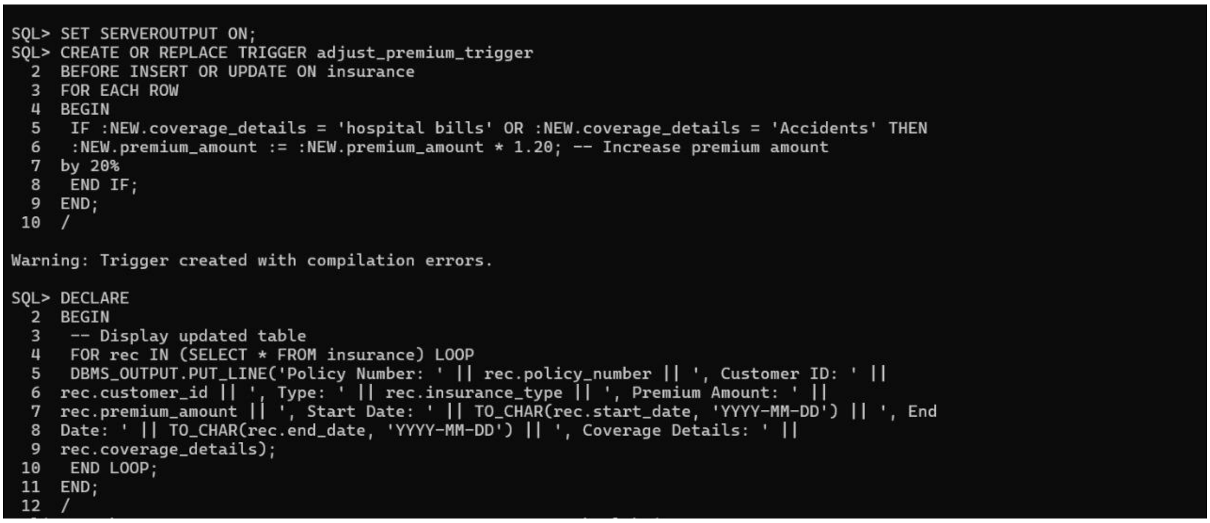
```
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
        CURSOR c_bank_cur IS
           SELECT bank_name, branch, amount, transaction_date, account_no
           FROM banktobank
           WHERE branch = 'NAVI BRANCH';  -- Use 'branch' instead of 'branch_name'

        v_bank_name banktobank.bank_name%TYPE;
        v_branch banktobank.branch%TYPE;  -- Correct variable name for branch
        v_amount DECIMAL(15, 2);  -- Correct data type declaration for amount
        v_timestamp TIMESTAMP;
        v_account_number banktobank.account_no%TYPE;  -- Use correct column name

     BEGIN
        FOR bank_rec IN c_bank_cur LOOP
           v_bank_name := bank_rec.bank_name;
           v_branch := bank_rec.branch;
           v_amount := bank_rec.amount - 1000;
           v_timestamp := bank_rec.transaction_date;
           v_account_number := bank_rec.account_no;

           UPDATE banktobank
           SET amount = v_amount
           WHERE bank_name = v_bank_name
             AND branch = v_branch
             AND transaction_date = v_timestamp
             AND account_no = v_account_number;

           COMMIT;
        END LOOP;

        -- Display updated table
        FOR rec IN (SELECT * FROM banktobank) LOOP
           DBMS_OUTPUT.PUT_LINE('Bank Name: ' || rec.bank_name || ', Branch: ' ||
                                rec.branch || ', Amount: ' || rec.amount ||
```



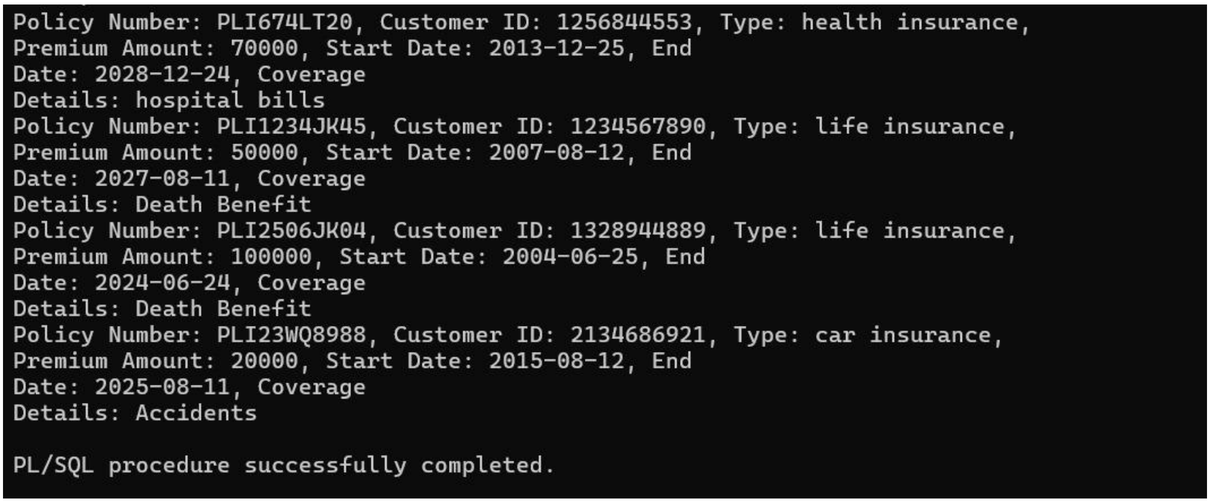
```
                                ', Timestamp: ' || TO_CHAR(rec.transaction_date, 'YYYY-MM-DD HH24:MI:SS') ||
                                ', Account Number: ' || rec.account_no);
        END LOOP;
     END;
     /
Bank Name: AXIS BANK, Branch: POTHERI BRANCH, Amount: 19800, Timestamp:
2024-01-19 12:23:55, Account Number: 1234567890
Bank Name: CITY BANK, Branch: ROHINI BRANCH, Amount: 40897, Timestamp:
2023-12-09 19:45:55, Account Number: 1256844553
Bank Name: AXIS BANK, Branch: CHRUCH ROAD
BRANCH, Amount: 23705, Timestamp:
2023-11-25 11:01:22, Account Number: 1328944889
Bank Name: BARODA BANK, Branch: NAVI BRANCH, Amount: 11500, Timestamp:
2023-11-25 11:01:22, Account Number: 1982465830
Bank Name: STATE BANK, Branch: NAVI BRANCH, Amount: 49000, Timestamp: 2024-08-01
15:22:34, Account Number: 2134686921

PL/SQL procedure successfully completed.
```

Q2) Write a PL/SQL block to change the insurancex table, premium_amount who have coverage_details of Hospital bills , Accident bills will increase premium amount by 20 percent more by using trigger ?

ANSWER:



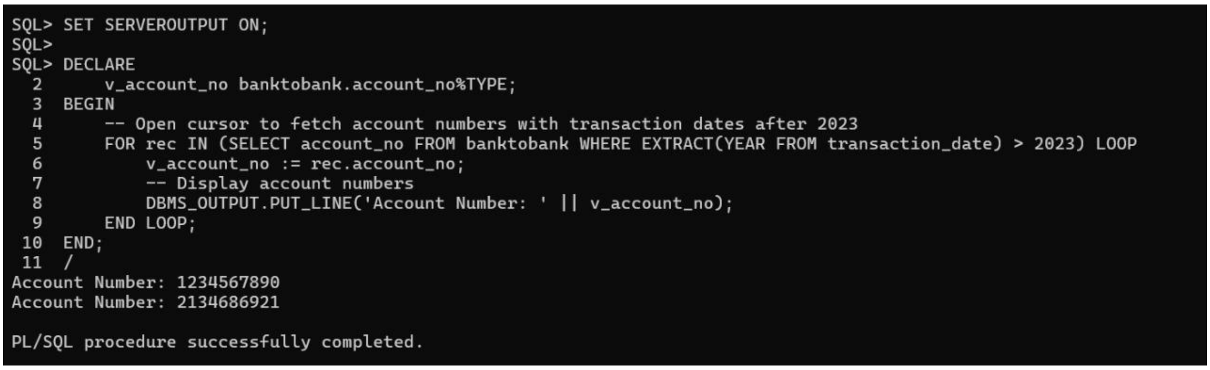
```
SQL> SET SERVEROUTPUT ON;
SQL> CREATE OR REPLACE TRIGGER adjust_premium_trigger
     BEFORE INSERT OR UPDATE ON insurance
     FOR EACH ROW
     BEGIN
      IF :NEW.coverage_details = 'hospital bills' OR :NEW.coverage_details = 'Accidents' THEN
      :NEW.premium_amount := :NEW.premium_amount * 1.20; -- Increase premium amount
     by 20%
      END IF;
     END;
     /

Warning: Trigger created with compilation errors.

SQL> DECLARE
     BEGIN
      -- Display updated table
      FOR rec IN (SELECT * FROM insurance) LOOP
      DBMS_OUTPUT.PUT_LINE('Policy Number: ' || rec.policy_number || ', Customer ID: ' ||
     rec.customer_id || ', Type: ' || rec.insurance_type || ', Premium Amount: ' ||
     rec.premium_amount || ', Start Date: ' || TO_CHAR(rec.start_date, 'YYYY-MM-DD') || ', End
     Date: ' || TO_CHAR(rec.end_date, 'YYYY-MM-DD') || ', Coverage Details: ' ||
     rec.coverage_details);
      END LOOP;
     END;
     /
```



```
Policy Number: PLI674LT20, Customer ID: 1256844553, Type: health insurance,
Premium Amount: 70000, Start Date: 2013-12-25, End
Date: 2028-12-24, Coverage
Details: hospital bills
Policy Number: PLI1234JK45, Customer ID: 1234567890, Type: life insurance,
Premium Amount: 50000, Start Date: 2007-08-12, End
Date: 2027-08-11, Coverage
Details: Death Benefit
Policy Number: PLI2506JK04, Customer ID: 1328944889, Type: life insurance,
Premium Amount: 100000, Start Date: 2004-06-25, End
Date: 2024-06-24, Coverage
Details: Death Benefit
Policy Number: PLI23WQ8988, Customer ID: 2134686921, Type: car insurance,
Premium Amount: 20000, Start Date: 2015-08-12, End
Date: 2025-08-11, Coverage
Details: Accidents

PL/SQL procedure successfully completed.
```

Q3) Write a PL/SQL code of implicit cursor to display the account_no of those who have transaction_date after year 2023?

ANSWER:



```
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
         v_account_no banktobank.account_no%TYPE;
     BEGIN
         -- Open cursor to fetch account numbers with transaction dates after 2023
         FOR rec IN (SELECT account_no FROM banktobank WHERE EXTRACT(YEAR FROM transaction_date) > 2023) LOOP
             v_account_no := rec.account_no;
             -- Display account numbers
             DBMS_OUTPUT.PUT_LINE('Account Number: ' || v_account_no);
         END LOOP;
     END;
     /
Account Number: 1234567890
Account Number: 2134686921

PL/SQL procedure successfully completed.
```

Chapter 8

PITFALLS, FUNCTIONAL DEPENDENCIES AND NORMALIZATION

TABLE CUSTOMER:

```
create table customer( account_no int primary key, Name char(20), phone_no int, Email_id varchar(10),DOB date, address varchar(10), user_id varchar(25) unique);
```

Functional Dependencies:

account_no → Name, phone_no, Email_id, DOB, address, user_id

Normalization Form:

While the customer table meets the criteria for First Normal Form (1NF) and Second Normal Form (2NF), it does not fully satisfy Third Normal Form (3NF) due to potential transitive dependencies.

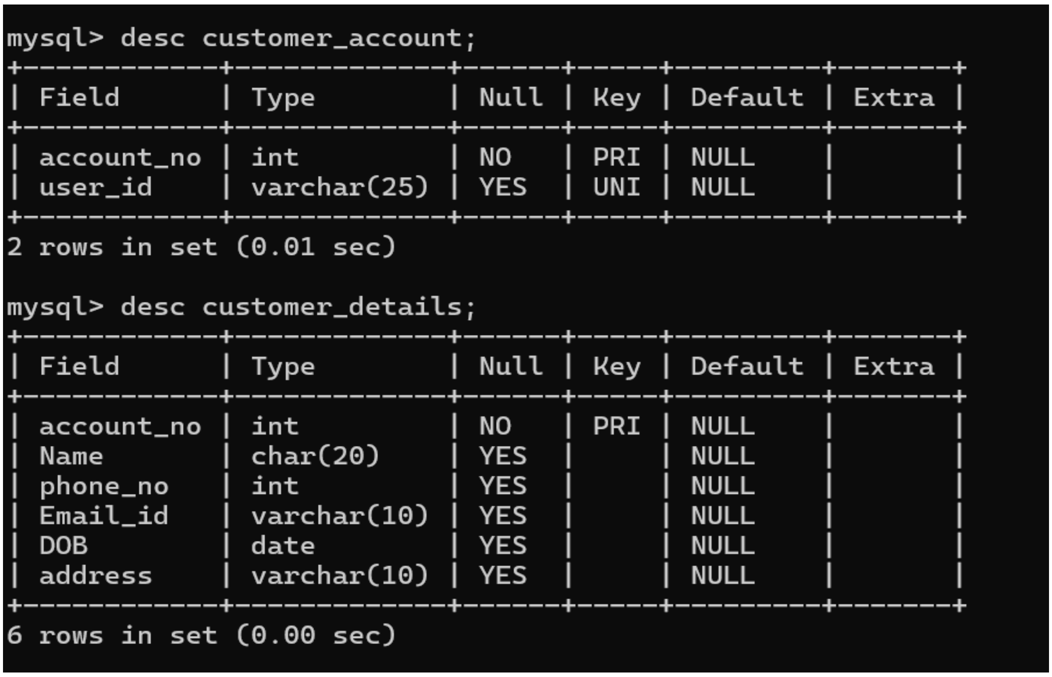
Decomposed Table (SQL*Plus):

```
CREATE TABLE Customer_Details (
account_no INT PRIMARY KEY,
Name CHAR(20),
phone_no INT,
Email_id VARCHAR(10),
DOB DATE,
address VARCHAR(10)
);

CREATE TABLE Customer_Account (
account_no INT PRIMARY KEY,
user_id VARCHAR(25) UNIQUE,
FOREIGN KEY (account_no) REFERENCES Customer_Details(account_no)
);
```

Pitfalls:

Partial Dependencies: Columns such as phone_no, Email_id, address might depend on the primary key account_no rather than being fully dependent on it.

For instance, if phone_no changes, it might require updating multiple rows in the table.



```
mysql> desc customer_account;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| account_no | int         | NO   | PRI | NULL    |       |
| user_id    | varchar(25) | YES  | UNI | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> desc customer_details;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| account_no | int         | NO   | PRI | NULL    |       |
| Name       | char(20)    | YES  |     | NULL    |       |
| phone_no   | int         | YES  |     | NULL    |       |
| Email_id   | varchar(10) | YES  |     | NULL    |       |
| DOB        | date        | YES  |     | NULL    |       |
| address    | varchar(10) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

TABLE PAYMENT:

```
CREATE TABLE Payment (
Reference_no int PRIMARY KEY,
Payment_amount DECIMAL(10, 2) NOT NULL,
Mode_of_payment VARCHAR(50) NOT NULL,
Transaction_id VARCHAR(50) NOT NULL UNIQUE,
Account_no int,
Date_of_purchase DATE NOT NULL,
FOREIGN KEY (Account_no) REFERENCES Account(Account_no)
);
```

Functional Dependencies:

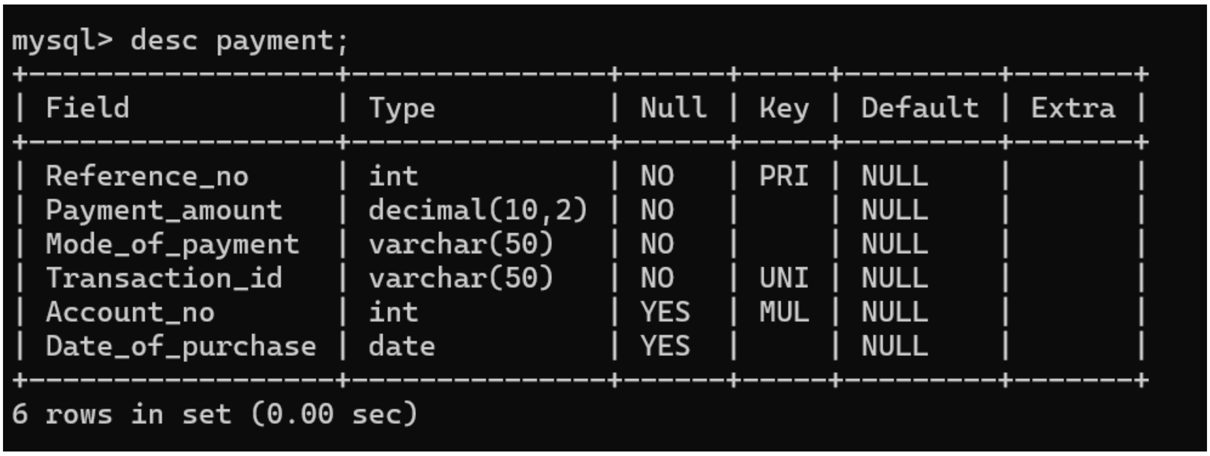
Reference_no → Payment_amount, Mode_of_payment, Transaction_id, Account_no, Date_of_purchase

Normalization Form:

The Payment table provided is in both 1NF and 2NF. It satisfies the criteria for 1NF by having atomic values and consistent data types in columns. It also meets the criteria for 2NF as it does not contain partial dependencies, and its non-key attributes are fully functionally dependent on the primary key.

Decomposed Table (SQL*Plus):

```
-- Assume Account is decomposed to Account_Details and Account_Transactions
  CREATE TABLE Payment (
  Reference_no INT PRIMARY KEY,
  Payment_amount DECIMAL(10, 2) NOT NULL,
  Mode_of_payment VARCHAR(50) NOT NULL,
  Transaction_id VARCHAR(50) NOT NULL UNIQUE,
  Account_no INT,
  Date_of_purchase DATE,
  FOREIGN KEY (Account_no) REFERENCES Account_Details(Account_no)
);
```



```
mysql> desc payment;
+------------------+---------------+------+-----+---------+-------+
| Field            | Type          | Null | Key | Default | Extra |
+------------------+---------------+------+-----+---------+-------+
| Reference_no     | int           | NO   | PRI | NULL    |       |
| Payment_amount   | decimal(10,2) | NO   |     | NULL    |       |
| Mode_of_payment  | varchar(50)   | NO   |     | NULL    |       |
| Transaction_id   | varchar(50)   | NO   | UNI | NULL    |       |
| Account_no       | int           | YES  | MUL | NULL    |       |
| Date_of_purchase | date          | YES  |     | NULL    |       |
+------------------+---------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Pitfalls:

Redundancy: Depending on the broader schema, there might be redundancy if payment-related information is also stored elsewhere (e.g., customer's account details).

TABLE CREDIT CARD:

```
CREATE TABLE CreditCard (
card_id INT PRIMARY KEY,
customer_id INT NOT NULL,
card_number VARCHAR(16) NOT NULL,
expiration_date DATE NOT NULL
card_holder_name VARCHAR(255) NOT NULL,
billing_address varchar(20)
FOREIGN KEY (customer_id) REFERENCES customer(account_no)
);
```

Functional Dependencies:

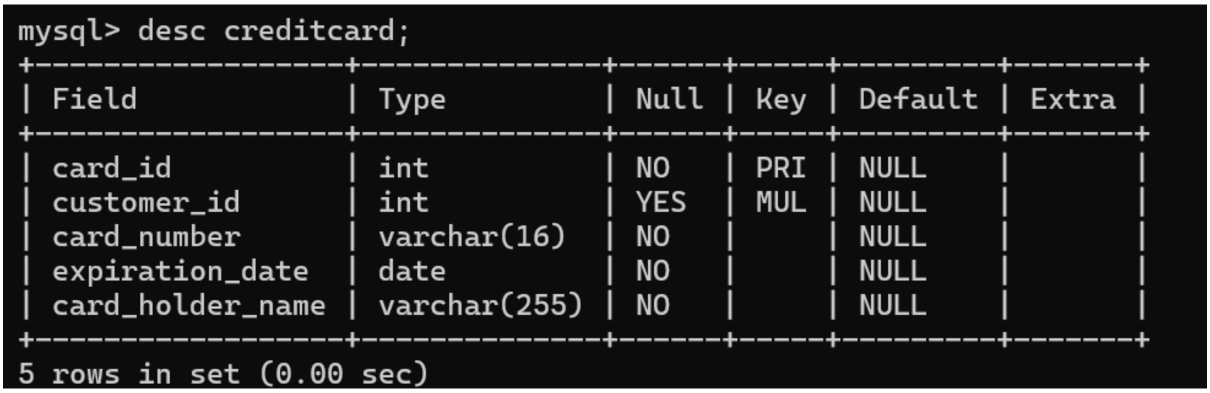
card_id → customer_id, card_number, expiration_date, card_holder_name

Normalization Form:

The CreditCard table satisfies the requirements of Third Normal Form (3NF) based on the given structure and dependency analysis.

Decomposed Table (SQL*Plus):

```
-- Assume Customer is decomposed to Customer_Details and Customer_Account
CREATE TABLE CreditCard (
card_id INT PRIMARY KEY,
customer_id INT,
card_number VARCHAR(16) NOT NULL,
expiration_date DATE NOT NULL,
card_holder_name VARCHAR(255) NOT NULL,
FOREIGN KEY (customer_id) REFERENCES Customer_Account(account_no)
);
```



```
mysql> desc creditcard;
+------------------+--------------+------+-----+---------+-------+
| Field            | Type         | Null | Key | Default | Extra |
+------------------+--------------+------+-----+---------+-------+
| card_id          | int          | NO   | PRI | NULL    |       |
| customer_id      | int          | YES  | MUL | NULL    |       |
| card_number      | varchar(16)  | NO   |     | NULL    |       |
| expiration_date  | date         | NO   |     | NULL    |       |
| card_holder_name | varchar(255) | NO   |     | NULL    |       |
+------------------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Pitfalls:

Transitive Dependencies: If card_number determines expiration_date or card_holder_name, this can lead to transitive dependencies.

TABLE DEBIT CARD:

```
CREATE TABLE DebitCard (
customer_id INT primary key,
card_number VARCHAR(16) NOT NULL,
expiration_date DATE NOT NULL,
card_holder_name VARCHAR(255) NOT NULL,
amount decimal(18,2),
FOREIGN KEY (customer_id) REFERENCES customer(account_no)
);
```

Functional Dependencies:

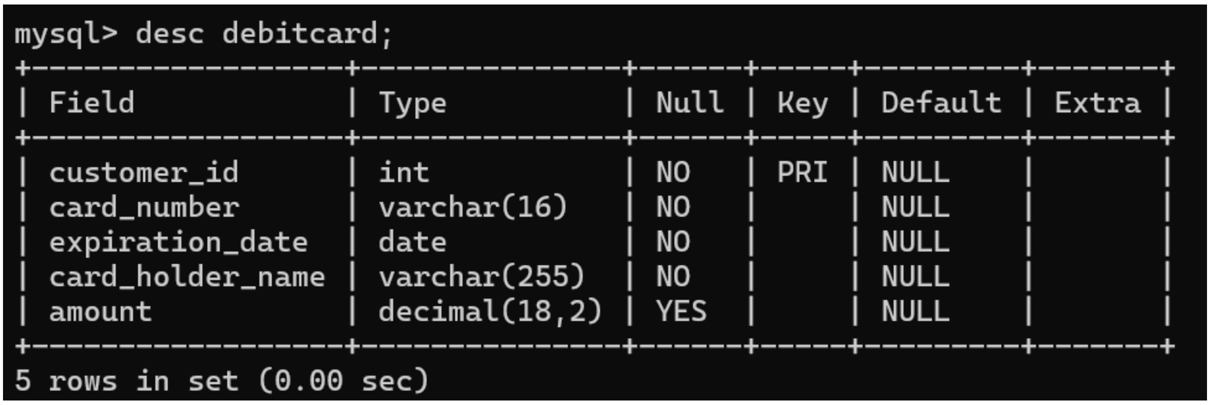
customer_id → card_number, expiration_date, card_holder_name, amount

Normalization Form:

The DebitCard table satisfies the requirements of Third Normal Form (3NF) based on the given structure and dependency analysis.

Decomposed Table (SQL*Plus):

```
-- Assume Customer is decomposed to Customer_Details and Customer_Account
CREATE TABLE DebitCard (
customer_id INT PRIMARY KEY,
card_number VARCHAR(16) NOT NULL,
expiration_date DATE NOT NULL,
card_holder_name VARCHAR(255) NOT NULL,
amount DECIMAL(18, 2),
FOREIGN KEY (customer_id) REFERENCES Customer_Account(account_no)
);
```



```
mysql> desc debitcard;
+------------------+---------------+------+-----+---------+-------+
| Field            | Type          | Null | Key | Default | Extra |
+------------------+---------------+------+-----+---------+-------+
| customer_id      | int           | NO   | PRI | NULL    |       |
| card_number      | varchar(16)   | NO   |     | NULL    |       |
| expiration_date  | date          | NO   |     | NULL    |       |
| card_holder_name | varchar(255)  | NO   |     | NULL    |       |
| amount           | decimal(18,2) | YES  |     | NULL    |       |
+------------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Pitfalls:

Transitive Dependencies: Similar to the CreditCard table, there may be transitive dependencies if expiration_date or card_holder_name rely on card_number rather than the primary key.

TABLE LOAN1:

CREATE TABLE loan1(
account_no int,
loan_type int,
loan_no varchar(20),
amount int,
interest int,
PRIMARY KEY (account_no),
FOREIGN KEY (account_no) REFERENCES customer(account_no));

Functional Dependencies:

account_no → loan_type, loan_no, amount, interest

Normalization Form:

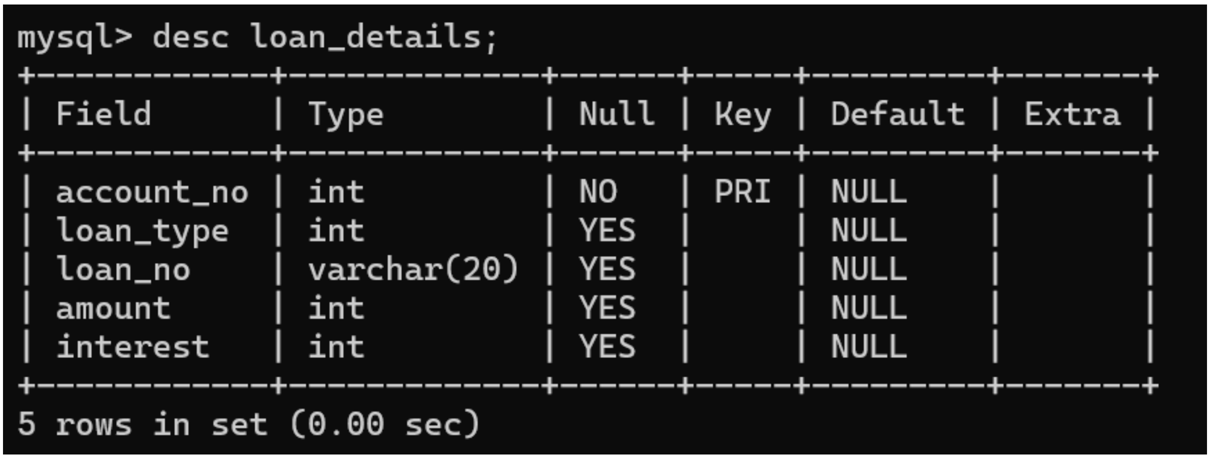
The loan1 table is in First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), and Fourth Normal Form (4NF) based on the provided structure and normalization principles. This table design effectively manages the relationships between account_no and associated loan details (loan_type, loan_no, amount, interest), ensuring data integrity and efficiency.

Decomposed Table (SQL*Plus):

```
-- Assume Customer is decomposed to Customer_Details and Customer_Account
CREATE TABLE Loan_Details (
account_no INT PRIMARY KEY,
loan_type INT,
loan_no VARCHAR(20),
amount INT,
interest INT,
FOREIGN KEY (account_no) REFERENCES Customer_Account(account_no)
);
```

Pitfalls:

Functional Dependencies: If attributes like loan_type, loan_no, amount, or interest depend on each other rather than solely on the primary key (account_no), it could lead to functional dependency issues.



```
mysql> desc loan_details;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| account_no | int         | NO   | PRI | NULL    |       |
| loan_type  | int         | YES  |     | NULL    |       |
| loan_no    | varchar(20) | YES  |     | NULL    |       |
| amount     | int         | YES  |     | NULL    |       |
| interest   | int         | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

TABLE TRANSACTION DETAILS:

```
CREATE TABLE Account_Transactions (
   transaction_id INT PRIMARY KEY,
   account_no INT,
   amount DECIMAL(10, 2) NOT NULL,
   transaction_date DATE NOT NULL,
   FOREIGN KEY (account_no) REFERENCES Account_Details(account_no)
);
```

Functional Dependencies:

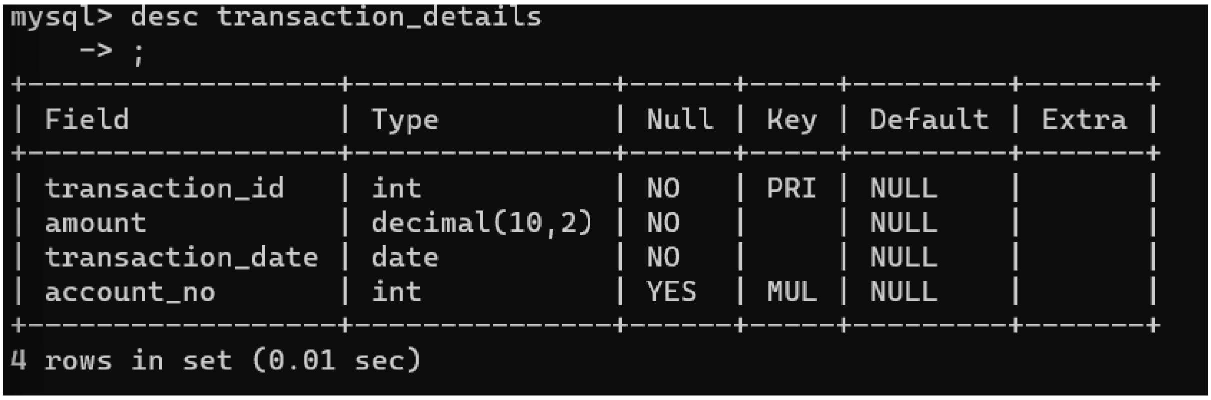
transaction_id -> amount, transaction_date, account_no

NF Forms:

Appears to be in 3NF.

Decomposed Table (SQL*Plus):

```
CREATE TABLE Transaction_Details (
    transaction_id INT PRIMARY KEY,
    amount DECIMAL(10, 2) NOT NULL,
    transaction_date DATE NOT NULL,
    account_no INT,
    FOREIGN KEY (account_no) REFERENCES Account_Details(account_no)
);
```



```
mysql> desc transaction_details
    -> ;
+------------------+---------------+------+-----+---------+-------+
| Field            | Type          | Null | Key | Default | Extra |
+------------------+---------------+------+-----+---------+-------+
| transaction_id   | int           | NO   | PRI | NULL    |       |
| amount           | decimal(10,2) | NO   |     | NULL    |       |
| transaction_date | date          | NO   |     | NULL    |       |
| account_no       | int           | YES  | MUL | NULL    |       |
+------------------+---------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Pitfalls:

By decomposing the table, we avoid redundancy and ensure that each table serves a specific purpose, improving data integrity and reducing anomalies.

TABLE GOVERNMENT SERVICES :

```
CREATE TABLE GovernmentServices (
ServiceID int PRIMARY KEY,
ServiceName VARCHAR(50) NOT NULL,
ServiceDescription VARCHAR(255) NOT NULL,
CustomerID varchar(15) unique,
FOREIGN KEY (CustomerID) REFERENCES Customer(user_id)
);
```

Functional Dependencies:

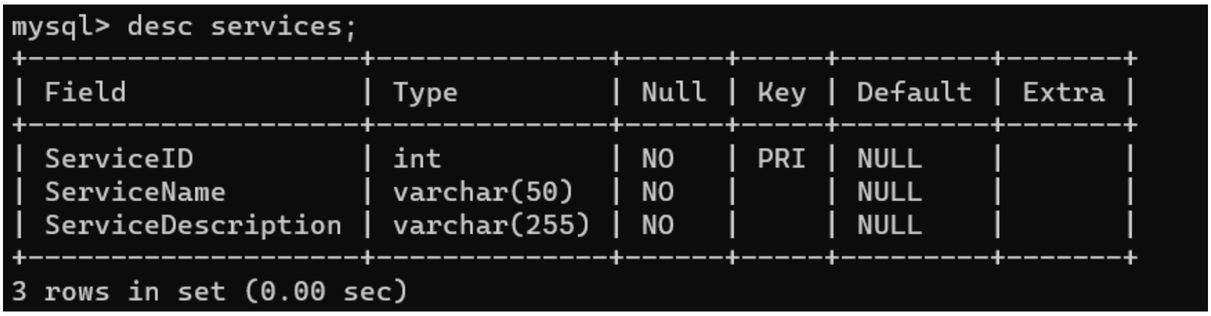
ServiceID -> ServiceName, ServiceDescription, CustomerID, account_no

NF Forms:

Appears to be in 3NF.

Decomposed Table (SQL*Plus):

```
CREATE TABLE ServiceCustomers (
   ServiceID INT,
   user_id VARCHAR(25),
   PRIMARY KEY (ServiceID, user_id),
   FOREIGN KEY (ServiceID) REFERENCES Services(ServiceID),
   FOREIGN KEY (user_id) REFERENCES Customer_Account(user_id)
);
```



```
mysql> desc services;
+--------------------+--------------+------+-----+---------+-------+
| Field              | Type         | Null | Key | Default | Extra |
+--------------------+--------------+------+-----+---------+-------+
| ServiceID          | int          | NO   | PRI | NULL    |       |
| ServiceName        | varchar(50)  | NO   |     | NULL    |       |
| ServiceDescription | varchar(255) | NO   |     | NULL    |       |
+--------------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Pitfalls:

However, querying customer information now requires joining the Customer_Profile and Customer_Account_Mapping tables, which can increase complexity and potentially impact performance. Proper indexing and optimization strategies should be implemented to mitigate this issue.

TABLE ACCOUNT:

CREATE TABLE account(

account_no INT,

Saving_account VARCHAR(5),

Current_account VARCHAR(5),

Balance INT,

PRIMARY KEY (account_no),

FOREIGN KEY (account_no) REFERENCES customer(account_no));

Functional Dependencies:

account_no → Saving_account

account_no → Current_account

account_no → Balance

NF Forms:

the account table is in 3NF.

DECOMPOSED TABLE:

```
CREATE TABLE Account_Details (
account_no INT PRIMARY KEY,
Saving_account VARCHAR(5),
Current_account VARCHAR(5),
Balance INT,
FOREIGN KEY (account_no) REFERENCES customer_account(account_no)
);
CREATE TABLE Account_Transactions (
transaction_id INT PRIMARY KEY,
account_no INT,
amount DECIMAL(10, 2) NOT NULL,
transaction_date DATE NOT NULL,
FOREIGN KEY (account_no) REFERENCES Account_Details(account_no)
);
```

PITFALLS:

Normalization: Refactor the table structure to eliminate redundancy and improve data integrity (e.g., using a separate account_type column with standardized values).

Table Branch:-

```
CREATE TABLE branch (
account_no INT,
branch_city CHAR(20),
IFSC_code VARCHAR(10),
assests VARCHAR(15),
PRIMARY KEY (account_no),
FOREIGN KEY (account_no) REFERENCES customer(account_no)
);
```

Functional Dependencies:

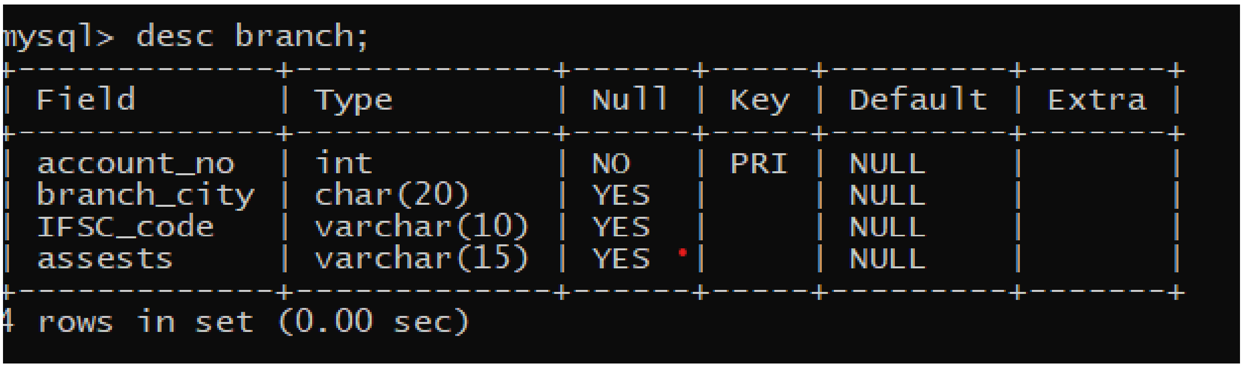
- account_no → branch_city, IFSC_code, assets

Normal Form:

- This table is in *2NF*. All non-key attributes (branch_city, IFSC_code, assets) are fully functionally dependent on the primary key (account_no).

Decomposed Table (SQL*Plus):

```
Create table branch (
account_no INT PRIMARY KEY,
branch_city CHAR(20),
IFSC_code VARCHAR(10),
assets VARCHAR(15),
FOREIGN KEY (account_no) REFERENCES Customer_Details(account_no)
);
```



```
mysql> desc branch;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| account_no  | int         | NO   | PRI | NULL    |       |
| branch_city | char(20)    | YES  |     | NULL    |       |
| IFSC_code   | varchar(10) | YES  |     | NULL    |       |
| assests     | varchar(15) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Pitfalls:

Pitfall: Redundancy could arise if multiple branches share the same attributes (branch_city, IFSC_code, assets).

Table:- Mobile Banking Apps

```
CREATE TABLE MobileBankingApps (
AppID int PRIMARY KEY,
AppName VARCHAR(50) NOT NULL,
MobilePayments char(2) NOT NULL,
BiometricLogin char(2) NOT NULL,
QRCodeTransactions char(2) NOT NULL,
CustomerID varchar(15) unique,
FOREIGN KEY (CustomerID) REFERENCES Customer(user_id)
);
```

Functional Dependencies:

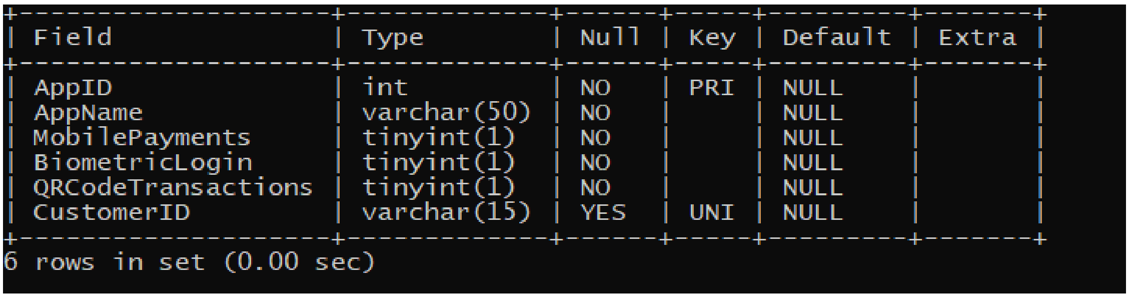
AppID → AppName, MobilePayments, BiometricLogin, QRCodeTransactions, CustomerID

Normal Form:

This table is in *3NF*. All non-key attributes are functionally dependent on the primary key (AppID) without any transitive dependencies.

Decomposed Table (SQL*Plus):

```
Create table MobileBankingApps (
AppID INT PRIMARY KEY,App
Name VARCHAR(50) NOT NULL,
MobilePayments CHAR(2) NOT NULL,
BiometricLogin CHAR(2) NOT NULL,
QRCodeTransactions CHAR(2) NOT NULL,
CustomerID VARCHAR(15) UNIQUE,
FOREIGN KEY (CustomerID) REFERENCES Customer_Account(user_id)
);
```



```
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| AppID              | int         | NO   | PRI | NULL    |       |
| AppName            | varchar(50) | NO   |     | NULL    |       |
| MobilePayments     | tinyint(1)  | NO   |     | NULL    |       |
| BiometricLogin     | tinyint(1)  | NO   |     | NULL    |       |
| QRCodeTransactions | tinyint(1)  | NO   |     | NULL    |       |
| CustomerID         | varchar(15) | YES  | UNI | NULL    |       |
+--------------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Pitfalls:

Pitfall: Redundant storage of CustomerID might lead to inconsistencies if updates are not properly synchronized with the Customer_Account table.

TABLE BANK TO BANK

```
create table banktobank(

bank_name varchar(255),

branch varchar(255),

amount decimal(15,2),

transaction_date timestamp,

account_no int,

FOREIGN KEY (account_no) REFERENCES account(account_no));
```

Functional Dependencies:

account_no → bank_name, branch, amount, transaction_date

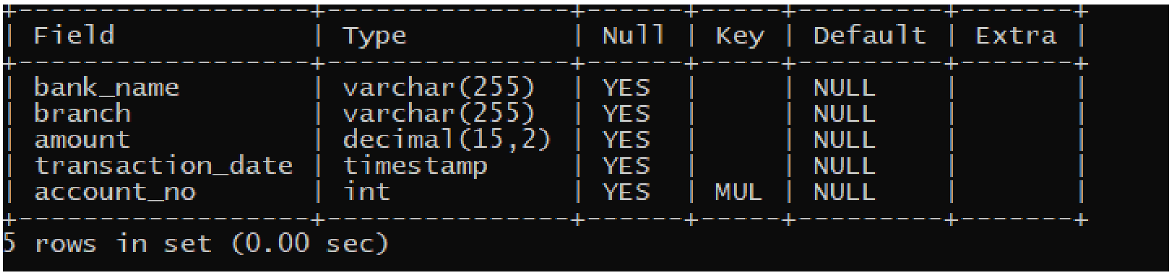
Normal Form:

This table is in *3NF*. All non-key attributes are dependent on the primary key(account_no) and there are no transitive dependencies.

Decomposed Table (SQL*Plus):

```
BankToBank (

bank_name VARCHAR(255),

branch VARCHAR(255),

amount DECIMAL(15,2),

transaction_date TIMESTAMP,

account_no INT,
```

FOREIGN KEY (account_no) REFERENCES
Customer_Account(account_no));



```
+------------------+---------------+------+-----+---------+-------+
| Field            | Type          | Null | Key | Default | Extra |
+------------------+---------------+------+-----+---------+-------+
| bank_name        | varchar(255)  | YES  |     | NULL    |       |
| branch           | varchar(255)  | YES  |     | NULL    |       |
| amount           | decimal(15,2) | YES  |     | NULL    |       |
| transaction_date | timestamp     | YES  |     | NULL    |       |
| account_no       | int           | YES  | MUL | NULL    |       |
+------------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

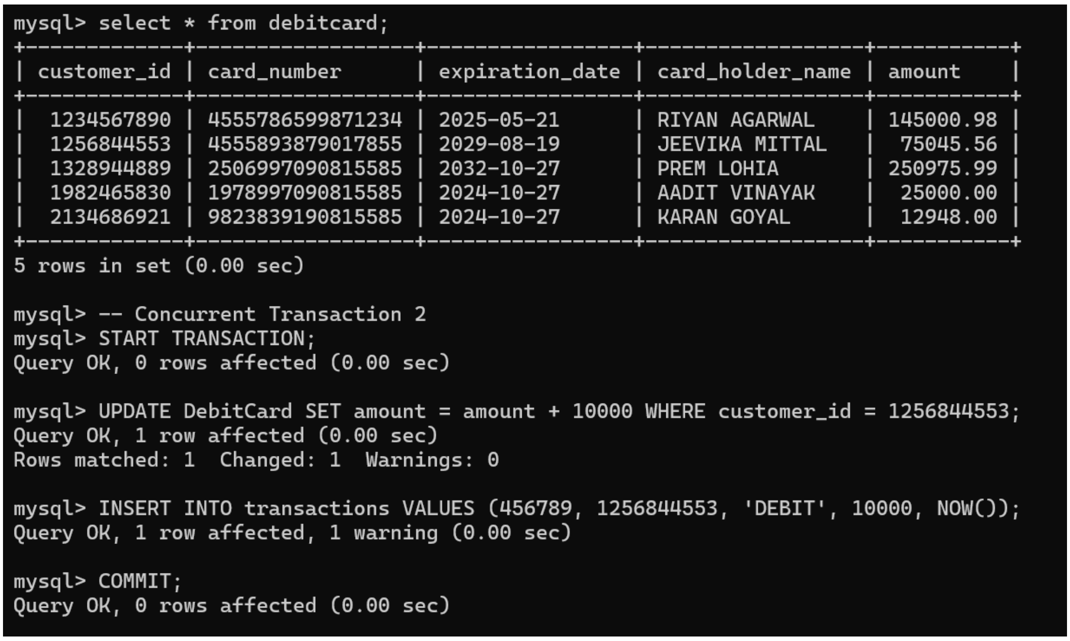
Pitfalls: Pitfall: Storing bank and branch information redundantly across multiple records could leadto inconsistencies or update anomalies.

Chapter 9

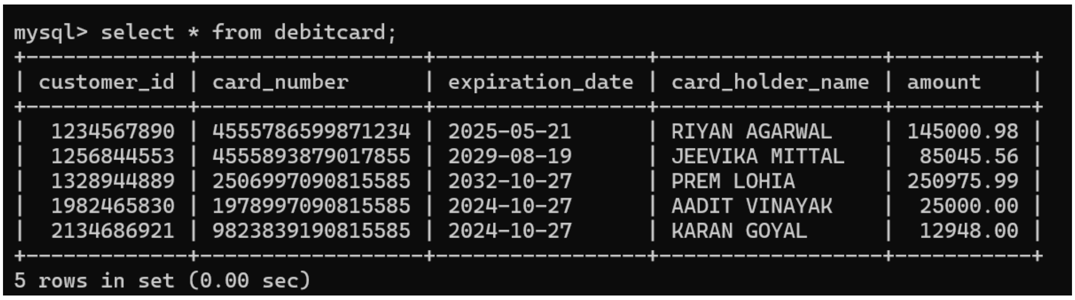
CONCURRENCY CONTROL

There are three codes for concurrent method

To add amount 10000 in the debit card in the account no 1256844553

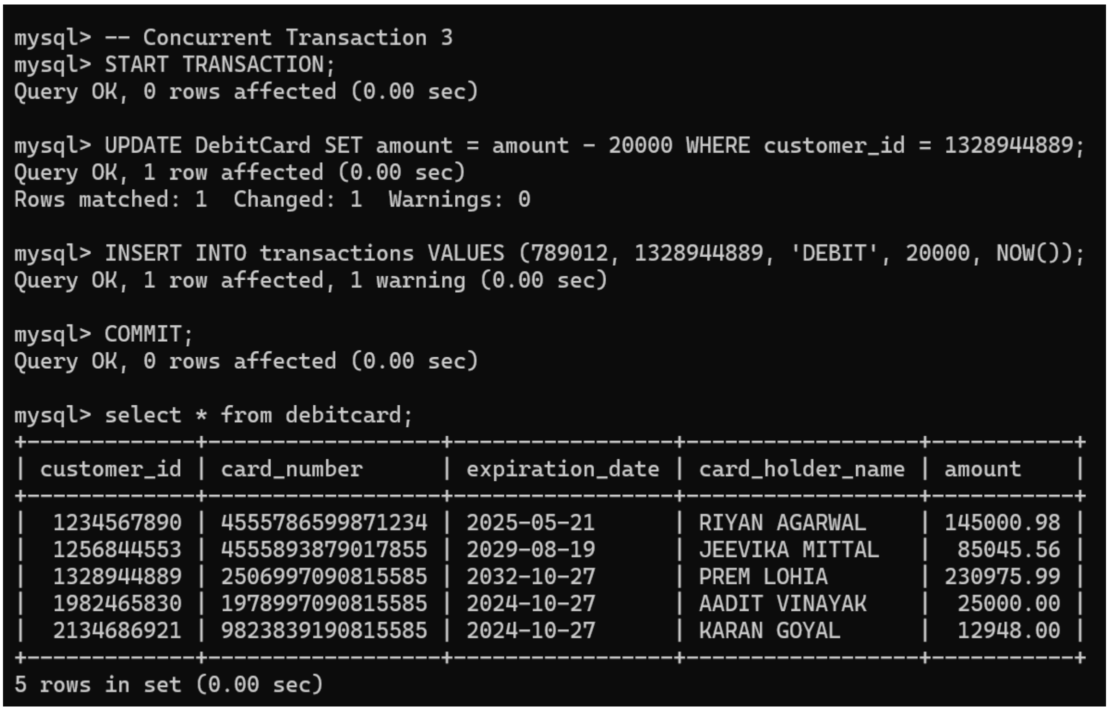


```
mysql> select * from debitcard;
+-------------+------------------+-----------------+------------------+-----------+
| customer_id | card_number      | expiration_date | card_holder_name | amount    |
+-------------+------------------+-----------------+------------------+-----------+
|  1234567890 | 4555786599871234 | 2025-05-21      | RIYAN AGARWAL    | 145000.98 |
|  1256844553 | 4555893879017855 | 2029-08-19      | JEEVIKA MITTAL   |  75045.56 |
|  1328944889 | 2506997090815585 | 2032-10-27      | PREM LOHIA       | 250975.99 |
|  1982465830 | 1978997090815585 | 2024-10-27      | AADIT VINAYAK    |  25000.00 |
|  2134686921 | 9823839190815585 | 2024-10-27      | KARAN GOYAL      |  12948.00 |
+-------------+------------------+-----------------+------------------+-----------+
5 rows in set (0.00 sec)

mysql> -- Concurrent Transaction 2
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE DebitCard SET amount = amount + 10000 WHERE customer_id = 1256844553;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> INSERT INTO transactions VALUES (456789, 1256844553, 'DEBIT', 10000, NOW());
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```



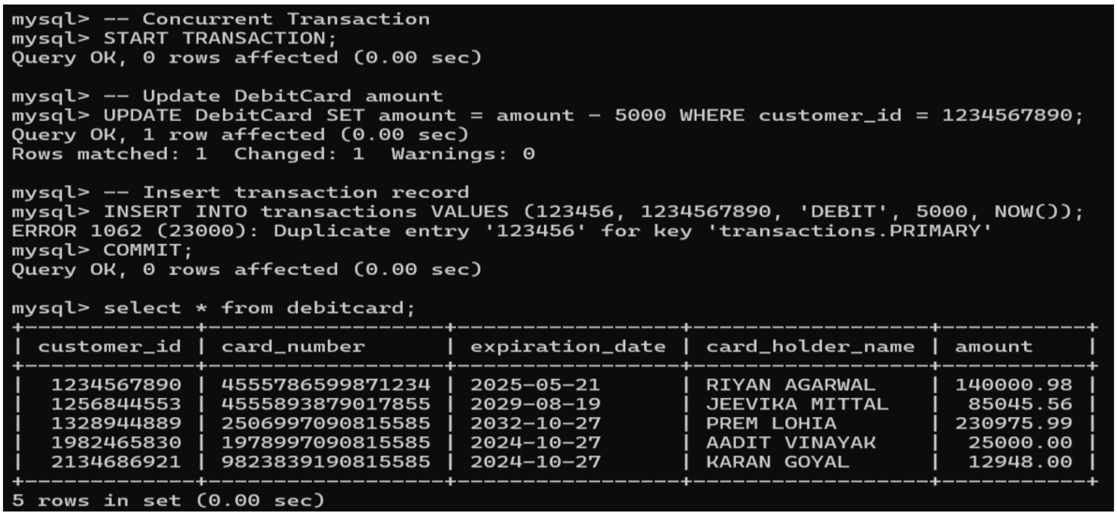
```
mysql> select * from debitcard;
+-------------+------------------+-----------------+------------------+-----------+
| customer_id | card_number      | expiration_date | card_holder_name | amount    |
+-------------+------------------+-----------------+------------------+-----------+
|  1234567890 | 4555786599871234 | 2025-05-21      | RIYAN AGARWAL    | 145000.98 |
|  1256844553 | 4555893879017855 | 2029-08-19      | JEEVIKA MITTAL   |  85045.56 |
|  1328944889 | 2506997090815585 | 2032-10-27      | PREM LOHIA       | 250975.99 |
|  1982465830 | 1978997090815585 | 2024-10-27      | AADIT VINAYAK    |  25000.00 |
|  2134686921 | 9823839190815585 | 2024-10-27      | KARAN GOYAL      |  12948.00 |
+-------------+------------------+-----------------+------------------+-----------+
5 rows in set (0.00 sec)
```

To subtract amount 20000 in the debit card in the account no 1328944889



```
mysql> -- Concurrent Transaction 3
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE DebitCard SET amount = amount - 20000 WHERE customer_id = 1328944889;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> INSERT INTO transactions VALUES (789012, 1328944889, 'DEBIT', 20000, NOW());
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from debitcard;
+-------------+------------------+-----------------+------------------+-----------+
| customer_id | card_number      | expiration_date | card_holder_name | amount    |
+-------------+------------------+-----------------+------------------+-----------+
|  1234567890 | 4555786599871234 | 2025-05-21      | RIYAN AGARWAL    | 145000.98 |
|  1256844553 | 4555893879017855 | 2029-08-19      | JEEVIKA MITTAL   |  85045.56 |
|  1328944889 | 2506997090815585 | 2032-10-27      | PREM LOHIA       | 230975.99 |
|  1982465830 | 1978997090815585 | 2024-10-27      | AADIT VINAYAK    |  25000.00 |
|  2134686921 | 9823839190815585 | 2024-10-27      | KARAN GOYAL      |  12948.00 |
+-------------+------------------+-----------------+------------------+-----------+
5 rows in set (0.00 sec)
```

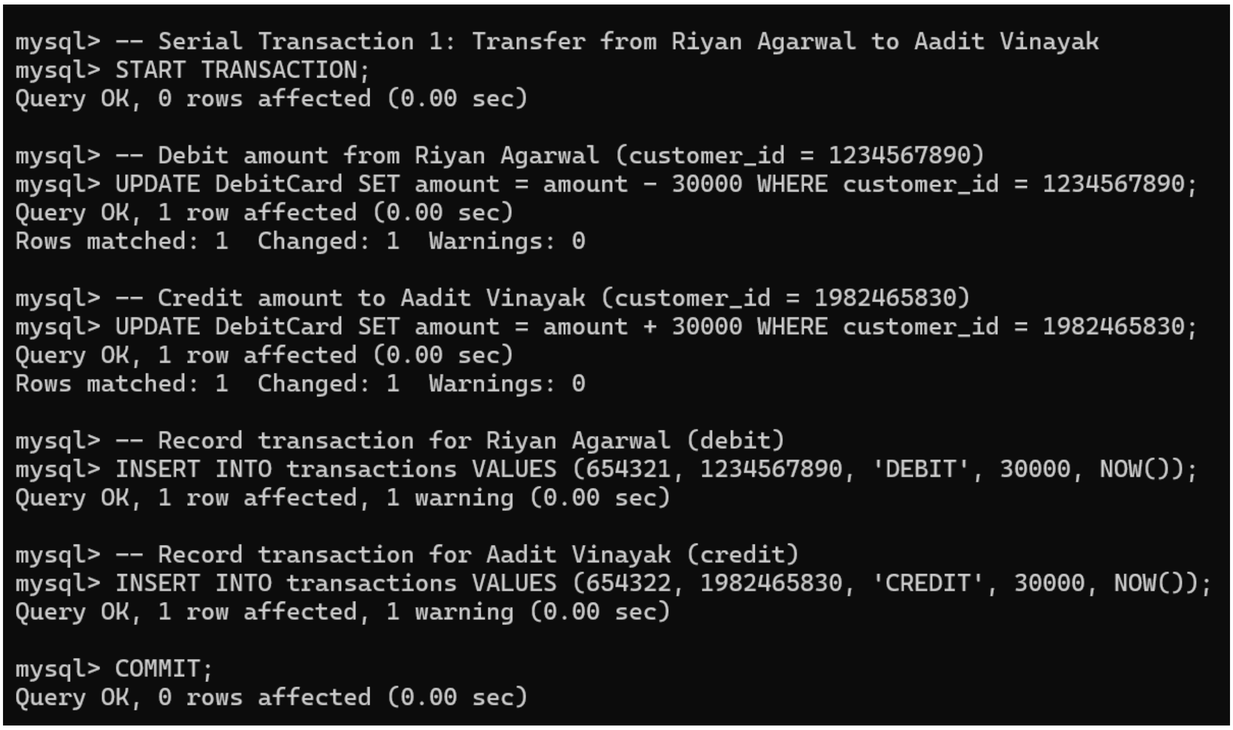
To subtract amount 5000 in the debit card in the account no 1234567890



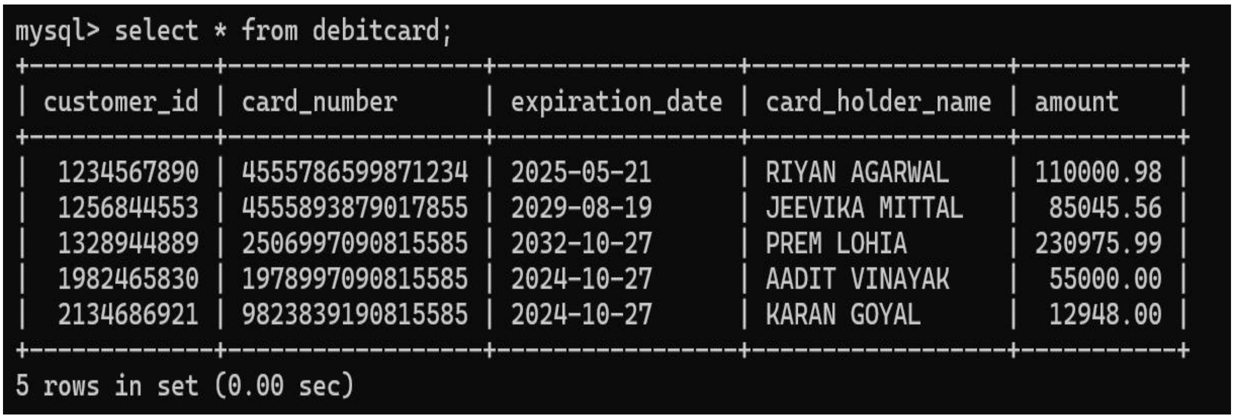
```
mysql> -- Concurrent Transaction
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> -- Update DebitCard amount
mysql> UPDATE DebitCard SET amount = amount - 5000 WHERE customer_id = 1234567890;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> -- Insert transaction record
mysql> INSERT INTO transactions VALUES (123456, 1234567890, 'DEBIT', 5000, NOW());
ERROR 1062 (23000): Duplicate entry '123456' for key 'transactions.PRIMARY'
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from debitcard;
+-------------+------------------+-----------------+------------------+-----------+
| customer_id | card_number      | expiration_date | card_holder_name | amount    |
+-------------+------------------+-----------------+------------------+-----------+
|  1234567890 | 4555786599871234 | 2025-05-21      | RIYAN AGARWAL    | 140000.98 |
|  1256844553 | 4555893879017855 | 2029-08-19      | JEEVIKA MITTAL   |  85045.56 |
|  1328944889 | 2506997090815585 | 2032-10-27      | PREM LOHIA       | 230975.99 |
|  1982465830 | 1978997090815585 | 2024-10-27      | AADIT VINAYAK    |  25000.00 |
|  2134686921 | 9823839190815585 | 2024-10-27      | KARAN GOYAL      |  12948.00 |
+-------------+------------------+-----------------+------------------+-----------+
5 rows in set (0.00 sec)
```

There are three codes for Serial Transaction method

Serial Transaction 1 (Transfer from Riyan Agarwal to Aadit Vinayak)

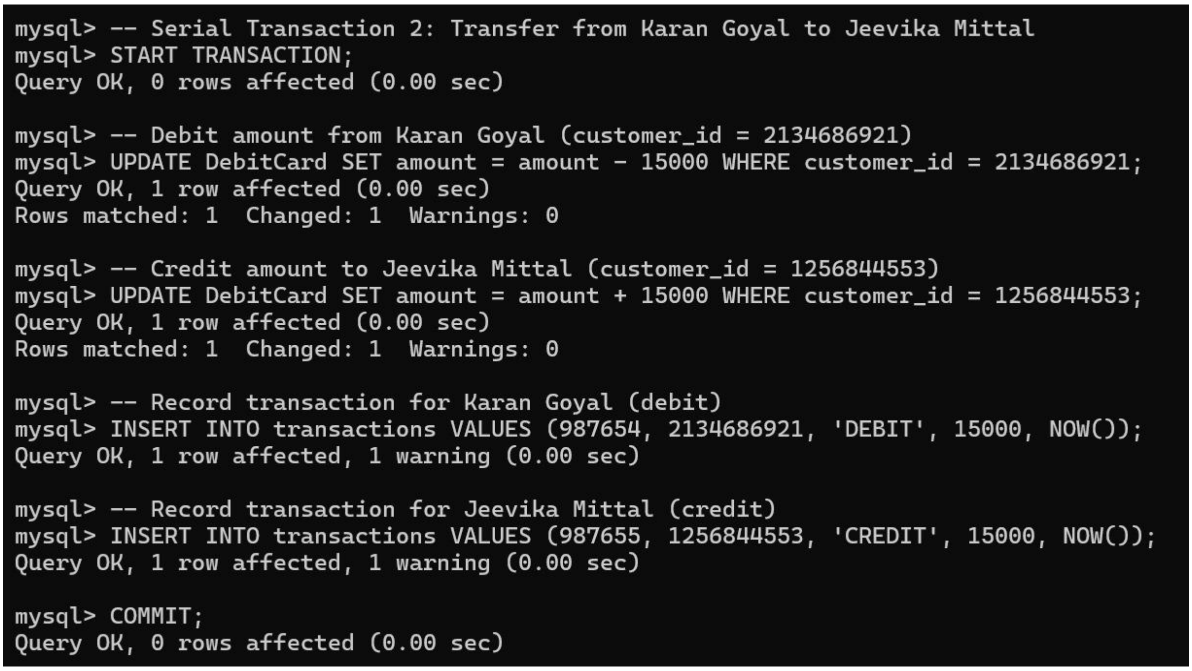


```
mysql> -- Serial Transaction 1: Transfer from Riyan Agarwal to Aadit Vinayak
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> -- Debit amount from Riyan Agarwal (customer_id = 1234567890)
mysql> UPDATE DebitCard SET amount = amount - 30000 WHERE customer_id = 1234567890;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> -- Credit amount to Aadit Vinayak (customer_id = 1982465830)
mysql> UPDATE DebitCard SET amount = amount + 30000 WHERE customer_id = 1982465830;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> -- Record transaction for Riyan Agarwal (debit)
mysql> INSERT INTO transactions VALUES (654321, 1234567890, 'DEBIT', 30000, NOW());
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> -- Record transaction for Aadit Vinayak (credit)
mysql> INSERT INTO transactions VALUES (654322, 1982465830, 'CREDIT', 30000, NOW());
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

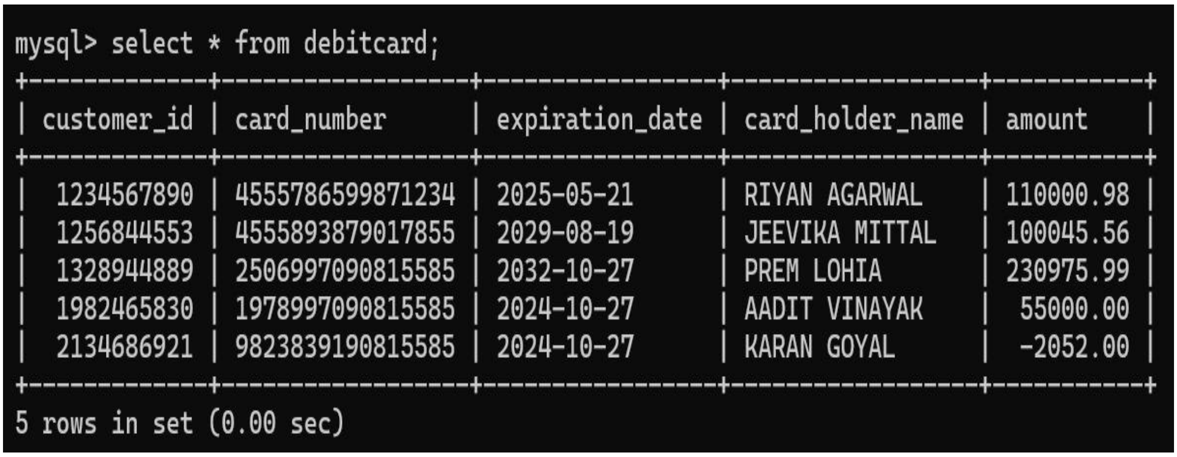


```
mysql> select * from debitcard;
+-------------+------------------+-----------------+------------------+-----------+
| customer_id | card_number      | expiration_date | card_holder_name | amount    |
+-------------+------------------+-----------------+------------------+-----------+
|  1234567890 | 4555786599871234 | 2025-05-21      | RIYAN AGARWAL    | 110000.98 |
|  1256844553 | 4555893879017855 | 2029-08-19      | JEEVIKA MITTAL   |  85045.56 |
|  1328944889 | 2506997090815585 | 2032-10-27      | PREM LOHIA       | 230975.99 |
|  1982465830 | 1978997090815585 | 2024-10-27      | AADIT VINAYAK    |  55000.00 |
|  2134686921 | 9823839190815585 | 2024-10-27      | KARAN GOYAL      |  12948.00 |
+-------------+------------------+-----------------+------------------+-----------+
5 rows in set (0.00 sec)
```

Serial Transaction 2 (Transfer from Karan Goyal to Jeevika Mittal)

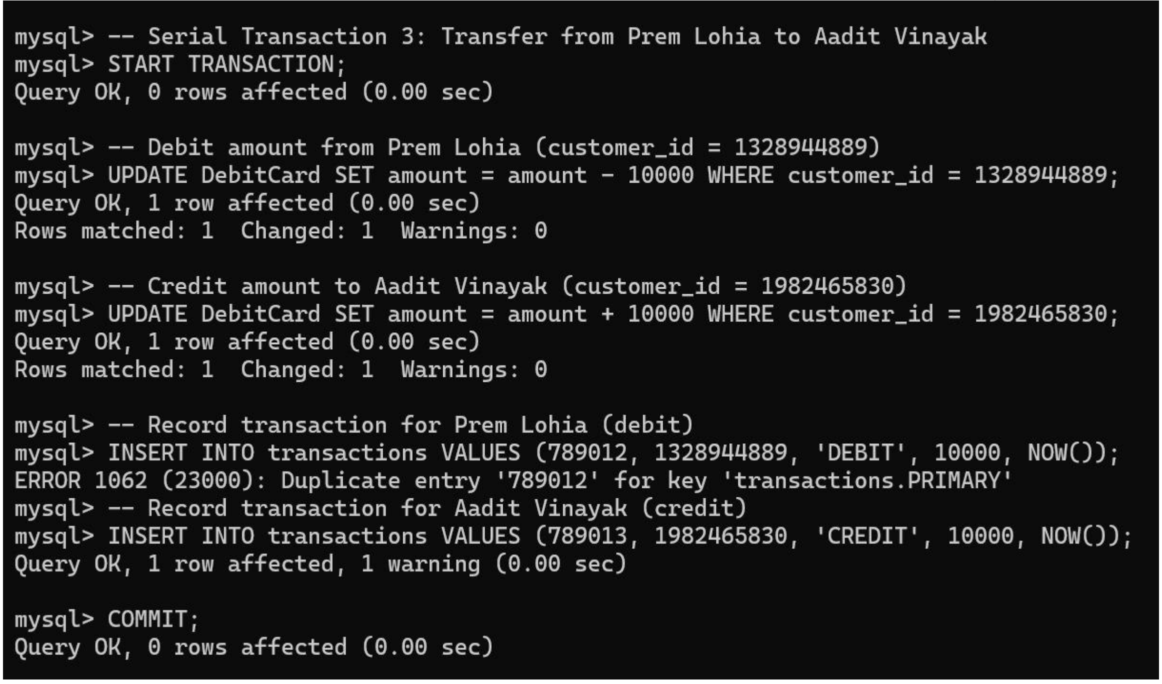


```
mysql> -- Serial Transaction 2: Transfer from Karan Goyal to Jeevika Mittal
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> -- Debit amount from Karan Goyal (customer_id = 2134686921)
mysql> UPDATE DebitCard SET amount = amount - 15000 WHERE customer_id = 2134686921;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> -- Credit amount to Jeevika Mittal (customer_id = 1256844553)
mysql> UPDATE DebitCard SET amount = amount + 15000 WHERE customer_id = 1256844553;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> -- Record transaction for Karan Goyal (debit)
mysql> INSERT INTO transactions VALUES (987654, 2134686921, 'DEBIT', 15000, NOW());
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> -- Record transaction for Jeevika Mittal (credit)
mysql> INSERT INTO transactions VALUES (987655, 1256844553, 'CREDIT', 15000, NOW());
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

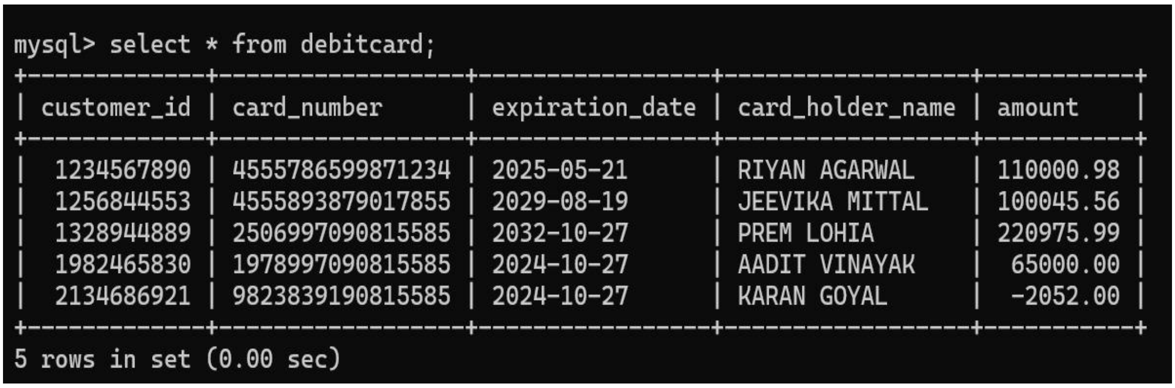


```
mysql> select * from debitcard;
+-------------+------------------+-----------------+------------------+-----------+
| customer_id | card_number      | expiration_date | card_holder_name | amount    |
+-------------+------------------+-----------------+------------------+-----------+
|  1234567890 | 4555786599871234 | 2025-05-21      | RIYAN AGARWAL    | 110000.98 |
|  1256844553 | 4555893879017855 | 2029-08-19      | JEEVIKA MITTAL   | 100045.56 |
|  1328944889 | 2506997090815585 | 2032-10-27      | PREM LOHIA       | 230975.99 |
|  1982465830 | 1978997090815585 | 2024-10-27      | AADIT VINAYAK    |  55000.00 |
|  2134686921 | 9823839190815585 | 2024-10-27      | KARAN GOYAL      |  -2052.00 |
+-------------+------------------+-----------------+------------------+-----------+
5 rows in set (0.00 sec)
```

Serial Transaction 3 (Transfer from Prem Lohia to Aadit Vinayak)



```
mysql> -- Serial Transaction 3: Transfer from Prem Lohia to Aadit Vinayak
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> -- Debit amount from Prem Lohia (customer_id = 1328944889)
mysql> UPDATE DebitCard SET amount = amount - 10000 WHERE customer_id = 1328944889;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> -- Credit amount to Aadit Vinayak (customer_id = 1982465830)
mysql> UPDATE DebitCard SET amount = amount + 10000 WHERE customer_id = 1982465830;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> -- Record transaction for Prem Lohia (debit)
mysql> INSERT INTO transactions VALUES (789012, 1328944889, 'DEBIT', 10000, NOW());
ERROR 1062 (23000): Duplicate entry '789012' for key 'transactions.PRIMARY'
mysql> -- Record transaction for Aadit Vinayak (credit)
mysql> INSERT INTO transactions VALUES (789013, 1982465830, 'CREDIT', 10000, NOW());
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

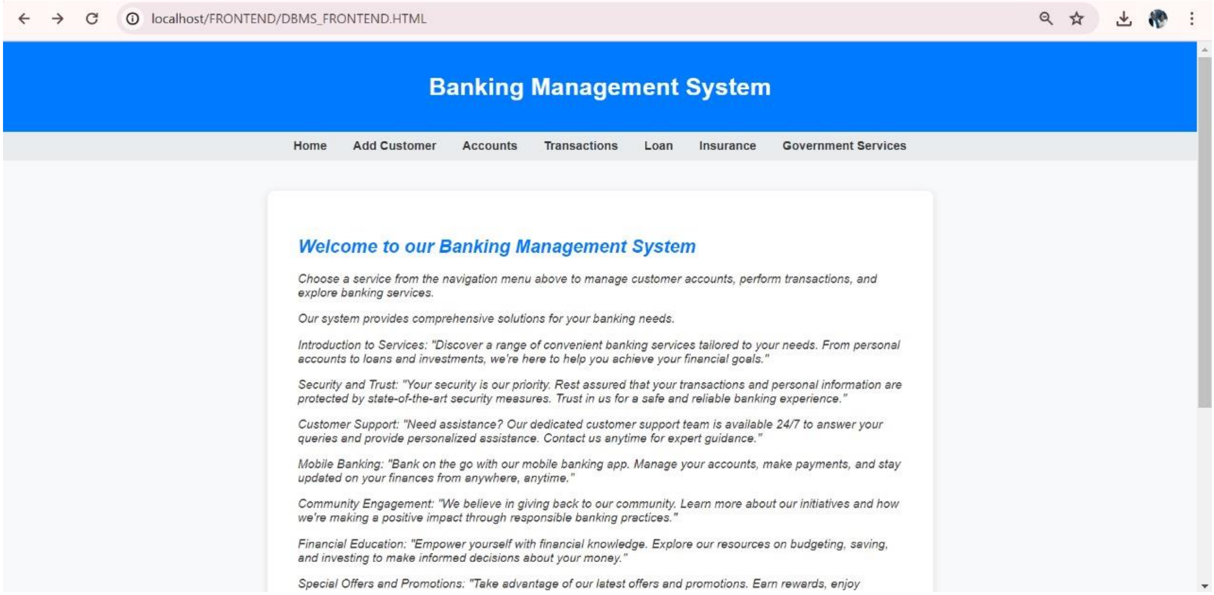


```
mysql> select * from debitcard;
+-------------+------------------+-----------------+------------------+-----------+
| customer_id | card_number      | expiration_date | card_holder_name | amount    |
+-------------+------------------+-----------------+------------------+-----------+
|  1234567890 | 4555786599871234 | 2025-05-21      | RIYAN AGARWAL    | 110000.98 |
|  1256844553 | 4555893879017855 | 2029-08-19      | JEEVIKA MITTAL   | 100045.56 |
|  1328944889 | 2506997090815585 | 2032-10-27      | PREM LOHIA       | 220975.99 |
|  1982465830 | 1978997090815585 | 2024-10-27      | AADIT VINAYAK    |  65000.00 |
|  2134686921 | 9823839190815585 | 2024-10-27      | KARAN GOYAL      |  -2052.00 |
+-------------+------------------+-----------------+------------------+-----------+
5 rows in set (0.00 sec)
```

Chapter 10

FRONT END

FRONT PAGE OF THE WEBSITE



Banking Management System

Welcome to our Banking Management System

Choose a service from the navigation menu above to manage customer accounts, perform transactions, and explore banking services.

Our system provides comprehensive solutions for your banking needs.

Introduction to Services: "Discover a range of convenient banking services tailored to your needs. From personal accounts to loans and investments, we're here to help you achieve your financial goals."

Security and Trust: "Your security is our priority. Rest assured that your transactions and personal information are protected by state-of-the-art security measures. Trust in us for a safe and reliable banking experience."

Customer Support: "Need assistance? Our dedicated customer support team is available 24/7 to answer your queries and provide personalized assistance. Contact us anytime for expert guidance."

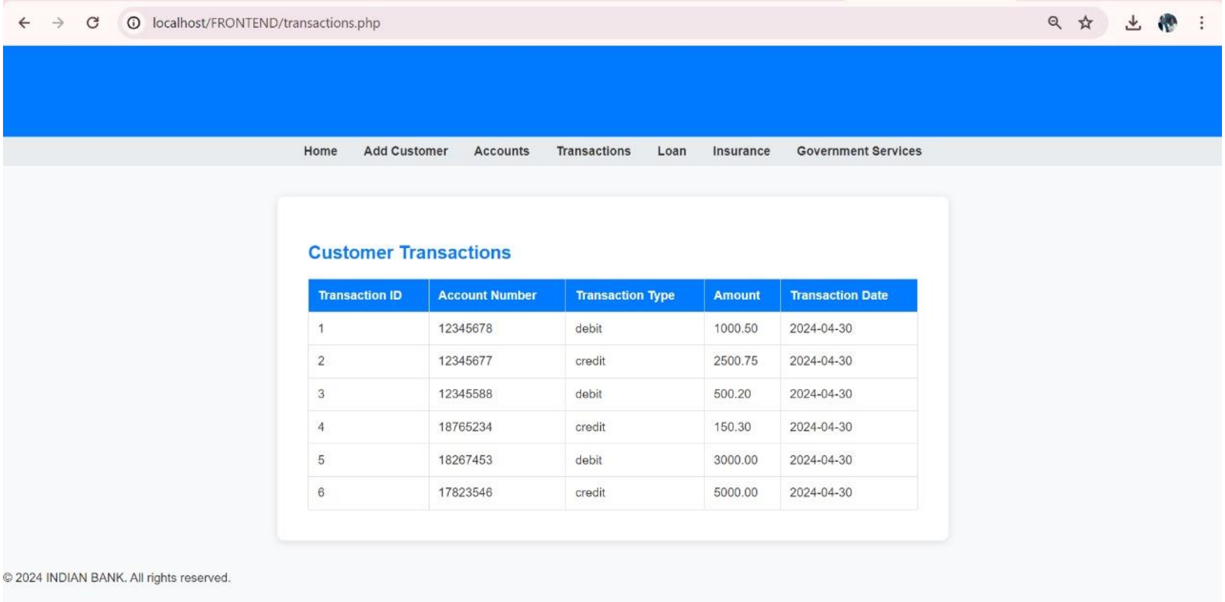
Mobile Banking: "Bank on the go with our mobile banking app. Manage your accounts, make payments, and stay updated on your finances from anywhere, anytime."

Community Engagement: "We believe in giving back to our community. Learn more about our initiatives and how we're making a positive impact through responsible banking practices."

Financial Education: "Empower yourself with financial knowledge. Explore our resources on budgeting, saving, and investing to make informed decisions about your money."

Special Offers and Promotions: "Take advantage of our latest offers and promotions. Earn rewards, enjoy

TRANSACTION OF THE CUSTOMER



Customer Transactions

Transaction ID	Account Number	Transaction Type	Amount	Transaction Date
1	12345678	debit	1000.50	2024-04-30
2	12345677	credit	2500.75	2024-04-30
3	12345588	debit	500.20	2024-04-30
4	18765234	credit	150.30	2024-04-30
5	18267453	debit	3000.00	2024-04-30
6	17823546	credit	5000.00	2024-04-30

© 2024 INDIAN BANK. All rights reserved.

CONCLUSION AND FUTURE ENHANCEMENTS

The Banking Management System (BMS) stands as a beacon of efficiency and innovation in the financial sector. By streamlining operations, enhancing security, and improving customer experience, BMS has become an indispensable tool for modern banks. Its ability to automate transactions, facilitate account management, and provide 24/7 access to customers has revolutionized the way banking services are delivered. However, the current banking system still faces challenges due to manual processes and paper records, leading to inefficiencies and missed opportunities. Despite these challenges, the adoption of BMS signifies a shift towards a more digitized and customer-centric banking landscape.

Future Enhancements:

Looking ahead, the evolution of BMS will continue to address the ever-changing needs of the banking industry. Future enhancements may include:

- **Advanced Data Analytics:** Implementing sophisticated data analytics tools to gain deeper insights into customer behavior, identify trends, and personalize services.
- **Enhanced Security Measures:** Continuously upgrading security measures such as biometric authentication, blockchain technology, and AI-powered fraud detection to safeguard against cyber threats.
- **Integration of Emerging Technologies:** Embracing emerging technologies like machine learning, Internet of Things (IoT), and augmented reality to create innovative banking solutions and improve operational efficiency.
- **Seamless Omni-channel Experience:** Providing a seamless omni-channel experience across various platforms including web, mobile, social media, and physical branches to meet the diverse preferences of customers.
- **Focus on Financial Inclusion:** Leveraging BMS to bridge the gap between the banked and unbanked population by offering accessible and affordable financial services to underserved communities. In essence, the future of BMS lies in its ability to adapt, innovate, and leverage technology to deliver superior banking experiences while addressing the evolving needs of customers and the industry as a whole.

REFERENCES

1. Elmasri, R., Navathe, S. B. (2011)
   "Fundamentals of Database Systems"
   Sixth Edition, Pearson Education
2. Silberschatz, A., Korth, H. F., Sudarshan, S. (2019)
   "Database System Concepts"
   Seventh Edition, Tata McGraw Hill
3. Date, C. J., Kannan, A., Swamynathan, S. (2006)
   "An Introduction to Database Systems"
   Pearson Education
4. Rob, P., Coronel, C. (2015)
   "Database Systems: Design, Implementation & Management"
   12th Edition, Cengage Learning
5. Ramakrishnan, R., Gehrke, J. (2002)
   "Database Management Systems"
   3rd Edition, McGraw-Hill Education

Study material
https://sircrrengg.ac.in/images/CSEMATERIALS/R19_DBMS_MATERIAL.pdf

https://www.researchgate.net/publication/375757563_Banking_Management_System_SRS_Report_Team_Guide