Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Project Report

On

MySQL

[COMP 232]

(For Partial Fulfillment of 2nd year/2nd Semester of Computer Science)

Submitted by:

Anmol Dahal

Roll No. - 11

Submitted to:

Mr. Sanjog Sigdel

The Department of Computer Science andEngineering

Date of Submission: 31st December, 2024

Acknowledgement

I would like to express my heartfelt gratitude to Mr. Sanjog Sigdel for his invaluable guidance and support throughout this project. His insightful feedback and encouragement have been instrumental in the successful completion of this work.

Abstract

This project report presents the design and implementation of a database for an ‘Online Learning Portal.’ The primary goal of the project is to create a structured database for managing courses, instructors and student registrations efficiently. It incorporates essential database concepts like normalization, stored procedure and transaction process. Additionally, the report demonstrates the Entity-Relationship Diagram, showcasing the database’s design and functionality.

Table of content

List of Figures

Chapter 1. Introduction

1.1 Objective

The objectives of this project are:

· To design and implement a structured database for an Online Learning Portal · To apply normalization techniques

· To create an Entity-Relationship Diagram that represents the database design · To demonstrate the use of essential database concepts such as stored procedure and transaction processing

· To execute SQL queries that showcase effective data retrieval and management

1.2 Scope

1. The database will store and manage:
   · Course details, including names and assigned instructors.
   · Instructor details and their association with courses.
   · Student details and their course registrations.
2. Supports efficient querying of student-course and instructor-course relationships.
3. Ensures scalability for an increasing number of records in the future.

Chapter 2. Database Design

Relational ER diagram

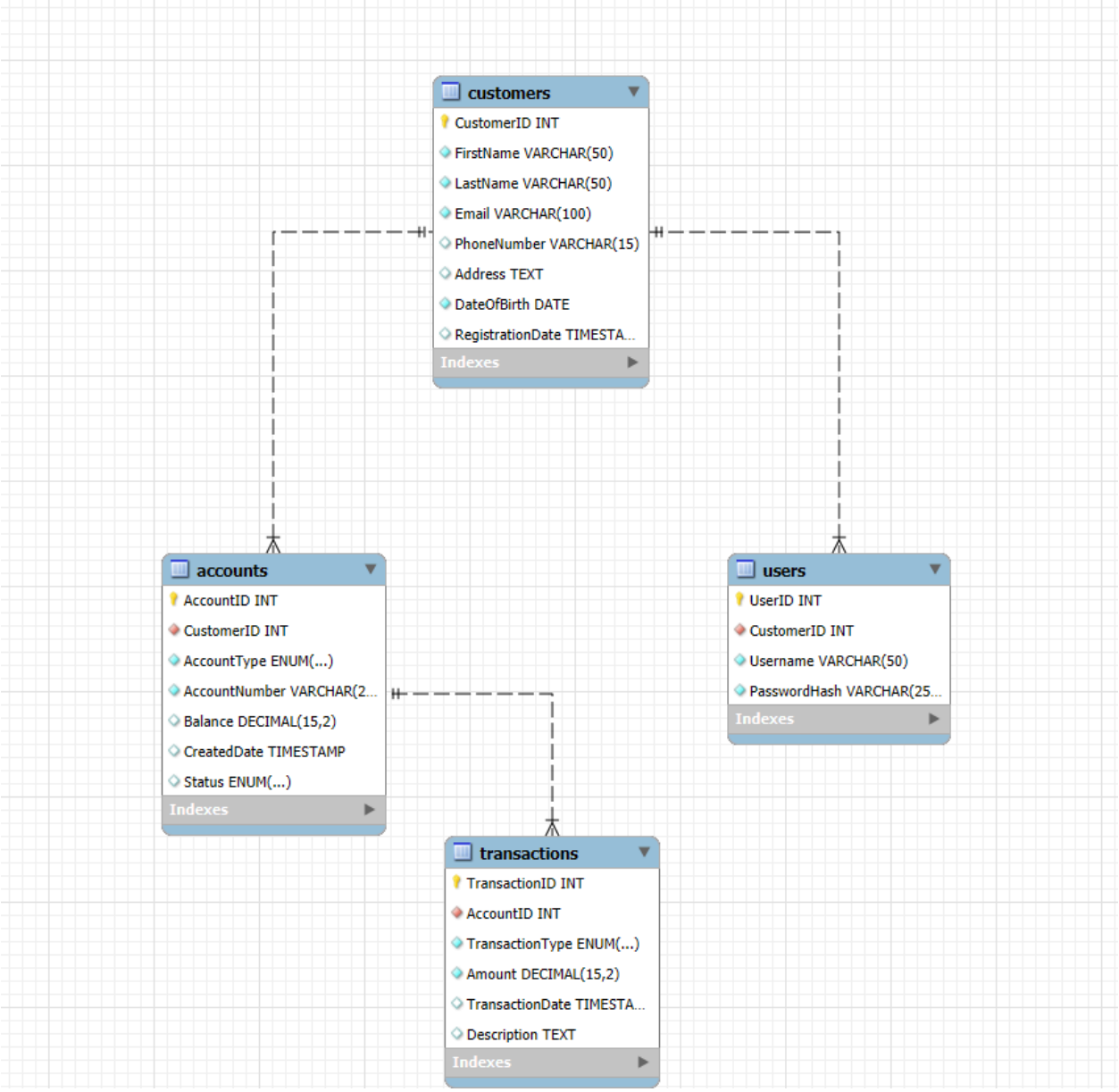


Figure 2 Relational ERD

2.2 Relationships

· customers and accounts: one to many

· customers and users: one to one

· Accounts and transactions : one to many

Chapter 3. Implementation

For the creation and implementation of the database, MySQL Command Line Client is used.

3.1 SQL Queries

Database: Online banking sevices

3.1.1 Use database

```
use online_banking_services;
```

3.1.2 Create tables

o Customers

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    PhoneNumber VARCHAR(15) UNIQUE,
    Address TEXT,
    DateOfBirth DATE NOT NULL,
    RegistrationDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

o accounts

```
CREATE TABLE Accounts (
    AccountID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID INT NOT NULL,
    AccountType ENUM('Checking', 'Savings', 'Loan') NOT NULL,
    AccountNumber VARCHAR(20) UNIQUE NOT NULL,
    Balance DECIMAL(15, 2) DEFAULT 0.00,
    CreatedDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Status ENUM('Active', 'Dormant', 'Closed') DEFAULT 'Active',
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE
);
```

o Transactions

```
CREATE TABLE Transactions (
    TransactionID INT PRIMARY KEY AUTO_INCREMENT,
    AccountID INT NOT NULL,
    TransactionType ENUM('Deposit', 'Withdrawal', 'Transfer') NOT NULL,
    Amount DECIMAL(15, 2) NOT NULL,
    TransactionDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Description TEXT,
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID) ON DELETE CASCADE
);
```

o users

```
CREATE TABLE Users (
    CustomerID INT NOT NULL,
    Username VARCHAR(50) UNIQUE NOT NULL,
    PasswordHash VARCHAR(255) NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Execute the selected portion of the script or everything, if there is no selection

8

3.1.3 Show tables

```
show tables;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Tables_in_online_banking_services
accounts
customers
transactions
users

3.1.4 Describe tables

o Accounts

```
desc accounts;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
AccountID	int	NO	PRI	NULL	auto_increment
CustomerID	int	NO	MUL	NULL	
AccountType	enum('Checking','Savings','Loan')	NO		NULL	
AccountNumber	varchar(20)	NO	UNI	NULL	
Balance	decimal(15,2)	YES		0.00	
CreatedDate	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
Status	enum('Active','Dormant','Closed')	YES		Active	

o Customers

```
1 •  desc customers;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
CustomerID	int	NO	PRI	NULL	auto_increment
FirstName	varchar(50)	NO		NULL	
LastName	varchar(50)	NO		NULL	
Email	varchar(100)	NO	UNI	NULL	
PhoneNumber	varchar(15)	YES	UNI	NULL	
Address	text	YES		NULL	
DateOfBirth	date	NO		NULL	
RegistrationDate	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED

o transactions

```
1 •  desc transactions;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
TransactionID	int	NO	PRI	NULL	auto_increment
AccountID	int	NO	MUL	NULL	
TransactionType	enum('Deposit','Withdrawal','Transfer')	NO		NULL	
Amount	decimal(15,2)	NO		NULL	
TransactionDate	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
Description	text	YES		NULL	

o users

```
1 •  desc users;
```

Execute the selected portion of the script or everything, if there is no

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
UserID	int	NO	PRI	NULL	auto_increment
CustomerID	int	NO	MUL	NULL	
Username	varchar(50)	NO	UNI	NULL	
PasswordHash	varchar(255)	NO		NULL	

o Students

3.1.5 Insert values into the tables & Show them

o customers

```
INSERT INTO Customers (FirstName, LastName, Email, PhoneNumber, Address, DateOfBirth)
VALUES
('Anmol', 'Singh', 'anmol.singh@example.com', '1234567890', '123 Main St, Cityville', '1990-01-01'),
('Amrit', 'Kaur', 'amrit.kaur@example.com', '2345678901', '456 Elm St, Townsville', '1992-02-02'),
('Olive', 'Johnson', 'olive.johnson@example.com', '3456789012', '789 Oak St, Villagetown', '1994-03-03'),
('Snuggle', 'Patel', 'snuggle.patel@example.com', '4567890123', '321 Pine St, Hamletburg', '1996-04-04'),
('Catbahadur', 'Rai', 'catbahadur.rai@example.com', '5678901234', '654 Cedar St, Parkland', '1998-05-05');
```

o accounts

```
INSERT INTO Accounts (CustomerID, AccountType, AccountNumber, Balance, Status)
VALUES
(1, 'Checking', 'CHK0001', 1500.50, 'Active'),
(2, 'Savings', 'SAV0002', 2500.00, 'Active'),
(3, 'Checking', 'CHK0003', 320.75, 'Active'),
(4, 'Savings', 'SAV0004', 5000.00, 'Active'),
(5, 'Checking', 'CHK0005', 125.25, 'Active');
```

o transactions

```
INSERT INTO Transactions (AccountID, TransactionType, Amount, TransactionDate)
VALUES
(1, 'Deposit', 500.00, '2024-12-01 10:15:00'),
(2, 'Withdrawal', 100.00, '2024-12-02 12:30:00'),
(3, 'Transfer', 200.75, '2024-12-03 09:45:00'),
(4, 'Deposit', 3000.00, '2024-12-04 14:00:00'),
(5, 'Withdrawal', 25.25, '2024-12-05 16:20:00');
```

o users

```
INSERT INTO Users (CustomerID, Username, PasswordHash)
VALUES
(1, 'anmol123', 'hashedpassword1'),
(2, 'amrit456', 'hashedpassword2'),
(3, 'olive789', 'hashedpassword3'),
(4, 'snuggle001', 'hashedpassword4'),
(5, 'catbahadur999', 'hashedpassword5');
```

3.1.6 Join operations

o Inner join

```
SELECT Customers.CustomerID, Customers.FirstName, Customers.LastName, Accounts.AccountNumber, Accounts.Balance
FROM Customers
INNER JOIN Accounts ON Customers.CustomerID = Accounts.CustomerID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CustomerID	FirstName	LastName	AccountNumber	Balance
1	Anmol	Singh	CHK0001	1500.50
2	Amrit	Kaur	SAV0002	2500.00
3	Olive	Johnson	CHK0003	320.75
4	Snuggle	Patel	SAV0004	5000.00
5	Catbahadur	Rai	CHK0005	125.25

o Left join

```
SELECT Customers.CustomerID, Customers.FirstName, Customers.LastName, Accounts.AccountNumber, Accounts.Balance
FROM Customers
LEFT JOIN Accounts ON Customers.CustomerID = Accounts.CustomerID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CustomerID	FirstName	LastName	AccountNumber	Balance
1	Anmol	Singh	CHK0001	1500.50
2	Amrit	Kaur	SAV0002	2500.00
3	Olive	Johnson	CHK0003	320.75
4	Snuggle	Patel	SAV0004	5000.00
5	Catbahadur	Rai	CHK0005	125.25

o Right join

```
SELECT Customers.CustomerID, Customers.FirstName, Customers.LastName, Accounts.AccountNumber, Accounts.Balance
FROM Customers
RIGHT JOIN Accounts ON Customers.CustomerID = Accounts.CustomerID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CustomerID	FirstName	LastName	AccountNumber	Balance
1	Anmol	Singh	CHK0001	1500.50
2	Amrit	Kaur	SAV0002	2500.00
3	Olive	Johnson	CHK0003	320.75
4	Snuggle	Patel	SAV0004	5000.00
5	Catbahadur	Rai	CHK0005	125.25

o Full outer join

```
SELECT Customers.CustomerID, Customers.FirstName, Customers.LastName, Accounts.AccountNumber, Accounts.Balance
FROM Customers
LEFT JOIN Accounts ON Customers.CustomerID = Accounts.CustomerID
UNION
SELECT Customers.CustomerID, Customers.FirstName, Customers.LastName, Accounts.AccountNumber, Accounts.Balance
FROM Customers
RIGHT JOIN Accounts ON Customers.CustomerID = Accounts.CustomerID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CustomerID	FirstName	LastName	AccountNumber	Balance
1	Anmol	Singh	CHK0001	1500.50
2	Amrit	Kaur	SAV0002	2500.00
3	Olive	Johnson	CHK0003	320.75
4	Snuggle	Patel	SAV0004	5000.00
5	Catbahadur	Rai	CHK0005	125.25

3.1.7 Specific querying operations

o List of all the customers and their account details

Limit to 1000 rows

```
SELECT Customers.CustomerID,
    CONCAT(Customers.FirstName, ' ', Customers.LastName) AS CustomerName,
    Accounts.AccountNumber,
    Accounts.Balance FROM
    Customers
JOIN Accounts ON Customers.CustomerID = Accounts.CustomerID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CustomerID	CustomerName	AccountNumber	Balance
1	Anmol Singh	CHK0001	1500.50
2	Amrit Kaur	SAV0002	2500.00
3	Olive Johnson	CHK0003	320.75
4	Snuggle Patel	SAV0004	5000.00
5	Catbahadur Rai	CHK0005	125.25

15

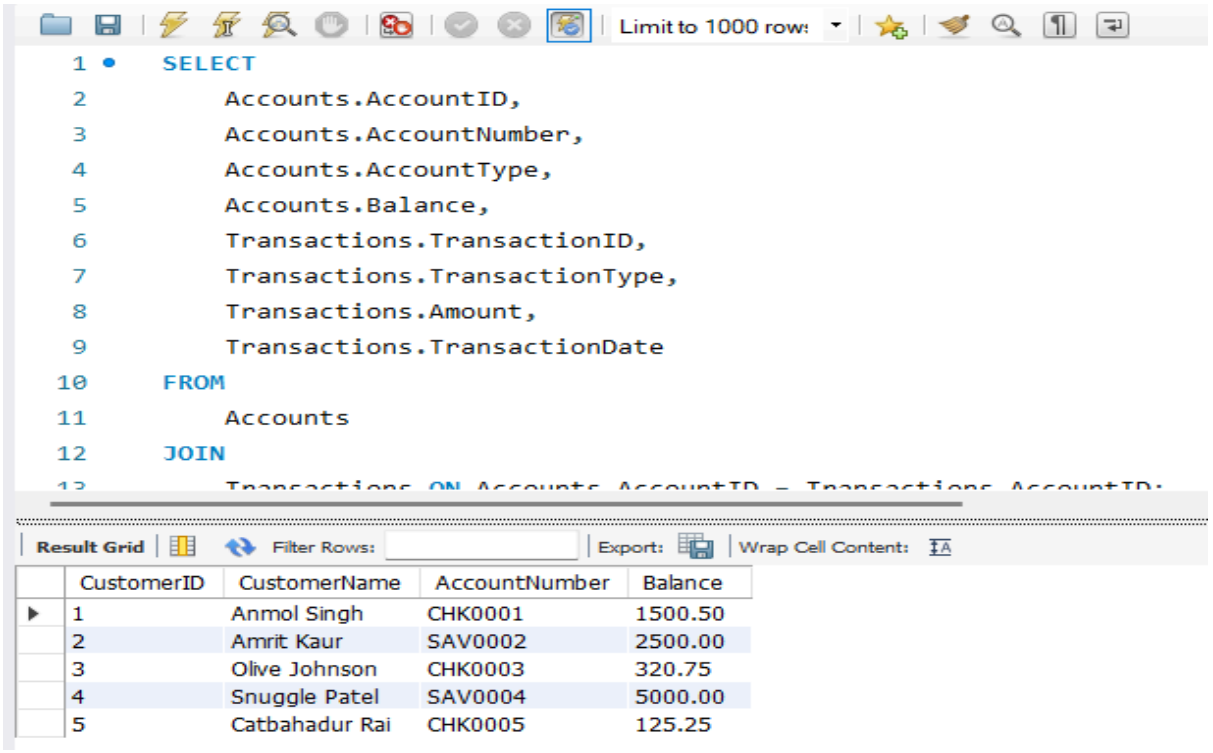
o List of the accounts and their users

```
SELECT
    Accounts.AccountID,
    Accounts.AccountNumber,
    Accounts.AccountType,
    Accounts.Balance,
    Users.Username,
    Users.PasswordHash
FROM
    Accounts
JOIN
    Users ON Accounts.CustomerID = Users.CustomerID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CustomerID	CustomerName	AccountNumber	Balance
1	Anmol Singh	CHK0001	1500.50
2	Amrit Kaur	SAV0002	2500.00
3	Olive Johnson	CHK0003	320.75
4	Snuggle Patel	SAV0004	5000.00
5	Catbahadur Rai	CHK0005	125.25

o List of all the transactions and accounts



3.1.8 Cross product operation

```
SELECT *
FROM Customers, Accounts;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CustomerID	FirstName	LastName	Email	PhoneNumber	Address	DateOfBirth	RegistrationDate	AccountID
5	Catbahadur	Rai	catbahadur.rai@example.com	5678901234	654 Cedar St, Parkland	1998-05-05	2024-12-28 11:33:30	1
4	Snuggle	Patel	snuggle.patel@example.com	4567890123	321 Pine St, Hamletburg	1996-04-04	2024-12-28 11:33:30	1
3	Olive	Johnson	olive.johnson@example.com	3456789012	789 Oak St, Villagetown	1994-03-03	2024-12-28 11:33:30	1
2	Amrit	Kaur	amrit.kaur@example.com	2345678901	456 Elm St, Townsville	1992-02-02	2024-12-28 11:33:30	1
1	Anmol	Singh	anmol.singh@example.com	1234567890	123 Main St, Cityville	1990-01-01	2024-12-28 11:33:30	1
5	Catbahadur	Rai	catbahadur.rai@example.com	5678901234	654 Cedar St, Parkland	1998-05-05	2024-12-28 11:33:30	2
4	Snuggle	Patel	snuggle.patel@example.com	4567890123	321 Pine St, Hamletburg	1996-04-04	2024-12-28 11:33:30	2
3	Olive	Johnson	olive.johnson@example.com	3456789012	789 Oak St, Villagetown	1994-03-03	2024-12-28 11:33:30	2
2	Amrit	Kaur	amrit.kaur@example.com	2345678901	456 Elm St, Townsville	1992-02-02	2024-12-28 11:33:30	2
1	Anmol	Singh	anmol.singh@example.com	1234567890	123 Main St, Cityville	1990-01-01	2024-12-28 11:33:30	2
5	Catbahadur	Rai	catbahadur.rai@example.com	5678901234	654 Cedar St, Parkland	1998-05-05	2024-12-28 11:33:30	3
4	Snuggle	Patel	snuggle.patel@example.com	4567890123	321 Pine St, Hamletburg	1996-04-04	2024-12-28 11:33:30	3
3	Olive	Johnson	olive.johnson@example.com	3456789012	789 Oak St, Villagetown	1994-03-03	2024-12-28 11:33:30	3
2	Amrit	Kaur	amrit.kaur@example.com	2345678901	456 Elm St, Townsville	1992-02-02	2024-12-28 11:33:30	3
1	Anmol	Singh	anmol.singh@example.com	1234567890	123 Main St, Cityville	1990-01-01	2024-12-28 11:33:30	3
5	Catbahadur	Rai	catbahadur.rai@example.com	5678901234	654 Cedar St, Parkland	1998-05-05	2024-12-28 11:33:30	4
4	Snuggle	Patel	snuggle.patel@example.com	4567890123	321 Pine St, Hamletburg	1996-04-04	2024-12-28 11:33:30	4
3	Olive	Johnson	olive.johnson@example.com	3456789012	789 Oak St, Villagetown	1994-03-03	2024-12-28 11:33:30	4
2	Amrit	Kaur	amrit.kaur@example.com	2345678901	456 Elm St, Townsville	1992-02-02	2024-12-28 11:33:30	4
1	Anmol	Singh	anmol.singh@example.com	1234567890	123 Main St, Cityville	1990-01-01	2024-12-28 11:33:30	4
5	Catbahadur	Rai	catbahadur.rai@example.com	5678901234	654 Cedar St, Parkland	1998-05-05	2024-12-28 11:33:30	5
4	Snuggle	Patel	snuggle.patel@example.com	4567890123	321 Pine St, Hamletburg	1996-04-04	2024-12-28 11:33:30	5
3	Olive	Johnson	olive.johnson@example.com	3456789012	789 Oak St, Villagetown	1994-03-03	2024-12-28 11:33:30	5
2	Amrit	Kaur	amrit.kaur@example.com	2345678901	456 Elm St, Townsville	1992-02-02	2024-12-28 11:33:30	5

3.1.9 Selection operation

```
SELECT *
FROM Customers
WHERE FirstName LIKE 'S%';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

CustomerID	FirstName	LastName	Email	PhoneNumber	Address	DateOfBirth	RegistrationDate
4	Snuggle	Patel	snuggle.patel@example.com	4567890123	321 Pine St, Hamletburg	1996-04-04	2024-12-28 11:33:30
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3.1.10 Projection operation

```
1 •  SELECT AccountNumber, Balance
2    FROM Accounts;
3
```

Result Grid | Filter Rows: | Export:

AccountNumber	Balance
CHK0001	1500.50
SAV0002	2500.00
CHK0003	320.75
SAV0004	5000.00
CHK0005	125.25

17

3.2 Normalization Technique

To demonstrate the normalization technique, let's start off with an unnormalized table:4

3.2.1 Unnormalized table

Table creation:

```
1 • CREATE TABLE UnnormalizedCustomers (
2       CustomerID INT,
3       CustomerName VARCHAR(100),
4       AccountNumber VARCHAR(20),
5       AccountType ENUM('Checking', 'Savings', 'Loan'),
6       Balance DECIMAL(15, 2)
7   );
8
```

Here, lets quickly change the name of the 'CustomerName' field to 'Customer_info':

```
ALTER TABLE UnnormalizedCustomers
CHANGE COLUMN CustomerName Customer_info VARCHAR(100);
```

Table description:

```
1 •  desc UnnormalizedCustomers;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
CustomerID	int	YES		NULL	
Customer_info	varchar(100)	YES		NULL	
AccountNumber	varchar(20)	YES		NULL	
AccountType	enum('Checking','Savings','Loan')	YES		NULL	
Balance	decimal(15,2)	YES		NULL	

Data insertion:

```
1   INSERT INTO UnnormalizedCustomers (CustomerID, Customer_info, AccountNumber, AccountType, Balance)
2   VALUES
3   (1, 'Ram Thapa', '1001', 'Checking', 1500.50),
4   (1, 'Ram Thapa', '1002', 'Savings', 2500.00),
5   (2, 'John Cena', '1003', 'Checking', 320.75),
6   (3, 'Hari Lal', '1004', 'Savings', 5000.00),
7   (3, 'Hari Lal', '1005', 'Checking', 125.25),
8   (4, 'Sita Kumari', '1006', 'Loan', 10000.00);
9
```

Show table with data:

```
1 •  select * from UnnormalizedCustomers;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CustomerID	Customer_info	AccountNumber	AccountType	Balance
1	Ram Thapa	1001	Checking	1500.50
1	Ram Thapa	1002	Savings	2500.00
2	John Cena	1003	Checking	320.75
3	Hari Lal	1004	Savings	5000.00
3	Hari Lal	1005	Checking	125.25
4	Sita Kumari	1006	Loan	10000.00

3.2.2 First Normal Form (1NF)

Rule:

- o Each column contains atomic values
- o Each row must be unique, identified by a primary key

o There must be no repeating groups or arrays in a single row

```
CREATE TABLE Customers_1NF (
    CustomerID INT,
    Customer_info VARCHAR(100),
    AccountNumber VARCHAR(20),
    AccountType ENUM('Checking', 'Savings', 'Loan'),
    Balance DECIMAL(15, 2),
    PRIMARY KEY (CustomerID, AccountNumber)
);
```

```
INSERT INTO Customers_1NF (CustomerID, Customer_info, AccountNumber, AccountType, Balance)
VALUES
(1, 'Ram Thapa', '1001', 'Checking', 1500.50),
(1, 'Ram Thapa', '1002', 'Savings', 2500.00);
```

```
select * from Customers_1NF ;
```

Execute the selected portion of the script or everything, if there is no selection

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

CustomerID	Customer_info	AccountNumber	AccountType	Balance
1	Ram Thapa	1001	Checking	1500.50
1	Ram Thapa	1002	Savings	2500.00
NULL	NULL	NULL	NULL	NULL

19

3.2.3 Second Normal Form (2NF)

Rule:

o Meet all the requirements of 1NF

o Eliminate partial dependency, meaning all non-prime attributes (non-key columns) must depend on the entire primary key, not just a part of it

```sql
-- Create the Customers table
CREATE TABLE Customers_2NF (
    CustomerID INT PRIMARY KEY,
    Customer_info VARCHAR(100)
);

-- Create the Accounts table
CREATE TABLE Accounts_2NF (
    CustomerID INT,
    AccountNumber VARCHAR(20) PRIMARY KEY,
    AccountType ENUM('Checking', 'Savings', 'Loan'),
    Balance DECIMAL(15, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customers_2NF(CustomerID)
);
```

```sql
-- Insert data into Customers_2NF
INSERT INTO Customers_2NF (CustomerID, Customer_info)
VALUES
(1, 'Ram Thapa'),
(2, 'John Cena');

-- Insert data into Accounts_2NF
INSERT INTO Accounts_2NF (CustomerID, AccountNumber, AccountType, Balance)
VALUES
(1, '1001', 'Checking', 1500.50),
(1, '1002', 'Savings', 2500.00);
```

```sql
SELECT * FROM Customers_2NF;
```

Execute the selected portion of the script or everything, if there is no selection

Result Grid | Filter Rows: | Edit: | Export/Import:

CustomerID	Customer_info
1	Ram Thapa
2	John Cena
NULL	NULL

```sql
SELECT * FROM Accounts_2NF;
```

Execute the selected portion of the script or everything, if there is no selectic

Result Grid | Filter Rows: | Edit: | Export/Import:

CustomerID	AccountNumber	AccountType	Balance
1	1001	Checking	1500.50
1	1002	Savings	2500.00
NULL	NULL	NULL	NULL

3.2.4 Third Normal Form (3NF)

Rules:

- o Meet all the requirements of 2NF
- o Eliminate transitive dependency, meaning all non-prime attributes must not depend on other non-prime attributes. They should depend only on the primary key.

(NOTE: There are no transitive dependencies in the tables after the 2NF in the above process. So 3NF is not needed. We move straight to BCNF. However, we will demonstrate 3NF with a separate example after BCNF.)

3.2.5 Boyce-Codd Normal Form (BCNF)

Rules:

- o Meet all the requirements of 3NF
- o Every determinant (an attribute on which other attributes depend) must be a candidate key

```
CREATE TABLE Customers_BCNF (
    CustomerID INT PRIMARY KEY,
    Customer_info VARCHAR(100)
);
```

```
CREATE TABLE Accounts_BCNF (
    CustomerID INT,
    AccountNumber VARCHAR(20) PRIMARY KEY,
    AccountType ENUM('Checking', 'Savings', 'Loan'),
    Balance DECIMAL(15, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customers_BCNF(CustomerID)
);
```

```
INSERT INTO Customers_BCNF (CustomerID, Customer_info)
VALUES
(1, 'Ram Thapa'),
(2, 'John Cena');
```



```
INSERT INTO Accounts_BCNF (CustomerID, AccountNumber, AccountType, Balance)
VALUES
(1, '1001', 'Checking', 1500.50),
(1, '1002', 'Savings', 2500.00);
```

3.2.6 Demonstration of 3NF

Start with a table that is in 2NF:

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100),
    Email VARCHAR(100),
    PhoneNumber VARCHAR(15),
    Address TEXT,
    DateOfBirth DATE
);
```

```
CREATE TABLE Accounts (
    AccountID INT PRIMARY KEY,
    CustomerID INT,
    AccountType ENUM('Checking', 'Savings', 'Loan') NOT NULL,
    AccountNumber VARCHAR(20) UNIQUE NOT NULL,
    Balance DECIMAL(15, 2) DEFAULT 0.00,
    CreatedDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Status ENUM('Active', 'Dormant', 'Closed') DEFAULT 'Active',
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

```sql
CREATE TABLE Transactions (
    TransactionID INT PRIMARY KEY,
    AccountID INT,
    TransactionType ENUM('Deposit', 'Withdrawal', 'Transfer') NOT NULL,
    Amount DECIMAL(15, 2),
    TransactionDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);
```

Now we move to convert the 2NF table to 3NF:

```sql
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100),
    Email VARCHAR(100),
    PhoneNumber VARCHAR(15),
    Address TEXT,
    DateOfBirth DATE
);
```

```sql
CREATE TABLE Accounts (
    AccountID INT PRIMARY KEY,
    CustomerID INT,
    AccountType ENUM('Checking', 'Savings', 'Loan') NOT NULL,
    AccountNumber VARCHAR(20) UNIQUE NOT NULL,
    Balance DECIMAL(15, 2) DEFAULT 0.00,
    CreatedDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Status ENUM('Active', 'Dormant', 'Closed') DEFAULT 'Active',
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

```sql
CREATE TABLE Transactions (
    TransactionID INT PRIMARY KEY,
    AccountID INT,
    TransactionType ENUM('Deposit', 'Withdrawal', 'Transfer') NOT NULL,
    Amount DECIMAL(15, 2),
    TransactionDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);
```

```
INSERT INTO Customers (CustomerID, CustomerName, Email, PhoneNumber, Address, DateOfBirth)
VALUES
(1, 'John Doe', 'john.doe@example.com', '1234567890', '123 Elm St', '1985-05-15'),
(2, 'Jane Smith', 'jane.smith@example.com', '0987654321', '456 Oak St', '1990-08-22');
```

```
INSERT INTO Accounts (AccountID, CustomerID, AccountType, AccountNumber, Balance, Status)
VALUES
(1, 1, 'Checking', '100000001', 1500.50, 'Active'),
(2, 1, 'Savings', '100000002', 3000.00, 'Active'),
(3, 2, 'Checking', '100000003', 500.00, 'Dormant');
```

```
INSERT INTO Transactions (TransactionID, AccountID, TransactionType, Amount)
VALUES
(1, 1, 'Deposit', 1500.00),
(2, 1, 'Withdrawal', 500.00),
(3, 2, 'Deposit', 3000.00),
(4, 3, 'Deposit', 500.00);
```

3.3 Stored procedure and Transaction process

3.3.1 Stored procedure

```
DELIMITER $$

CREATE PROCEDURE TransferFunds(
    IN senderAccountID INT,
    IN receiverAccountID INT,
    IN transferAmount DECIMAL(15, 2)
)
BEGIN
    DECLARE senderBalance DECIMAL(15, 2);
    DECLARE receiverBalance DECIMAL(15, 2);

    -- Get current balances for both sender and receiver
    SELECT Balance INTO senderBalance FROM Accounts WHERE AccountID = senderAccountID;
    SELECT Balance INTO receiverBalance FROM Accounts WHERE AccountID = receiverAccour

    -- Check if the sender has enough balance
    IF senderBalance >= transferAmount THEN
        -- Deduct the transfer amount from the sender's account
        UPDATE Accounts
        SET Balance = Balance - transferAmount
        WHERE AccountID = senderAccountID;

        -- Add the transfer amount to the receiver's account
        UPDATE Accounts
        SET Balance = Balance + transferAmount
        WHERE AccountID = receiverAccountID;

        -- Insert the transaction records for both sender and receiver
        INSERT INTO Transactions (AccountID, TransactionType, Amount)
        VALUES (senderAccountID, 'Withdrawal', transferAmount);
```

```
        -- Add the transfer amount to the receiver's account
        UPDATE Accounts
        SET Balance = Balance + transferAmount
        WHERE AccountID = receiverAccountID;

        -- Insert the transaction records for both sender and receiver
        INSERT INTO Transactions (AccountID, TransactionType, Amount)
        VALUES (senderAccountID, 'Withdrawal', transferAmount);

        INSERT INTO Transactions (AccountID, TransactionType, Amount)
        VALUES (receiverAccountID, 'Deposit', transferAmount);
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient funds';
    END IF;
END$$

DELIMITER ;
```

3.3.2 Transaction process

```
START TRANSACTION;

-- Step 1: Deduct the transfer amount from the sender's account
UPDATE Accounts
SET Balance = Balance - 150
WHERE AccountID = 1;

-- Step 2: Add the transfer amount to the receiver's account
UPDATE Accounts
SET Balance = Balance + 150
WHERE AccountID = 2;

-- Step 3: Record the transaction for sender (withdrawal)
INSERT INTO Transactions (AccountID, TransactionType, Amount)
VALUES (1, 'Withdrawal', 150);

-- Step 4: Record the transaction for receiver (deposit)
INSERT INTO Transactions (AccountID, TransactionType, Amount)
VALUES (2, 'Deposit', 150);

-- If no error occurs, commit the transaction
COMMIT;
```

Chapter 4. Conclusion

The Online Learning Portal database effectively handles the management of online banking services The design ensures scalability and efficient querying.

4.1 Future scope

· Implementing a frontend interface

· Adding more features like statements prerequisite.