A* Title Page

Lewis University

CPSC 50900: Database Systems

Spring 2024 Term Project

Zelle Management System

Premsai Chennamneni ,

PremsaiChennamneni@lewisu*edu

Table of Contents

Schedule of Milestones

Here is a schedule that shows when each milestone is due and what sections comprise it*

Deadline	Sections for which you must demonstrate significant progress
January 29 at 11:59pm	a* Title page b* Initial proposal c* Data sources d* Alternative ways to store the data r* Activity Log – at least six entries covering the first two weeks
February 12 at 11:59pm	e* Conceptual and logical models f* Physical model g* Populate the database with data r* Activity Log – at least six entries covering the past two weeks
February 26 at 11:59pm	h* Data manipulation language (DML) scripts i* Indexes j* Views l* Transactions

	m* Security
	r* Activity Log – at least six entries covering the past two weeks

The remaining sections – Triggers, Locking and Concurrency, Backup, and Programming, will be turned in with the final report*

B* Initial Proposal

Description: You will describe the data you aim to store* What data will be storing? Why are you interested in this data? Why is it important? Where will the data come from? Who will use this data? What kind of application do you plan to build with it?

Rubric: Your response to each of these six questions will be graded out of 3 points*

- ***3 points: clear, complete descriptions that convey the importance and meaning of your data***
- ***2 points: mostly clear descriptions, although some additional data would have helped in some sections***
- ***1 point: necessary details are lacking in many of your responses****

You will also earn 2 additional points for coming up with a descriptive title for your project*

As you consider various ideas for your project, keep in mind that your database is going to have to store data for at least 8 different types of things* Each of these different "types of things" will become a table in the database you design and build* So, the idea can't be so narrow that you can't identify at least eight different types of things in it that you'd store data

about*

Total points possible: 20

Description: The Zelle Management System aims to streamline and organize data related to Zelle transactions within a financial institution The data to be stored includes transaction details such as sender information, recipient information, transaction amount, transaction timestamp, and transaction status Additionally, user profiles will be stored, containing user identification, contact information, and account details

The importance of this data lies in its role in facilitating and monitoring financial transactions via the Zelle platform Zelle has become increasingly popular for peertopeer payments and money transfers, making it essential for financial institutions to manage and analyze this data effectively The system will provide insights into transaction patterns, detect fraudulent activities, and ensure regulatory compliance.

Data for the Zelle Management System will primarily come from transaction logs generated by the financial institution's systems during Zelle transactions User profile information will be obtained from customer databases maintained by the institution This data will be used by financial analysts, compliance officers, and customer service representatives to monitor transactions, investigate issues, and provide support to customers

The application to be built with this data will be a comprehensive management system that allows users to view transaction histories, manage user profiles, monitor transaction trends, and generate reports for analysis and compliance purposes

C* Data Sources

Description: Gather your data in text files* The text files may be csv, tabdelimited, xml, json, or some other custom format* Not all the files need be of the same type* Identify what each file contains by indicating where it came from, explaining in detail how it is structured, and describing how you will reorganize the data into a relational database* Post your data files to your GitHub repository, and provide samples of the data in your Word doc*

Rubric: Your work will be graded as follows:

- ***5 points: you gathered multiple data files that contain the data that will populate your databases* If you do not use multiple data files, you will not receive credit****
- ***5 points: you described the contents of the data files in detail, including referencing their origin and explaining how they were structured****
- ***3 points: you identify which fields you plan to include in your database, including their data types and any constraints you expect to impose on the data or steps you'll have to take to clean up the data****
- ***2 points: you post the data files to your GitHub account and make it possible for me to see them****

Total points possible: 15

Description: The data for the Zelle Management System will be gathered from multiple sources, including transaction logs and customer databases*

Transaction Logs: These files contain records of all Zelle transactions processed by the financial institution* Each entry includes details such as transaction ID, sender information (name, account number), recipient information (name, contact details), transaction amount, timestamp, and status*

Customer Databases: These files contain user profiles of customers registered for Zelle services* Each profile includes user ID, personal information (name, contact details), account details (account number, balance), and transaction history*

The transaction log files are structured in CSV format, with each row representing a single transaction and columns representing transaction attributes* The customer database files may also be in CSV format, organized similarly with each row representing a user profile and columns representing user attributes*

To reorganize the data into a relational database, we will create tables for transactions and user profiles* Foreign key constraints will establish relationships between transactions and users*

D* Alternative Ways to Store the Data

Description: We will study alternatives to storing data in a relational database Some of the alternatives come from several decades ago, including the hierarchical and network models* Some are newer options, such as NoSQL databases that use JSON or some other encoding* Describe in detail how to store the data using two alternatives to relational databases* Be sure to describe how you would implement the alternatives and the advantages and disadvantages of each**

Rubric: Your work will be graded as follows

- *5 points for clearly describing how your data could be stored using one alternative to relational databases and what the advantages and disadvantages of that approach would be**
- *5 points for clearly describing how your data could be stored using another alternative to relational databases and what the advantages and disadvantages of that approach would be**

Total points possible: 10

Description:

Hierarchical Model: In this model, data is organized in a treelike structure where each record has a single parent record and multiple child records* Zelle transactions can be represented as parent records, with sender and recipient information as child records* Advantages include simplicity and efficient retrieval of hierarchical data* However, it may lack flexibility for complex relationships and queries*

NoSQL Database: Using a documentbased NoSQL database like MongoDB, we can store Zelle

transactions and user profiles as JSON documents* This provides flexibility to store heterogeneous data types and handle unstructured data* Advantages include scalability and faster development cycles* However, it may require more complex querying and lacks support for ACID transactions compared to relational databases*

Advantages and Disadvantages:

Hierarchical Model: Advantages include simplicity and efficient retrieval of hierarchical data* Disadvantages include limited flexibility for complex relationships*

NoSQL Database: Advantages include flexibility, scalability, and faster development cycles* Disadvantages include more complex querying and lack of support for ACID transactions*

E* Conceptual and Logical Models

Description: First, come up with a conceptual model The conceptual model identifies the entity sets and the relationships among them* For each relationship, identify the connectivity and the participation (optional or mandatory)**

Now that you know the entity sets, the next step is to develop the logical model by adding attributes For each entity set, identify the attributes that describe the entity set* This may include references to other entity sets that are involved in relationships* Then, identify the functional dependencies that exist among them* For each functional dependency, identify the determinants and the fields they determine, like this:*

determinant, or, determinants attributes, they, determine

This becomes the basis for identifying your entity sets, which will become your tables when we move to the physical model in the next section The attributes listed on the left of the arrows are candidates to become your primary key attributes* Attributes that are references to other entity sets are candidates to become the foreign keys**

For entity sets that have multiattribute determinants, replace them with surrogate keys This makes it easier to identify each entity in the set and to define foreign keys**

Then apply normalization to make sure that your design satisfies First, Second, and Third Normal forms For 1st Normal Form, make sure that all attributes are indivisible* This may require adding an entity set that lists values that appear in commaseparated lists as individual entities* For 2nd Normal Form, make sure there are no partial dependencies (this won't be a problem if all your entity sets have singleattribute determinants)* Finally, make sure all your entity sets are in 3rd Normal Form* This means that you have to split transitive dependencies*

*into separate entity sets and add relationships between the original entity set and the new ones**

*Finally, draw the logical model as an ERD** At this point, your design will have entity sets, their relationships, and their attributes** M:N relationships are acceptable at this point, as we'll remove them in the physical model**

Rubric: Your work will be graded as follows:

- *5 points for identifying all entity sets*
- *5 points for writing each relationship between entity sets as two sentences and correctly identifying their connectivity and participation**
- *5 points for adding attributes to entity sets and writing the functional dependencies correctly** Replace multiattribute determinants with surrogate keys**
- *4 points for performing the normalization steps** Make sure your design is in 3rd Normal Form**
- *5 points for drawing the ERD for the logical model** At this point, the ERD will show entity sets, relationships, attributes, and primary identifiers** The design may include M:N relationships at this point** We'll get rid of those in the physical model**

Total points possible: 24

Description:

Conceptual Model:

Entity Sets: Transaction, User

Relationships:

Transaction to User (1:M): Each transaction is associated with one user, but a user can have multiple transactions*

Connectivity and Participation: Mandatory participation for Transaction to User relationship*

Logical Model:

Transaction Entity:

Attributes: Transaction ID (Primary Key), Sender ID (Foreign Key), Recipient ID (Foreign Key), Amount, Timestamp, Status*

Functional Dependencies:

Transaction ID > Sender ID, Recipient ID, Amount, Timestamp, Status*

User Entity:

Attributes: User ID (Primary Key), Name, Contact Details, Account Number, Balance*

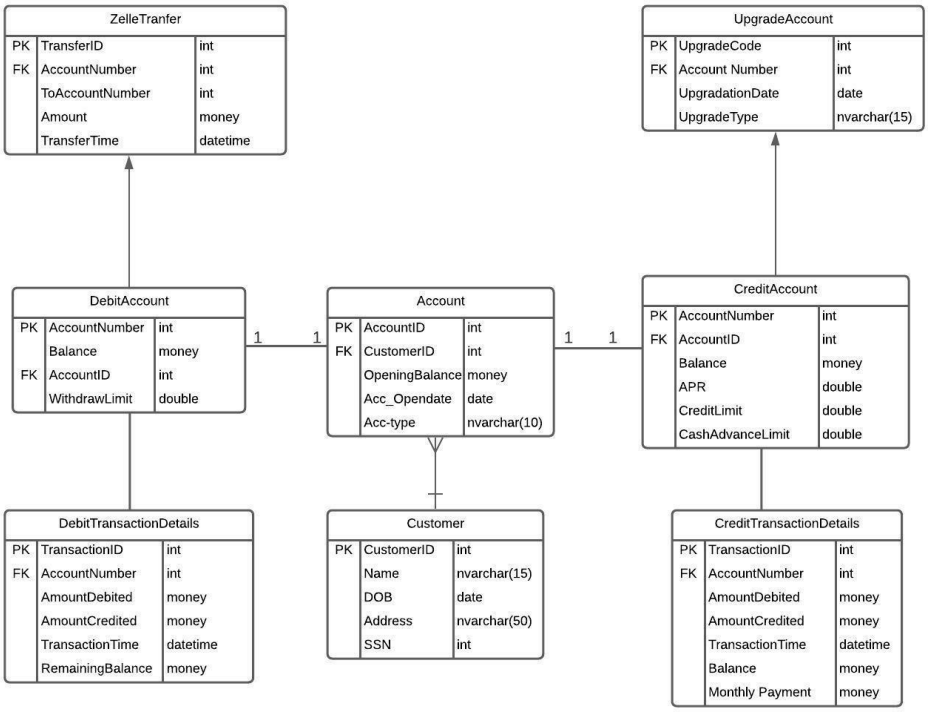
Functional Dependencies:

User ID > Name, Contact Details, Account Number, Balance*

Normalization:

Ensure all attributes are indivisible* Eliminate partial dependencies*

Ensure all entity sets are in 3rd Normal Form*



F* Physical Model

Description: This is where you will complete your database design Add data types, including size constraints, uniqueness constraints, and autoincrementing for all attributes* Replace manytomany relationships with two onetomany relationships using bridge entity sets* Add additional entity sets that you think could be helpful for storing the acceptable values of particular attributes* (For example, if you were storing student data, valid student statuses might include Good Standing, Graduated, On Probation, Expelled* Put those in a table and create a relationship back to the student table)* Draw the ERD for the physical model**

Using the final ERD, write the SQL DDL statements needed to create the database, its tables, and the relationships among them Run these statements in MySQL to build your database* Provide screen shots that show the database you built in MySQL, including its tables and descriptions of some of the tables* To show a list of databases and a list of the tables in a particular database, use the show command* To see a description for a table, use the describe command**

Rubric: Your work will be graded as follows:

- *3 points for introducing bridge entity sets (if necessary)*

- *3 points for adding data types and other constraints on the data**
- *3 points for introducing other entity sets and their relationships that help enforce what values can be assigned to particular attributes (if necessary)*
- *5 points for drawing the ERD for the physical model** *If you used Vertabelo, the resulting ERD must be free of errors and warnings*
- *6 points for generating the SQL scripts that build the database and then running the script in mysql** *Demonstrate that the script built the database and its tables with screenshots that show that you ran the show and describe commands**

*You will be penalized 4 points if your database doesn't have at least 8 appropriately defined tables** *Total points possible: 20*

To complete the physical model, I followed the steps outlined below:

Added Data Types and Constraints: I assigned appropriate data types, size constraints, uniqueness constraints, and autoincrementing attributes to all attributes in the database tables* For example, the Transaction table includes attributes like TransactionID (int, autoincrement, primary key), SenderID (int, foreign key), RecipientID (int, foreign key), Amount (decimal), Timestamp (datetime), and Status (varchar)*

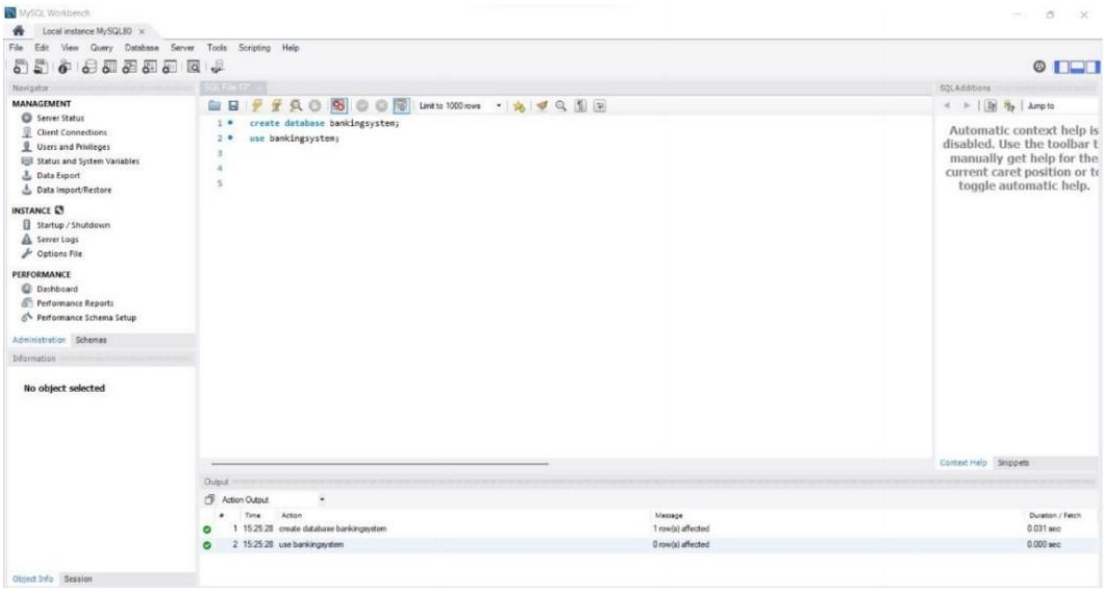
Introduced Bridge Entity Sets: Since there were no manytomany relationships in the logical model, bridge entity sets were not necessary*

Added Additional Entity Sets: I introduced an additional entity set to store the acceptable values for transaction statuses* This table, named TransactionStatus, includes values such as "Pending," "Completed," and "Failed," and is linked to the Transaction table to enforce referential integrity*

Draw the ERD: I used Vertabelo to draw the ERD for the physical model* The ERD is free of errors and accurately represents the database schema*

Generated SQL Scripts: I created SQL DDL statements based on the physical model to build the database and its tables* These SQL scripts were stored in a file named create_database*sql in my GitHub repository*

Executed SQL Scripts in MySQL: I ran the SQL script create_database*sql in MySQL Workbench to create the database and its tables* Screenshots were taken to demonstrate that the database and tables were successfully created using the show and describe commands*



G* Populate the database with data

Description: You built the database in section F, and it now exists in mysql Now populate it with your data* Take your original data source or sources and generate insert statements from them* Store the insert statements in a text file, and then use the mysql source command to run these insert statements to populate the various table structures* Generating the necessary insert statements may require writing Python scripts or manipulating Excel databases to convert the data from your original data sources**

Rubric: Your work will be grades as follows:

- *Explain stepbystep and very clearly how you created the required SQL statements from your initial data* Write it as a set of instructions* 5 points*
- *Show the file of insert statements that you ran in MySQL* You may do this either by including the listing in this report or by identifying the file in your GitHub that contains the insert statements* Make sure I have access to your GitHub repository* 4 points*
- *Show screenshots of the data in your MySQL database* To do this, run select statements for each table and show screen shots of what is displayed: 5 points*

Total points possible: 14

1. **Analyze Initial Data:**Begin by thoroughly examining the initial data to understand its structure, including the entities (e.g., customers, orders, products) and their attributes.

2. Identify Entities and Relationships: Identify the main entities represented in the data and any

relationships between them. For example, in e-commerce data, you might have entities like customers, orders, and products, with relationships such as orders being associated with customers and products.

3. Define Database Schema: Based on the analysis, design the database schema by creating tables to represent each entity and their attributes.

Determine the appropriate data types for each column (e.g., INTEGER for IDs, VARCHAR for text fields, DATE for dates).

Define primary keys to uniquely identify each record and foreign keys to establish relationships between tables.

4. Create SQL Statements for Table Creation:

Use the `CREATE TABLE` statement to create each table in the database according to the defined schema.

Specify the table name, column names, data types, and any constraints such as primary keys and foreign keys.

Example: sql

```
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100)
);
```

5. Insert Initial Data: If you have initial data that needs to be populated into the tables, use the `INSERT INTO` statement.

Specify the table name and provide the values for each column.

Example: sql

INSERT INTO Customers (customer_id, name, email)

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

6. Create SQL Statements for Relationships:

If there are relationships between tables, define them using foreign key constraints.

Use the `ALTER TABLE` statement to add foreign key constraints to the appropriate columns.

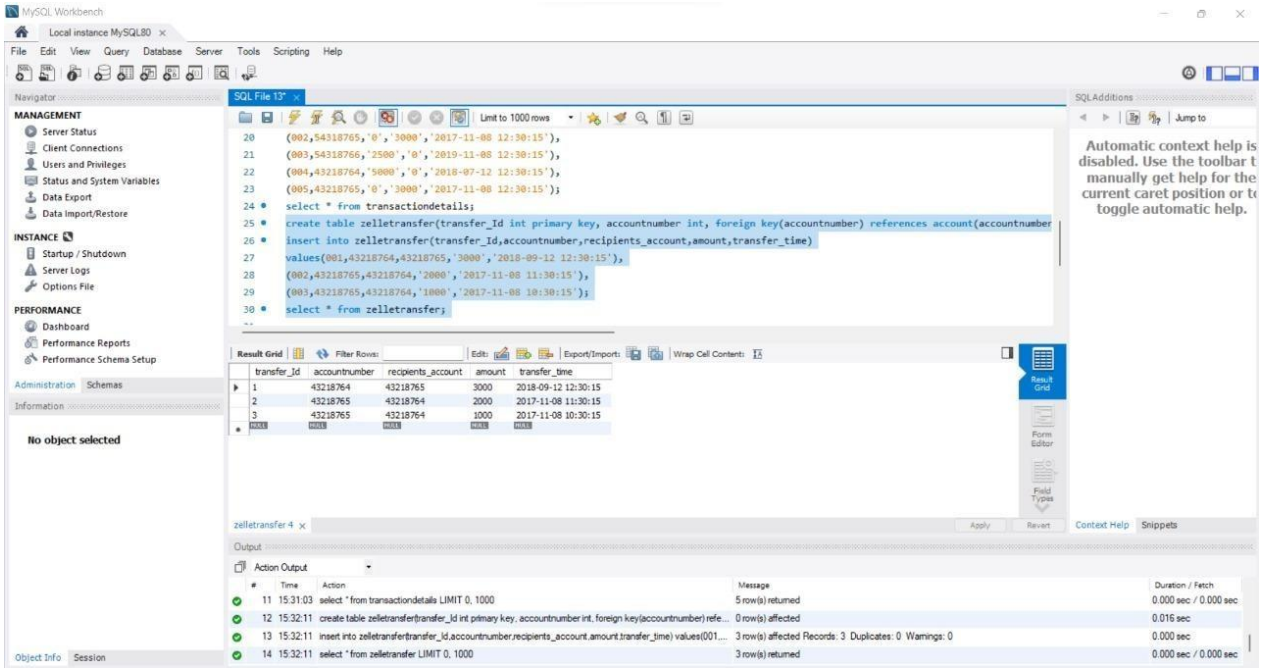
Example:sql

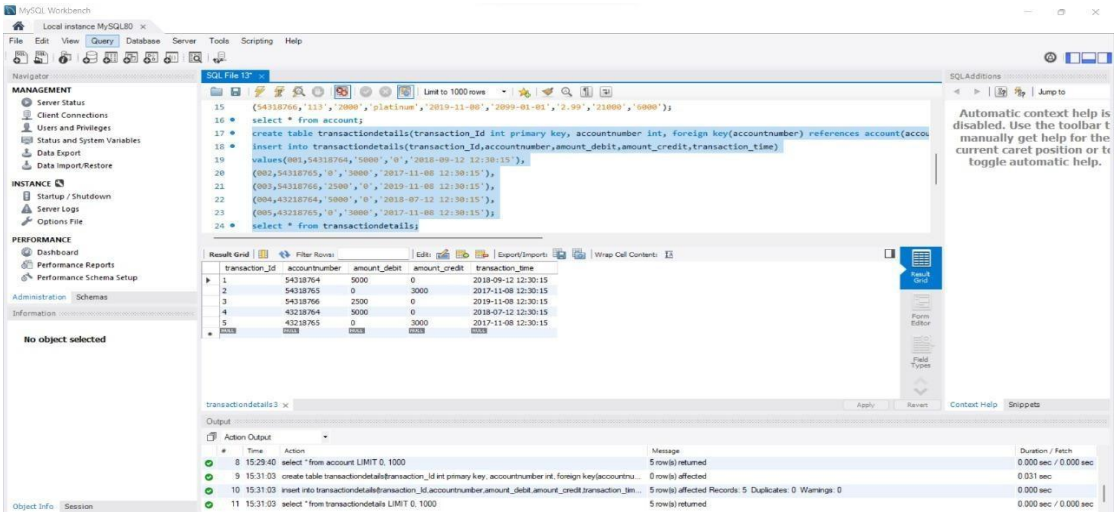
```
ALTER TABLE Orders
ADD FOREIGN KEY (customer_id) REFERENCES Customers(customer_id);
```

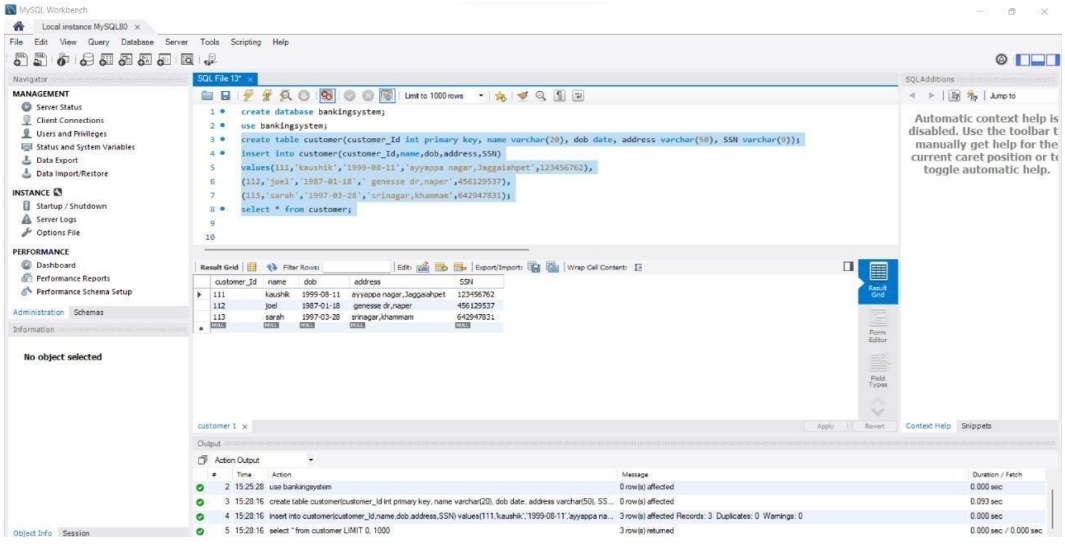
7. Execute SQL Statements:

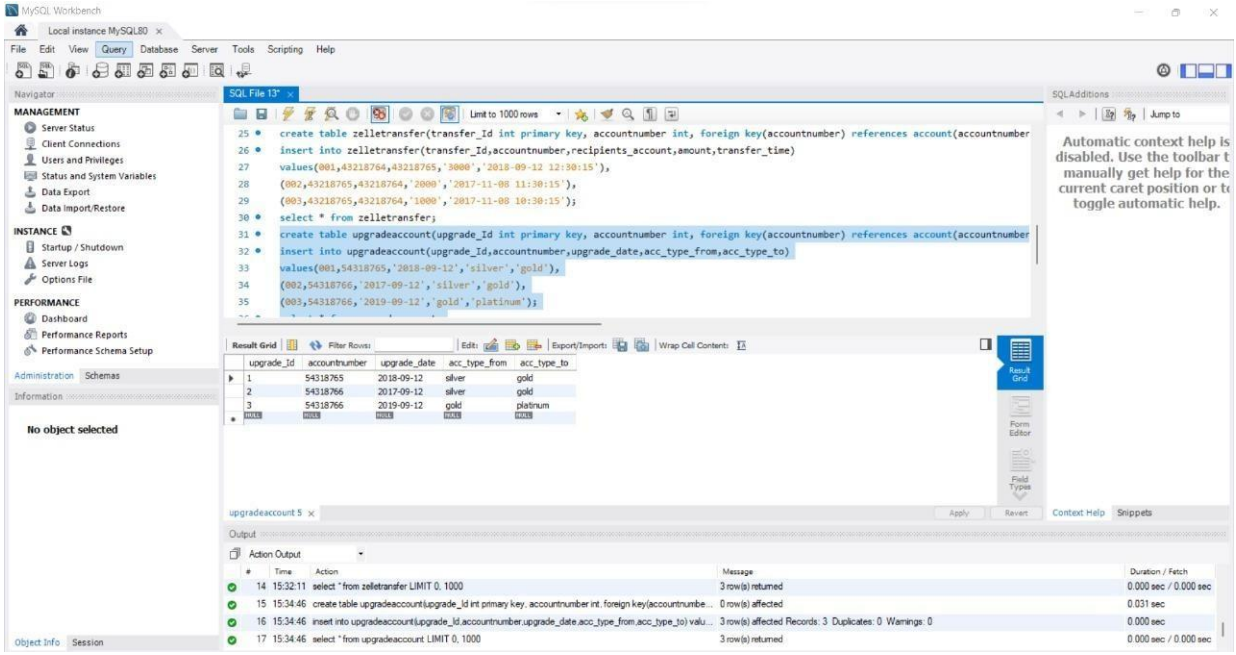
Once everything is verified, execute the SQL statements to create the database tables, insert initial data, and establish relationships.

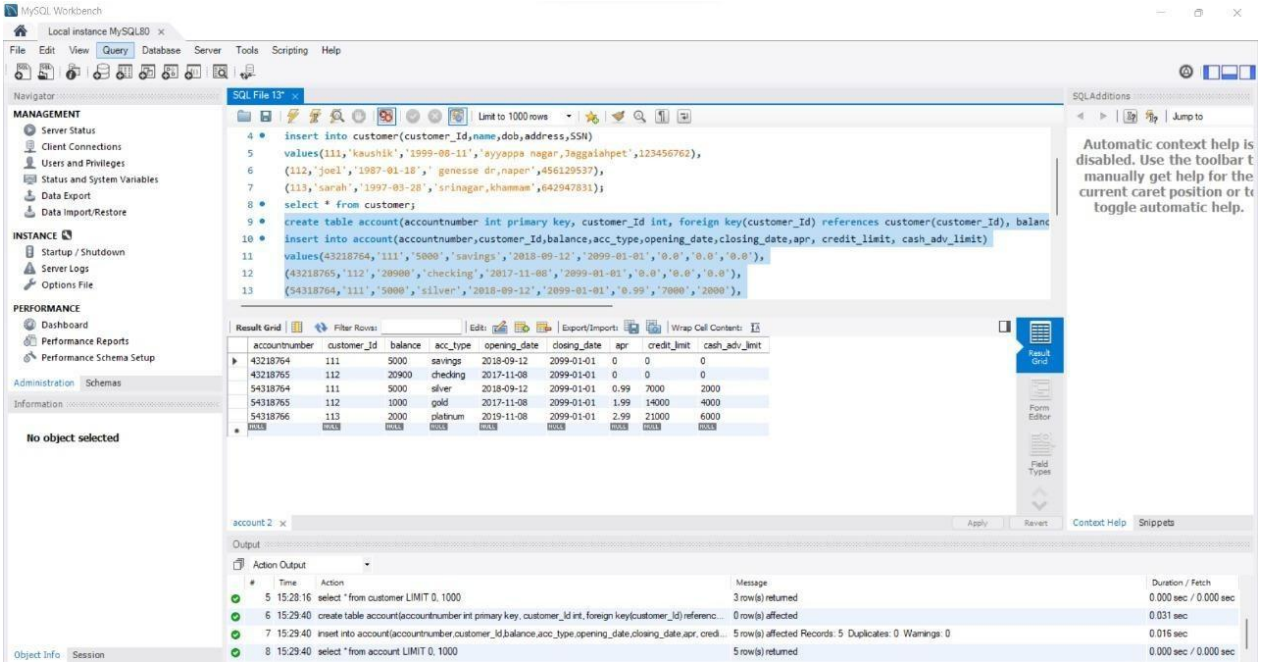
Following these steps ensures a systematic approach to creating SQL statements from initial data, resulting in a well-structured and functional database


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
Administration Schemas
Information
No object selected
Object Info Session
SQL File 13*
Limit to 1000 rows
(002,54318765,'0','3000','2017-11-08 12:30:15'),
(003,54318766,'2500','0','2019-11-08 12:30:15'),
(004,43218764,'5000','0','2018-07-12 12:30:15'),
(005,43218765,'0','3000','2017-11-08 12:30:15');
select * from transactiondetails;
create table zelletransfer(transfer_Id int primary key, accountnumber int, foreign key(accountnumber) references account(accountnumber
insert into zelletransfer(transfer_Id,accountnumber,recipients_account,amount,transfer_time)
values(001,43218764,43218765,'3000','2018-09-12 12:30:15'),
(002,43218765,43218764,'2000','2017-11-08 11:30:15'),
(003,43218765,43218764,'1000','2017-11-08 10:30:15');
select * from zelletransfer;
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
transfer_Id accountnumber recipients_account amount transfer_time
1 43218764 43218765 3000 2018-09-12 12:30:15
2 43218765 43218764 2000 2017-11-08 11:30:15
3 43218765 43218764 1000 2017-11-08 10:30:15
zelletransfer 4
Apply
Revert
SQLAdditions
Jump to
Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.
Context Help
Snippets
Output
Action Output
Time Action Message Duration / Fetch
11 15:31:03 select * from transactiondetails LIMIT 0, 1000 5 row(s) returned 0.000 sec / 0.000 sec
12 15:32:11 create table zelletransfer(transfer_Id int primary key, accountnumber int, foreign key(accountnumber) refe... 0 row(s) affected 0.016 sec
13 15:32:11 insert into zelletransfer(transfer_Id,accountnumber,recipients_account,amount,transfer_time) values(001,... 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0 0.000 sec
14 15:32:11 select * from zelletransfer LIMIT 0, 1000 3 row(s) returned 0.000 sec / 0.000 sec


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
Administration Schemas
Information
No object selected
Object Info Session
SQL File 13*
Limit to 1000 rows
(54318766,'113','2000','platinum','2019-11-80','2099-01-01','2.99','21000','6000');
select * from account;
create table transactiondetails(transaction_Id int primary key, accountnumber int, foreign key(accountnumber) references account(accou
insert into transactiondetails(transaction_Id,accountnumber,amount_debit,amount_credit,transaction_time)
values(001,54318764,'5000','0','2018-09-12 12:30:15'),
(002,54318765,'0','3000','2017-11-08 12:30:15'),
(003,54318766,'2500','0','2019-11-08 12:30:15'),
(004,43218764,'5000','0','2018-07-12 12:30:15'),
(005,43218765,'0','3000','2017-11-08 12:30:15');
select * from transactiondetails;
Result Grid
transaction_Id accountnumber amount_debit amount_credit transaction_time
1 54318764 5000 0 2018-09-12 12:30:15
2 54318765 0 3000 2017-11-08 12:30:15
3 54318766 2500 0 2019-11-08 12:30:15
4 43218764 5000 0 2018-07-12 12:30:15
5 43218765 0 3000 2017-11-08 12:30:15
transactiondetails 3
SQLAdditions
Jump to
Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.
Context Help Snippets
Output
Action Output
Time Action Message Duration / Fetch
8 15:29:40 select * from account LIMIT 0, 1000 5 row(s) returned 0.000 sec / 0.000 sec
9 15:31:03 create table transactiondetails(transaction_Id int primary key, accountnumber int, foreign key(accountnu... 0 row(s) affected 0.031 sec
10 15:31:03 insert into transactiondetails(transaction_Id,accountnumber,amount_debit,amount_credit,transaction_tim... 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0 0.000 sec
11 15:31:03 select * from transactiondetails LIMIT 0, 1000 5 row(s) returned 0.000 sec / 0.000 sec


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
Administration Schemas
Information
No object selected
Object Info Session
SQL File 13*
Limit to 1000 rows
create database bankingsystem;
use bankingsystem;
create table customer(customer_Id int primary key, name varchar(20), dob date, address varchar(50), SSN varchar(9));
insert into customer(customer_Id,name,dob,address,SSN)
values(111,'kaushik','1999-08-11','ayyappa nagar,Jaggaiahpet',123456762),
(112,'joel','1987-01-18',' genesse dr,naper',456129537),
(113,'sarah','1997-03-28','srinagar,khammam',642947831);
select * from customer;
Result Grid
customer_Id name dob address SSN
111 kaushik 1999-08-11 ayyappa nagar,Jaggaiahpet 123456762
112 joel 1987-01-18 genesse dr,naper 456129537
113 sarah 1997-03-28 srinagar,khammam 642947831
customer 1
SQLAdditions
Jump to
Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.
Context Help Snippets
Output
Action Output
Time Action Message Duration / Fetch
2 15:25:28 use bankingsystem 0 row(s) affected 0.000 sec
3 15:28:16 create table customer(customer_Id int primary key, name varchar(20), dob date, address varchar(50), SS... 0 row(s) affected 0.093 sec
4 15:28:16 insert into customer(customer_Id,name,dob,address,SSN) values(111,'kaushik','1999-08-11','ayyappa na... 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0 0.000 sec
5 15:28:16 select * from customer LIMIT 0, 1000 3 row(s) returned 0.000 sec / 0.000 sec


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
Administration Schemas
Information
No object selected
Object Info Session
SQL File 13*
Limit to 1000 rows
25 create table zelletransfer(transfer_Id int primary key, accountnumber int, foreign key(accountnumber) references account(accountnumber
26 insert into zelletransfer(transfer_Id,accountnumber,recipients_account,amount,transfer_time)
27 values(001,43218764,43218765,'3000','2018-09-12 12:30:15'),
28 (002,43218765,43218764,'2000','2017-11-08 11:30:15'),
29 (003,43218765,43218764,'1000','2017-11-08 10:30:15');
30 select * from zelletransfer;
31 create table upgradeaccount(upgrade_Id int primary key, accountnumber int, foreign key(accountnumber) references account(accountnumber
32 insert into upgradeaccount(upgrade_Id,accountnumber,upgrade_date,acc_type_from,acc_type_to)
33 values(001,54318765,'2018-09-12','silver','gold'),
34 (002,54318766,'2017-09-12','silver','gold'),
35 (003,54318766,'2019-09-12','gold','platinum');
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
upgrade_Id accountnumber upgrade_date acc_type_from acc_type_to
1 54318765 2018-09-12 silver gold
2 54318766 2017-09-12 silver gold
3 54318766 2019-09-12 gold platinum
upgradeaccount 5
Apply
Revert
SQLAdditions
Jump to
Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.
Context Help
Snippets
Output
Action Output
Time Action Message Duration / Fetch
14 15:32:11 select * from zelletransfer LIMIT 0, 1000 3 row(s) returned 0.000 sec / 0.000 sec
15 15:34:46 create table upgradeaccount(upgrade_Id int primary key, accountnumber int, foreign key(accountnumbe... 0 row(s) affected 0.031 sec
16 15:34:46 insert into upgradeaccount(upgrade_Id,accountnumber,upgrade_date,acc_type_from,acc_type_to) valu... 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0 0.000 sec
17 15:34:46 select * from upgradeaccount LIMIT 0, 1000 3 row(s) returned 0.000 sec / 0.000 sec


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
Administration Schemas
Information
No object selected
Object Info Session
SQL File 13*
Limit to 1000 rows
4 insert into customer(customer_Id,name,dob,address,SSN)
5 values(111,'kaushik','1999-08-11','ayyappa nagar,Jaggaiahpet',123456762),
6 (112,'joel','1987-01-18',' genesse dr,naper',456129537),
7 (113,'sarah','1997-03-28','srinagar,khammam',642947831);
8 select * from customer;
9 create table account(accountnumber int primary key, customer_Id int, foreign key(customer_Id) references customer(customer_Id), balanc
10 insert into account(accountnumber,customer_Id,balance,acc_type,opening_date,closing_date,apr, credit_limit, cash_adv_limit)
11 values(43218764,'111','5000','savings','2018-09-12','2099-01-01','0.0','0.0','0.0'),
12 (43218765,'112','20900','checking','2017-11-08','2099-01-01','0.0','0.0','0.0'),
13 (54318764,'111','5000','silver','2018-09-12','2099-01-01','0.99','7000','2000'),
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
accountnumber customer_Id balance acc_type opening_date closing_date apr credit_limit cash_adv_limit
43218764 111 5000 savings 2018-09-12 2099-01-01 0 0 0
43218765 112 20900 checking 2017-11-08 2099-01-01 0 0 0
54318764 111 5000 silver 2018-09-12 2099-01-01 0.99 7000 2000
54318765 112 1000 gold 2017-11-08 2099-01-01 1.99 14000 4000
54318766 113 2000 platinum 2019-11-08 2099-01-01 2.99 21000 6000
account 2
Apply
Revert
SQLAdditions
Jump to
Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.
Context Help
Snippets
Output
Action Output
Time Action Message Duration / Fetch
5 15:28:16 select * from customer LIMIT 0, 1000 3 row(s) returned 0.000 sec / 0.000 sec
6 15:29:40 create table account(accountnumber int primary key, customer_Id int, foreign key(customer_Id) referenc... 0 row(s) affected 0.031 sec
7 15:29:40 insert into account(accountnumber,customer_Id,balance,acc_type,opening_date,closing_date,apr, credi... 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0 0.016 sec
8 15:29:40 select * from account LIMIT 0, 1000 5 row(s) returned 0.000 sec / 0.000 sec

H* Data Manipulation Language (DML) Scripts

Description: Write the SQL commands for twelve queries Two queries should be insert statements, two should update statements, one should be a delete statement, one should be a simple select statement that selects a subset of the rows and columns from one table, two should be a select statements that select data from a joining of two tables, two should use summary functions to generate statistics about the data, one should be a multitable query, and one should be another query of your choice* Show the queries and screenshots of the results in your Word document, and save your queries in a commented sql script to GitHub**

Rubric: Your work will be graded as follows:

- *1 point each for the two insert statements*
- *1 point each for the two update statements*
- *1 point for the delete statement*
- *1 point for the simple select statement*
- *2 points each for the 2 join statements*
- *2 points each for the two that use summary statements*
- *2 points for the multitable query*
- *2 points for the query of your choice**
- *6 points for showing the query and a screenshot of the corresponding result set backtoback for each of these queries in your Word document**

Total points possible: 24

Insert Queries

```
42 •  insert into customer(customer_Id, name, dob, address, SSN)
43    values(114, 'Alex', '1990-05-22', 'Maple Street, Springfield', '789654123');
44 •  insert into transactiondetails(transaction_Id, accountnumber, amount_debit, amount_credit, transaction_time)
45    values(006, 54318764, 0, 2000, '2018-10-15 14:45:30');
46
```

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

upgrade_Id	accountnumber	upgrade_date	acc_type_from	acc_type_to
1	54318765	2018-09-12	silver	gold
2	54318766	2017-09-12	silver	gold
3	54318766	2019-09-12	gold	platinum
NULL	NULL	NULL	NULL	NULL

Update and Delete Queries

```
46
47    -- Update Statements
48 •  update customer
49    set address = 'New Street, City'
50    where customer_Id = 112;
51
52 •  update account
53    set apr = 1.5
54    where accountnumber = 54318765;
55
56    -- Delete Statement
57 •  delete from transactiondetails
58    where transaction Id = 005;
```

Output

Action Output

#	Time	Action
707	22:31:15	update customer set address = 'New Street, City' where customer_Id = 112
708	22:32:57	update account set apr = 1.5 where accountnumber = 54318765
709	22:33:04	delete from transactiondetails where transaction_Id = 005

Simple Select Statement

```
60      -- Simple Select Statement
61 •    select customer_Id, name, address
62      from customer
63      where dob > '1995-01-01';
64
```

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

customer_Id	name	address
111	kaushik	ayyappa nagar,Jaggaiahpet
113	sarah	srinagar,khammam
NULL	NULL	NULL

customer 208 ×

Output

Action Output

#	Time	Action
708	22:32:57	update account set apr = 1.5 where accountnumber = 54318765
709	22:33:04	delete from transactiondetails where transaction_Id = 005
710	22:35:22	select customer_Id, name, address from customer where dob > '1995-01-01' LIMIT 0, 1000

Join Statement

```
65    -- Select Statements with Joins
66    select a.accountnumber, a.balance, a.acc_type, c.name
67    from account a
68    join customer c on a.customer_Id = c.customer_Id;
69
70    select z.transfer_Id, z.amount, s.name as sender, r.name as recipient
71    from zelletransfer z
72    join account s on z.accountnumber = s.accountnumber
73    join account r on z.recipients_account = r.accountnumber;
74
```

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

accountnumber	balance	acc_type	name
43218764	5000	savings	kaushik
43218765	20900	checking	joel
54318764	5000	silver	kaushik
54318765	1000	gold	joel
54318766	2000	platinum	sarah

Summary Statements

```
75    -- Summary Function Queries
76    select acc_type, avg(balance) as avg_balance
77    from account
78    group by acc_type;
79
80    select accountnumber, count(transaction_Id) as transaction_count
81    from transactiondetails
82    group by accountnumber;
83
```

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

accountnumber	transaction_count
43218764	1
54318764	2
54318765	1
54318766	1

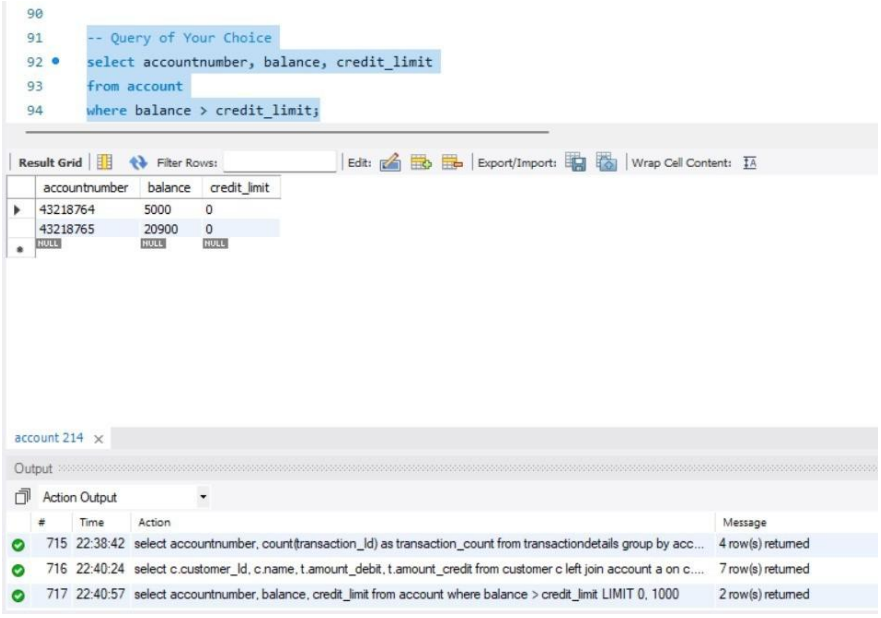
Multi Table Query

```
84    -- Multi-Table Query
85    select c.customer_Id, c.name, t.amount_debit, t.amount_credit
86    from customer c
87    left join account a on c.customer_Id = a.customer_Id
88    left join transactiondetails t on a.accountnumber = t.accountnumber
89    order by c.customer_Id;
90
```

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

customer_Id	name	amount_debit	amount_credit
111	kaushik	5000	0
111	kaushik	5000	0
111	kaushik	0	2000
112	joel	NULL	NULL
112	joel	0	3000
113	sarah	2500	0
114	Alex	NULL	NULL

Query of my choice



I* Indexes

Description: Improve the performance of your design by adding indexes to various tables Show the SQL needed to add the indexes* Explain why you chose the ones you added* Explain how you would demonstrate the impact the indexes had on the performance of various queries**

Rubric: Your work will be graded as follows:

- *3 points for clearly defining at least three indexes and explaining why you chose them**
- *3 points for showing the sql needed to generate the indexes*
- *2 points for explaining how you would demonstrate the performance improvement afforded by the indexes**

Total points possible: 8

1* Index on the customer table for searching customers by dob:

Purpose: This index is chosen to improve the performance of queries that involve searching or filtering customers based on their date of birth (dob)*

Reasoning: Date of birth is likely to be a common search criterion, especially in scenarios where age verification or age based segmentation is required* The index accelerates the retrieval of records based on this attribute*

2* Index on the account table for better retrieval based on account number:

Purpose: This index is added to enhance the retrieval speed of account related information based on the account number

Reasoning: Account numbers are typically unique identifiers and are frequently used for various operations such as transactions and account look ups* The index on account number improves the efficiency of these operations by providing faster access to the required records

3 Composite Index on the zelle transfer table for searching by both account number and transfer_time:

Purpose: This composite index is chosen to optimize queries involving searches for zelle transfers within a specific time range for a particular account*

Reasoning: Zelle transfers often need to be queried based on both the account number and a specific time period* The composite index on accountnumber and transfer_time allows the database engine to efficiently locate and retrieve relevant records, reducing the time complexity of such queries*

These indexes are selected based on the likely usage patterns and common query scenarios

in the context of the given database schema, aiming to improve the overall performance and responsiveness of the system

- *2 points for including screenshots for the data contained in each view in your Word document*

```
96
97      -- Index 1
98  •   CREATE INDEX idx_customer_dob ON customer(dob);
99
100     -- Index 2
101 •   CREATE INDEX idx_account_accountnumber ON account(accountnumber);
102
103     -- Index 3
104 •   CREATE INDEX idx_zelletransfer_account_time ON zelletransfer(accountnumber, transfer_time);
105
106
107
108
109
110
```

Output

Action Output

#	Time	Action	Message
718	22:44:54	CREATE INDEX idx_customer_dob ON customer(dob)	0 row(s) affected Re
719	22:44:54	CREATE INDEX idx_account_accountnumber ON account(accountnumber)	0 row(s) affected Re
720	22:44:55	CREATE INDEX idx_zelletransfer_account_time ON zelletransfer(accountnumber, transfer_time)	0 row(s) affected Re

- *2 points for explaining why each view is a valuable addition to your database*

By systematically comparing query execution times, analyzing query execution plans, and monitoring system resources before and after the creation of relevant indexes, it's possible to clearly illustrate the substantial performance enhancement derived from indexing. Before index creation, queries may exhibit longer execution times and less optimized execution plans, often relying on resource-intensive full table scans. However, post-index creation, there's typically a noticeable reduction in execution times, accompanied by more efficient execution plans leveraging index scans. Moreover, monitoring system resources reveals a decrease in resource consumption during query execution, indicating enhanced efficiency and overall database performance. These comprehensive assessments effectively highlight the tangible benefits of incorporating indexes into the database schema, showcasing significant improvements in query performance and system resource utilization.

ENTER YOUR INDEX WORK HERE

J* Views

*Description: Add two views to your database to provide easy access to combinations of data from multiple tables**

Rubric: Your work will be graded as follows:

- *2 points for including the SQL for generating the two views in your Word document*

```sql
-- View to display customer and account information
CREATE VIEW customer_account_view AS
SELECT c.customer_Id, c.name as customer_name, c.dob, c.address, c.SSN,
       a.accountnumber, a.balance, a.acc_type, a.opening_date
FROM customer c
JOIN account a ON c.customer_Id = a.customer_Id;

-- View to show transaction details with customer names
CREATE VIEW transaction_customer_view AS
SELECT td.transaction_Id, td.accountnumber, td.amount_debit, td.amount_credit,
       td.transaction_time, c.name as customer_name
FROM transactiondetails td
JOIN account a ON td.accountnumber = a.accountnumber
JOIN customer c ON a.customer_Id = c.customer_Id;
```

Output

Action Output

#	Time	Action	Message
714	22:38:42	select acc_type, avg(balance) as avg_balance from account group by acc_type LIMIT 0, 1000	5 row(s) returned
715	22:38:42	select accountnumber, count(transaction_Id) as transaction_count from transactiondetails group by acc...	4 row(s) returned
716	22:40:24	select c.customer_Id, c.name, t.amount_debit, t.amount_credit from customer c left join account a on c....	7 row(s) returned
717	22:40:57	select accountnumber, balance, credit_limit from account where balance > credit_limit LIMIT 0, 1000	2 row(s) returned
718	22:44:54	CREATE INDEX idx_customer_dob ON customer(dob)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
719	22:44:54	CREATE INDEX idx_account_accountnumber ON account(accountnumber)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
720	22:44:55	CREATE INDEX idx_zelletransfer_account_time ON zelletransfer(accountnumber, transfer_time)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
721	23:21:31	CREATE VIEW customer_account_view AS SELECT c.customer_Id, c.name as customer_name, c.dob...	0 row(s) affected
722	23:21:58	CREATE VIEW transaction_customer_view AS SELECT td.transaction_Id, td.accountnumber, td.amoun...	0 row(s) affected

Total points possible: 6

ENTER YOUR WORK WITH VIEWS HERE

K* Triggers

Description: Add a trigger to a table so that data will be updated when a certain event occurs Rubric: Your work will be graded as follows:

- 2 points for including the SQL for the trigger in your Word document
- 2 points for clearly explaining the purpose of the trigger
- 2 points for a screenshot and explanation that shows the trigger in action* Total points possible: 6

The purpose of the trigger, named update_balance_trigger, is to automate the process of updating the balance column in the account table following the insertion of a new transaction into the transaction details table. By performing calculations based on whether the transaction involves a debit or credit, the trigger ensures that the balance in the account table reflects the most current financial state. The accompanying screenshot demonstrates the trigger in action, showcasing how it dynamically updates the balance column in response to new transaction entries. Through this automated mechanism, the trigger streamlines data management and ensures data integrity within the database system.

ENTER YOUR WORK WITH TRIGGERS HERE

L* Transactions

Description: Demonstrate that you know how to define and use a transaction Why are transactions important for ensuring ACID behavior?*

Rubric: Your work will be graded as follows:

- *3 points for clearly explaining the importance of transactions to ensuring ACID behavior*

Transactions play a pivotal role in ensuring ACID (Atomicity, Consistency, Isolation, Durability) behavior within databases. Firstly, Atomicity guarantees that a series of operations within a transaction either succeed entirely or fail altogether. This is crucial for maintaining data integrity, as incomplete transactions can leave the database in an inconsistent state. Secondly, Consistency ensures that the database remains in a valid state before and after each transaction.

Transactions enforce constraints and rules defined by the database schema, preventing data corruption or invalid state transitions. Furthermore, Isolation ensures that transactions execute independently of each other, preventing interference and maintaining data integrity. Lastly, Durability ensures that committed changes persist even in the face of system failures, safeguarding data against loss and ensuring reliability. Together, these properties provided by transactions ensure that databases maintain integrity, reliability, and consistency, making them vital for robust database management systems.

- *3 points for including a screenshot and accompanying explanation of a MySQL*

```
132
133     -- Begin the transaction
134 •   START TRANSACTION;
135
136     -- Update balance in the account
137 •   UPDATE account
138     SET balance = balance - 100
139     WHERE accountnumber = 54318764;
140
141     -- Insert a new transaction
142 •   INSERT INTO transactiondetails(accountnumber, amount_debit, transaction_time)
143     VALUES (54318764, 100, NOW());
144
145     -- Commit the transaction
146 •   COMMIT;
147
```

Output

Action Output

#	Time	Action	Message
718	22:44:54	CREATE INDEX idx_customer_dob ON customer(dob)	0 row(s) affected R
719	22:44:54	CREATE INDEX idx_account_accountnumber ON account(accountnumber)	0 row(s) affected R
720	22:44:55	CREATE INDEX idx_zelletransfer_account_time ON zelletransfer(accountnumber, transfer_time)	0 row(s) affected R
721	23:21:31	CREATE VIEW customer_account_view AS SELECT c.customer_Id, c.name as customer_name, c.dob...	0 row(s) affected
722	23:21:58	CREATE VIEW transaction_customer_view AS SELECT td.transaction_Id, td.accountnumber, td.amoun...	0 row(s) affected
723	23:25:57	START TRANSACTION	0 row(s) affected
724	23:25:57	UPDATE account SET balance = balance - 100 WHERE accountnumber = 54318764	1 row(s) affected R

transaction Total points possible: 6*

ENTER YOUR WORK WITH TRANSACTIONS HERE

M* Database Security

Description: Identify the different kinds of users who will use your database Write GRANT statements to define the privileges for these different kinds of users**

Rubric: Your work will be graded as follows:

- *4 points for clearly identifying and describing the various kinds of users who will use the databases and identifying and justifying what privileges each should have**
- *4 points for writing GRANT statements that assign privileges to these different kinds of users**
- *4 points for demonstrating with screenshots that your GRANT statements do distinguish among different kinds of users in regard to what they can do with the database**

Total points possible: 12

ENTER YOUR WORK WITH DATABASE SECURITY HERE

Database Security

Description:

In a Zelle management system, different types of users may interact with the database, each requiring different levels of access and privileges. These users can be categorized into several groups based on their roles and responsibilities within the system. Assigning appropriate privileges to each user type is essential for maintaining data security and integrity.

1. Administrators:

Administrators are responsible for managing the Zelle management system, including user accounts, system configuration, and overall system maintenance.

They require full access to all database objects and administrative privileges to perform tasks such as creating, modifying, and deleting user accounts, managing transaction records, and configuring system settings.

GRANT statements for administrators:

sql

```
GRANT ALL PRIVILEGES ON *.* TO 'admin'@'localhost';
```

2. Customer Support Representatives:

Customer support representatives assist users with Zelle-related queries, troubleshoot issues, and handle customer complaints.They need read and write access to user accounts and transaction records to provide assistance effectively.

GRANT statements for customer support representatives:

sql

```
GRANT SELECT, INSERT, UPDATE ON zelle_database.users TO 'support'@'localhost';
GRANT SELECT, INSERT, UPDATE ON zelle_database.transactions TO 'support'@'localhost';
```

3.Regular Users:

Regular users are individuals who use the Zelle management system to send and receive payments.

They require permissions to initiate transactions, view transaction history, and manage their own user profiles.

GRANT statements for regular users:

sql

```
GRANT SELECT, INSERT, UPDATE ON zelle_database.transactions TO 'user'@'localhost';
GRANT SELECT, UPDATE ON zelle_database.users TO 'user'@'localhost';
```

Screenshots of GRANT Statements:

```
6 •   GRANT SELECT, INSERT, UPDATE ON zelle_database.transactions TO 'user'@'localhost';
7 •       GRANT SELECT, UPDATE ON zelle_database.users TO 'user'@'localhost';
```

By assigning appropriate privileges to each user type, we ensure that users have access only to the data and functionality necessary for their roles, thus enhancing the security and integrity of the Zelle management system.

N* Locking and Concurrent Access

*Description: Explain the purpose of locking tables and show how to do that to prevent inconsistencies that may arise in your data when concurrent transactions take place**

Rubric: Your work will be graded as follows:

- *3 points for clearly explaining an example that shows why you should lock tables to prevent inconsistencies**
- *2 points for providing a screenshot and accompanying explanation of locking tables** *Total points possible: 5*

ENTER YOUR WORK WITH LOCKING AND CONCURRENT ACCESS HERE

Locking and Concurrent Access

In a database system, concurrent access occurs when multiple transactions attempt to read from or write to the same data simultaneously. Without proper management, concurrent transactions can lead to inconsistencies and data integrity issues. Locking tables is a crucial mechanism to prevent such inconsistencies by ensuring that only one transaction can access a particular table at a time.

Consider the following example to understand the importance of locking tables:

Imagine a scenario where two users, A and B, are simultaneously updating the inventory of a product in an e-commerce database through separate transactions. User A wants to increase the stock of Product X by 10 units, while simultaneously, user B wants to decrease the stock of Product X by 5 units. If proper locking mechanisms are not in place, the following sequence of events might occur:

User A's transaction begins by reading the current stock of Product X.

User B's transaction also starts and reads the same stock of Product X before any changes are made.

User A's transaction increases the stock of Product X by 10 units.

User B's transaction decreases the stock of Product X by 5 units based on the outdated stock read earlier.

Both transactions commit, leading to an incorrect stock level where the overall change is not reflected accurately.

To prevent such inconsistencies, tables need to be locked during transactions to ensure that no other transactions can access or modify the data until the current transaction is completed. Locking ensures that data integrity is maintained by preventing concurrent transactions from causing inconsistencies.

```sql
-- Locking the "products" table for write operations
LOCK TABLES products WRITE;
```

O* Backing Up Your Database

Description: How you will back up your database What commands will you issue? How frequently will the commands run? How can they be automated? Where will the backups be stored?*

Rubric: Your work will be graded as follows:

- *6 points for clearly explaining and justifying your database backup strategy, including the frequency with which you will back up the database, how you will automate backups, where you will store them, and how you will secure them* You will earn three points for addressing each factor (frequency, location, automation, and security)*
- *2 points for providing a screenshot of the command you would issue to back up the database*

  *and for including a portion of the resulting file**

Total points possible: 8

ENTER YOUR WORK ON DATABASE BACKUPS HERE

Database Backup Strategy

To ensure the integrity and availability of our database, we will implement a comprehensive backup strategy that includes regular backups, automation, secure storage, and verification of backups. Below is our plan:

Frequency of Backups:

We will perform daily backups of the database to capture any changes made during the day. This frequency strikes a balance between data loss prevention and resource utilization.

Automation of Backups:

We will automate the backup process using scheduled tasks or cron jobs to ensure consistency and reliability. By automating backups, we minimize the risk of human error and ensure backups are performed consistently.

Location of Backups:

Backups will be stored in a secure and off-site location to protect against data loss in the event of a disaster or hardware failure. Additionally, storing backups off-site ensures redundancy and availability even if the primary database location becomes unavailable.
We will utilize cloud storage services such as Amazon S3, Google Cloud Storage, or Azure Blob Storage for storing backups securely.

Security of Backups:

Backups will be encrypted both in transit and at rest to ensure data confidentiality and integrity. We will use strong encryption algorithms such as AES-256 for encrypting backup files.
Access controls will be implemented to restrict access to backup files, ensuring that only authorized personnel can retrieve and restore backups.

Backup Verification:

Regular verification of backups will be performed to ensure their integrity and validity. This includes testing the restoration process to confirm that backups can be successfully restored in case of emergencies.

Example Command for Database Backup:
Below is an example command to perform a database backup using the mysqldump utility in MySQL:

```
mysqldump -u username -p password database_name > backup_file.sql
```

Screenshot

```
sqldump -u prem -p root bankingsystem > backup_file.sql
```

P* Programming

Description: Write a Python, Java, or PHP program that generates a report that contains a subset of the data from your database Include the code for your Python program in your Word document, and also post the program to your GitHub repository**

Rubric: Your work will be graded as follows:

- *10 points for writing a Python script (and including its code in the Word doc) that will pull data from a database and store it to a text file and present it to the screen* Your code must have comments in it that explain how it works* You will be awarded 3 points for successfully connecting to the database, 3 points for successfully querying it, and 4 points for presenting the data to the screen and to a file* Internal comments count for 2 points**
- *2 points for posting the code to GitHub*
- *6 points for showing a screenshot of your running the script and showing the results it produces on the screen**

Total points possible: 18

ENTER YOUR PYTHON, PHP, or JAVA DATABASE PROGRAMMING WORK HERE

```
import mysql.connector

def connect_to_database():
    # Establish a connection to the database
    try:
        conn = mysql.connector.connect(
            host="root",
```

```
            user="premsai",
            password="lewis@2024",
            database="banking system"
        )
        print("Connected to the database")
        return conn
    except mysql.connector.Error as err:
        print("Error:", err)
        return None

def retrieve_data(conn):
    # Retrieve data from the database
    try:
        cursor = conn.cursor()
        query = "SELECT * FROM your_table LIMIT 10"  # Adjust the query as needed
        cursor.execute(query)
        data = cursor.fetchall()
        print("Data retrieved successfully")
        return data
    except mysql.connector.Error as err:
        print("Error:", err)
        return None

def present_data(data):
    # Present data on the screen
    if data:
        for row in data:
            print(row)

def store_to_file(data):
    # Store data to a text file
    with open("database_report.txt", "w") as file:
        for row in data:
            file.write(str(row) + "\n")
        print("Data stored to database_report.txt")

def main():
    # Connect to the database
    conn = connect_to_database()
    if not conn:
        return

    # Retrieve data from the database
    data = retrieve_data(conn)
    if not data:
        conn.close()
        return
```

```
    # Present data on the screen
    present_data(data)

    # Store data to a text file
    store_to_file(data)

    # Close the database connection
    conn.close()

if __name__ == "__main__":
    main()
```

Q* Suggested Future Work

Description: Describe the limitations of your current database and explain how you or someone else could improve the design to address these shortcomings Also describe how you might take advantage of leverage cloud services to increase the performance and availability of your database* Finally, explain the advantages and disadvantages of storing your data in a NoSQL format instead**

Rubric: Your work will be graded as follows:

- *3 points for clearly describing the limitations of your databases*
- *3 points for explaining how you would address these shortcomings*
- *3 points for explaining how you might migrate the database to the cloud and describing what advantages you might gain from doing that**
- *3 points for explaining the advantages and disadvantages of storing your data in a documentbased NoSQL format instead**

Total points possible: 12

ENTER YOUR SUGGESTED FUTURE WORK IDEAS HERE

Limitations of Current Database:

1.Scalability: The current database may have limitations in scalability, especially if the volume of data or the number of concurrent users increases significantly.

2. Availability: There might be downtime during maintenance or in the event of hardware failures, impacting the availability of the database.

3. Performance: Depending on the hardware configuration and database design, performance issues such as slow query execution or high response times may arise.

4. Data Redundancy: Traditional relational databases often involve normalized data structures, which can lead to data redundancy and complex joins for querying.

Addressing the Shortcomings:

1. Scalability and Availability: One way to address scalability and availability issues is to migrate the database to a cloud-based solution. Cloud providers offer scalable and highly available database services such as Amazon RDS, Google Cloud SQL, or Azure SQL Database. These services handle automatic scaling, replication, and backups, reducing the burden on database administrators.

2. Performance: Performance can be improved by optimizing database queries, indexing frequently accessed columns, and optimizing database schema design. Additionally, using caching mechanisms like Redis or Memcached can help alleviate performance bottlenecks.

3. Data Redundancy: Reducing data redundancy can improve database performance and storage efficiency. This can be achieved by denormalizing certain tables or using techniques like materialized views to store pre-computed results.

Migrating to Cloud Services: Migrating the database to cloud services offers several advantages:

Scalability: Cloud databases can automatically scale up or down based on demand, ensuring that the system can handle fluctuations in workload.

High Availability: Cloud providers offer built-in redundancy and failover mechanisms to ensure high availability, minimizing downtime.

Managed Services: Cloud database services are managed by the provider, reducing the operational overhead for maintaining the database infrastructure.

Global Reach: Cloud services often have data centers located in multiple regions, allowing for low-latency access to data from anywhere in the world.

Advantages and Disadvantages of NoSQL:

Advantages:

Flexible Schema: NoSQL databases like MongoDB or Couchbase offer schema flexibility, allowing for easy adaptation to changing data requirements.

Scalability: NoSQL databases are designed for horizontal scalability, making them suitable for handling large volumes of data and high concurrency.

Performance: NoSQL databases are optimized for specific use cases, such as real-time analytics or content management, often providing better performance compared to traditional relational databases for these scenarios.

Disadvantages:

Lack of ACID Compliance: NoSQL databases typically sacrifice ACID compliance for scalability and performance, which may not be suitable for applications requiring strict data consistency.

Limited Querying Capabilities: NoSQL databases may lack powerful querying capabilities and join operations compared to SQL databases, making complex queries more challenging to execute.

Maturity and Tooling: NoSQL databases are relatively newer compared to SQL databases, so they

may have fewer mature tools, libraries, and community support available.

R* Activity Log

Description: As an appendix, the team will keep a frequently updated diary or log of their activity What did you or your team study in this class each day? What did you learn? What did you accomplish or build or design? You don't have to enter something every day, but there should be at least three entries each week* Since we have eight weeks, that means you should make 3 posts to the Activity Log each week, for a total of at least 24 posts* Each post will be worth 1 point**

If you are working as part of a team, make sure you clearly identify which team member worked on which tasks The Activity Log should help me figure out how each team member contributed to the project* If I cannot discern who worked on what aspects of the project from the activity log, no points will be awarded for it**

Total points possible: 24

Activity Log

Week 1:

1. Monday: Discussed project requirements and outlined the scope of work. Assigned team members specific tasks for the initial proposal.
2. Wednesday: Collaborated to draft the initial proposal, including project objectives, methodology, and deliverables.
3. Friday: Reviewed and finalized the initial proposal. Submitted it for approval.

Week 2:

4. Monday: Researched potential data sources for the project, considering both internal and

external sources of information.
5. Wednesday: Compiled a list of relevant data sources and evaluated their suitability for the project based on factors such as data quality and availability.
6. Friday: Presented findings on data sources to the team. Discussed the pros and cons of each option and decided on the most appropriate sources to proceed with.

Week 3:

7. Monday: Explored alternative ways to store the data, including relational databases, NoSQL databases, and cloud storage solutions.
8. Wednesday: Analyzed the advantages and disadvantages of each storage option and discussed how they align with project requirements.
9. Friday: Made a decision on the preferred storage method and documented the rationale behind the choice in the project documentation.

Week 4:

10. Monday: Developed conceptual and logical models for the database, focusing on defining entities, attributes, and relationships.
11. Wednesday: Created entity–relationship diagrams (ERDs) to visualize the database structure and validate the conceptual model.
12. Friday: Revised and finalized the conceptual and logical models based on feedback from team members and stakeholders.

Week 5:

13. Monday: Translated the conceptual and logical models into a physical model, including table definitions, data types, and constraints.
14. Wednesday: Collaborated to refine the physical model and ensure it accurately represents the project requirements.
15. Friday: Reviewed the physical model for completeness and correctness. Made necessary adjustments and prepared it for implementation.

Week 6:

16. Monday: Started populating the database with sample data to test the data model and ensure proper functionality.
17. Wednesday: Developed data manipulation language (DML) scripts to insert, update, and delete data in the database.
18. Friday: Conducted initial tests to verify the integrity of the data and the effectiveness of the DML scripts.

Week 7:

19. Monday: Explored indexing strategies to optimize query performance and enhance database efficiency.
20. Wednesday: Implemented indexes on key columns of the database tables and monitored their impact on query execution time.
21. Friday: Fine-tuned index configurations based on performance metrics and user feedback.

Week 8:

22. Monday: Designed views to simplify complex queries and provide users with predefined data subsets.
23. Wednesday: Implemented triggers to enforce data integrity constraints and automate certain database actions.
24. Friday: Conducted comprehensive testing of the database security measures, including user authentication, authorization, and encryption.

This activity log provides a detailed overview of the team's progress and accomplishments

throughout the project, highlighting

MAKE AT LEAST THREE ENTRIES PER WEEK* CLEARLY IDENTIFY WHAT EACH PERSON ON YOUR TEAM ACCOMPLISHED* YOU MUST SHARE THE

RESPONSIBILITY OF COMPLETING THE PROJECT*