CBE Bank Management

ER Diagram for Bank Database

Schemas for Bank Database

Creating Bank Database Tables Using MySQL

Overview

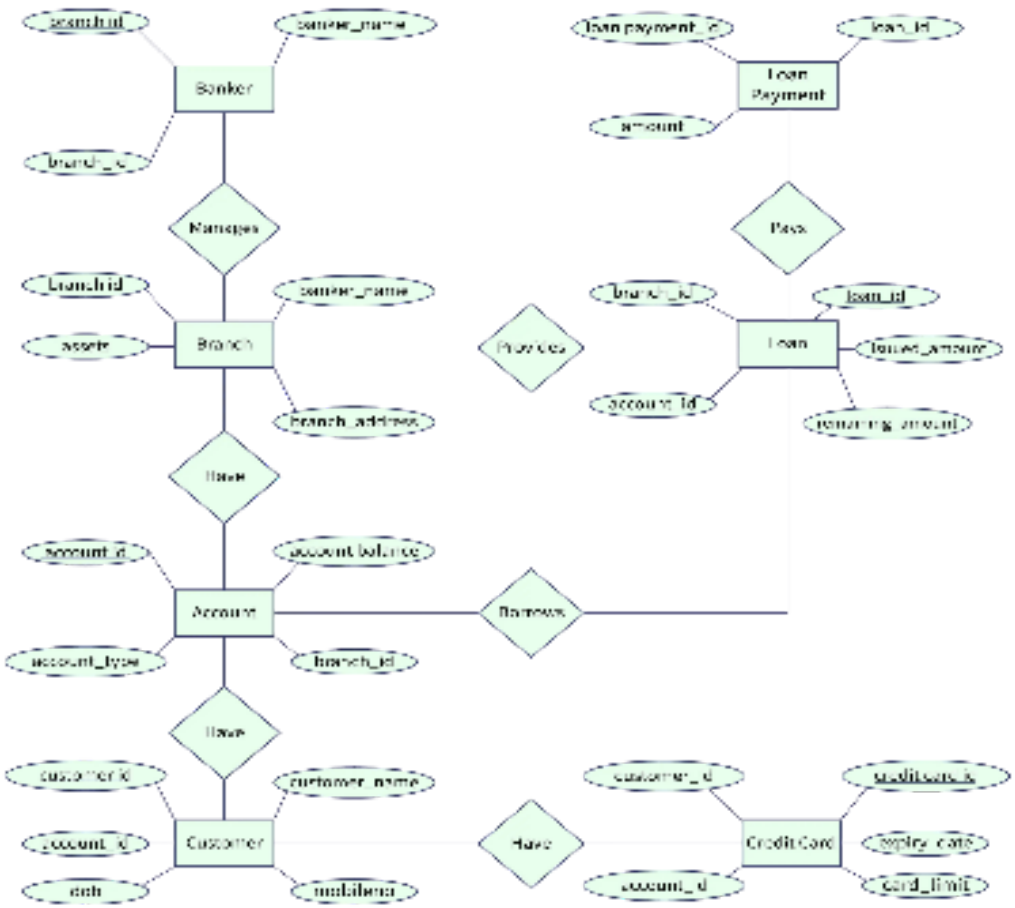
The **structured framework**, **outline**, or **plan for a database system** is created before starting working on actual implementation in the database and that is called a Schema.
It represents how the entities of the real world, which we are going to implement in the database are interlinked with each other and what are their properties.

ER Diagram for Bank Database

- An "**entity-relationship diagram**" is a kind of flowchart of a database that helps us to analyze the requirements and design of the database. It conveys the relationship between several entities of a specified system and their attributes.
- It is a basic diagrammatic structure to represent a database and is considered good to start with an ER diagram before implementing the database system.
- In this er diagram of the bank database, we have eight entities,

1. **Customer**, To represent the customers.
2. **Banker**, To represent the Banker, who manages the entire branch.
3. **Branch**, To represent a branch of a bank.
4. **Loan**, To represent the loan granted by the branch to the customer's account.
5. **Account**, To represent the bank account of any customer.
6. **Transaction**, To represent the transactions of customers for any account.
7. **Credit Card**, To represent the Credit card of any associated customer and account
8. **Loan Payment**, To represent the Payment towards the loan.

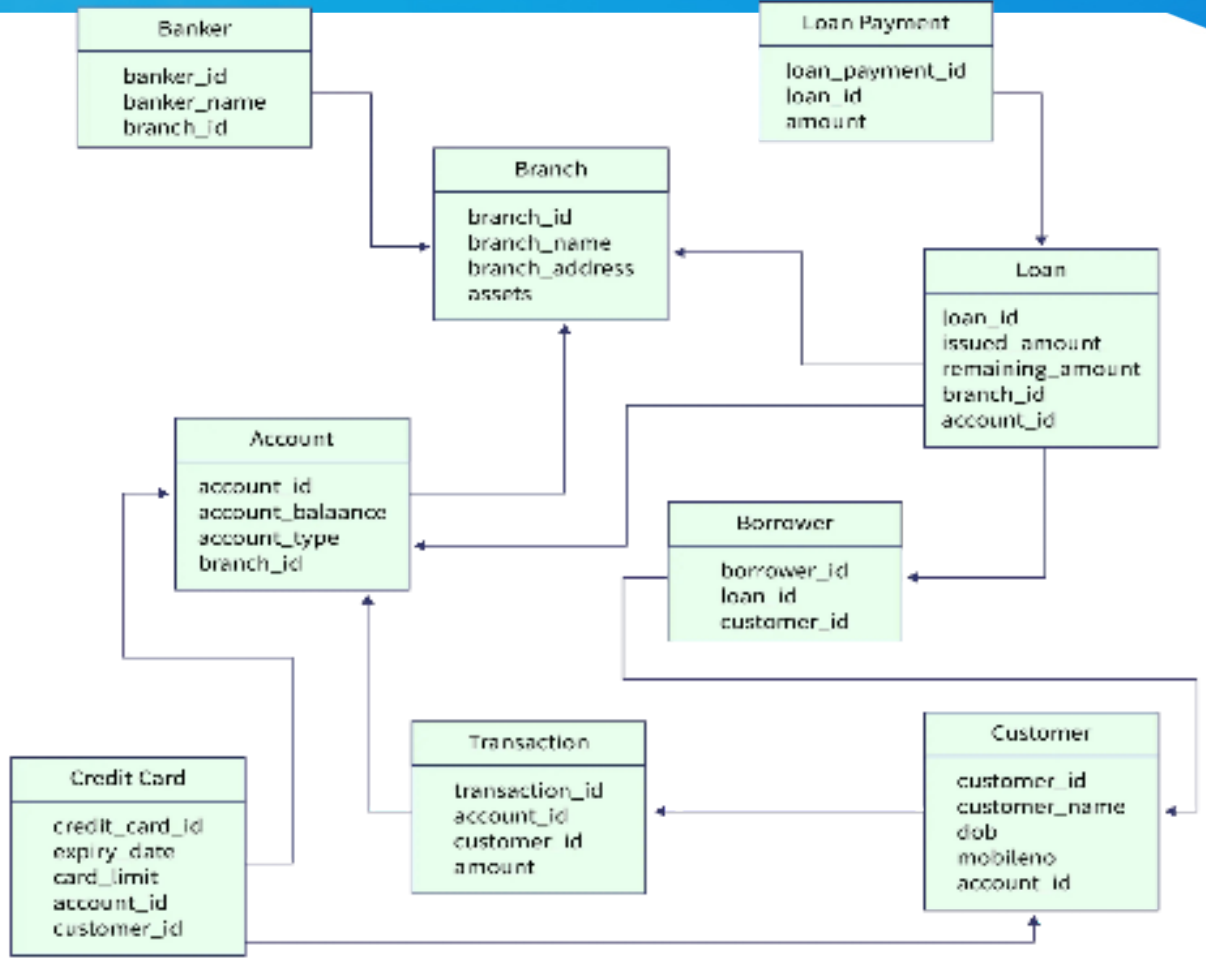
.


Banker
Manages
Branch
Have
Account
Have
Customer
Loan Payment
Pays
Loan
Provides
Borrows
Have
CreditCard

Schemas

- The schema is an analytical layout or blueprint of the database that represents the logical view, like how the tables will look in the actual implementation of the database.
- The schema diagram also represents the entity and its relationships but in the form of tables.
- Below a schema diagram is shown for the banking system. The arrow outwards the rectangular box represents the relationship between them.

SCHEMA DIAGRAM



Branch Schema

The following schemas will be designed for the banking database, We will perform sql queries for the banking database after visualizing the Schemas; it will help us to understand the system in a better way.

Branch Schema
It will represent the branches of a bank.
- **branch_id :** It is an id given to each branch to identify them uniquely.
- **branch_name :** The name of the branch.
- **branch_address :**
- **assets :** The total assets of that branch.

Field	Type
branch_id	INT
assets	INT
branch_name	TEXT
branch_address	TEXT

Account Schema

Account Schema
The account schema is for the customer's account in the bank.
•account_id:
The unique id to identify any account.
•account_balance :
The total balance in the account.
•account_type :
The type of the account.
•branch_id :
The branch with which the account is associated.

Field	Type
banker_id	INT
banker_name	TEXT
branch_id	INT

Customer Schema

This schema represents the customers of the bank.
•customer_id :
The unique id for each customer
•customer_name :
The name of the customer.
•dob :
Date of birth of the customer.
•mobileno :
The mobile number of the customer.
account_id :
The accounts associated with the particular customer

Field	Type
customer_id	INT
customer_name	TEXT
mobile_no	INT
dob	DATE
account_id	INT

Transaction Schema

This schema represents the transactions of accounts through any customers of the bank.
•transaction_id :
The unique id for each transaction.
•transaction_type :
The type of transaction i.e debit/credit.
•amount :
The amount of transaction.
•customer_id :
The customer who initiated the transaction.
•account_id :
The accounts associated with the particular customer.

Field	Type
transaction_id	INT
transaction_type	TEXT
amount	INT
customer_id	INT
account_id	INT

Loan Payment Schema

The payment for a loan will be represented by this schema, each loan payment corresponds to some amount along with the loan id.

•loan_payment_id :
The unique id of each payment towards the loan.
•loan_id :
The loan which is associated with the payment.
amount :
The amount of payment

Field	Type
loan_payment_id	INT
amount	INT
loan_id	INT

Borrower Schema

The customers who have taken any loan will be represented by this schema. Each borrower id corresponds to a loan id and associated customer along with that loan.

•borrower_id
The unique id of each borrower.
•loan_id
The loan which is associated with the borrower.
•customer_id
The id of the customer who has taken the loan.
•customer_name
The name of the customer who has taken the loan.

Field	Type
borrower_id	INT
customer_id	INT
loan_id	INT
customer_name	TEXT

Credit Card Schema

This schema represents the credit card and related details for any customer.
•credit_card_id :
Unique id to identify any credit card.
•customer_id :
The customer associated with the credit card.
•account_id :
The account id associated with the credit card.
•expiry_date :
The expiry date of the credit card.
•card_limit :
Total amount of limit of the card.

Field	Type
credit_card_id	INT
card_limit	INT
expiry_date	DATE
customer_id	INT
account_id	INT

Normalization

is a process of organizing data in a database. It involves creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and

- ## 1NF

- - The first table named "branch" is in 1NF (First Normal Form) because it has no repeating groups.
- - The second table named "banker_info" is also in 1NF because it has no repeating groups.
- - The third table named "account" is in 1NF because it has no repeating groups.
- - The fourth table named "customer" is in 1NF because it has no repeating groups.
- - The fifth table named "transaction" is in 1NF because it has no repeating groups.
- - The sixth table named "customer_credit_card" is in 1NF because it has no repeating groups.
- - The seventh table named "loan" is in 1NF because it has no repeating groups.
- - The eighth table named "loan_payment" is in 1NF because it has no repeating groups.
- - The ninth table named "borrower" is also in 1NF because it has no repeating groups.

- The code creates tables for branches, bankers, accounts, customers, transactions, customer credit cards, loans and borrowers. Each table has its own set of columns and constraints.
- The tables are not in 2NF because there are no partial dependencies in the tables. Each table has a primary key that uniquely identifies each row in the table. The primary key is used to create foreign keys in other tables to establish relationships between the tables

3NF

- - The first table named "branch" is in 3NF because all the non-key attributes are dependent only on the primary key.
- - The second table named "banker_info" is also in 3NF because all the non-key attributes are dependent only on the primary key.
- - The third table named "account" is in 3NF because all the non-key attributes are dependent only on the primary key.
- - The fourth table named "customer" is not in 3NF because the attribute "mobileno" is dependent on the customer_id and not on the primary key account_id. To achieve 3NF, we need to create a new table for customer details and move "mobileno" and "dob" to that table.
- - The fifth table named "transaction" is not in 3NF because the attribute "transaction_type" is dependent on the customer_id and not on the primary key account_id. To achieve 3NF, we need to create a new table for transaction details and move " transaction_type" to that table.
- - The sixth table named "customer_credit_card" is in 3NF because all the non-key attributes are dependent only on the primary key.
- - The seventh table named "loan" is also in 3NF because all the non-key attributes are dependent only on the primary key.
- - The eighth table named "loan_payment" is in 3NF because all the non-key attributes are dependent only on the primary key.
- - The ninth table named "borrower" is also in 3NF because all the non-key attributes are dependent only on the primary key.

Creating Bank Database Tables Using MySQL

1. *Create Database*
2. *Create Tables*
3. *Perform Querying*

Create Database

Initially, we are going to create a database.

```
CREATE DATABASE banking_system;
```

Show Databases

We can view databases with show databases statement.

```
SHOW databases;
+--------------------+
| Database           |
+--------------------+
| banking_system     |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

Switch Database

When the database is successfully created we have to switch to it for work.

```
USE banking_system;
```

Create Tables

- **Branch**

we will create tables corresponding to each entity and their relationship as described in the Schema Diagram.

This query will create a table named branch having branch id as primary key, branch name, assets, and branch address.

```
CREATE TABLE branch(
    branch_id INT NOT NULL AUTO_INCREMENT,
    branch_name VARCHAR(30) NOT NULL,
    assets INT NOT NULL,
    branch_address VARCHAR(255) NOT NULL,
    PRIMARY KEY(branch_id)
);
```

Banker Info

This query will create a table named banker_info, having banker id as a primary key, banker name, and branch id.

```
CREATE TABLE banker_info(
    banker_id INT NOT NULL AUTO_INCREMENT,
    banker_name VARCHAR(255) NOT NULL,
    branch_id INT NOT NULL,
    PRIMARY KEY (banker_id),
    FOREIGN KEY (branch_id) REFERENCES branch(branch_id)
);
```

Account

This query will create a table named account, having account id as a primary key, account type, and account balance. This table will refer to the branch table by the foreign key branch id.

```
CREATE TABLE account(
    account_id INT NOT NULL AUTO_INCREMENT,
    account_type VARCHAR(30) NOT NULL,
    account_balance INT NOT NULL,
    branch_id INT NOT NULL,
    PRIMARY KEY (account_id),
    FOREIGN KEY (branch_id) REFERENCES branch(branch_id)
);
```

Customer

The query written below will create a table named customer which will contain the customer id as primary key, customer name, mobile number, and date of birth. The account id is a foreign key that will be used to refer to the account table and will use to create an association between customers and their accounts.

```
CREATE TABLE customer(
    customer_id INT NOT NULL AUTO_INCREMENT,
    customer_name VARCHAR(30) NOT NULL,
    mobileno VARCHAR(10) NOT NULL,
    dob DATE,
    account_id INT NOT NULL,
    PRIMARY KEY (customer_id),
    FOREIGN KEY (account_id) REFERENCES account(account_id)
);
```

Transaction

The query will create a table named transaction, having transaction id as a primary key, amount, customer id, and account id. This table will refer to the account and customer table by the foreign key.

```
CREATE TABLE transaction(
    transaction_id INT NOT NULL AUTO_INCREMENT,
    amount INT NOT NULL,
    customer_id INT NOT NULL,
    account_id INT NOT NULL,
    PRIMARY KEY (transaction_id),
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (account_id) REFERENCES account(account_id)
);
```

Credit Card

The query written below will create a table named customer credit card having credit card id as a primary key, expiry date for that credit card, and card limit. The customer id and account id are foreign and will be used to refer to the associated customer and account for any particular credit card.

```
CREATE TABLE customer_credit_card(
    credit_card_id INT NOT NULL AUTO_INCREMENT,
    expiry_date DATE NOT NULL,
    card_limit INT NOT NULL,
    customer_id INT NOT NULL,
    account_id INT NOT NULL,
    PRIMARY KEY (credit_card_id),
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (account_id) REFERENCES account(account_id)
);
```

Loan

This query will create a table loan having loan id as a primary key, issued amount, and the remaining amount. The branch id will be used as a foreign key to refer to the branch that provided the loan and the account id will refer to the account on which the loan is being borrowed.

```
CREATE TABLE loan(
    loan_id INT NOT NULL AUTO_INCREMENT,
    issued_amount INT NOT NULL,
    remaining_amount INT NOT NULL,
    branch_id INT NOT NULL,
    account_id INT NOT NULL,
    PRIMARY KEY(loan_id),
    FOREIGN KEY (branch_id) REFERENCES branch(branch_id),
    FOREIGN KEY (account_id) REFERENCES account(account_id)
);
```

Borrower Table

This query will create a table named borrower which will have the borrower id as the primary key, customer id, customer name, and loan id.

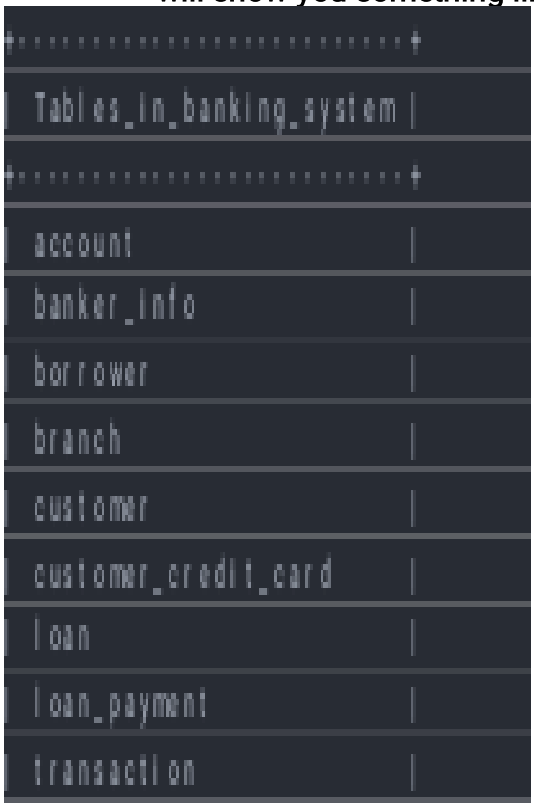
```
CREATE TABLE borrower(
    borrower_id INT NOT NULL AUTO_INCREMENT,
    customer_id INT NOT NULL,
    loan_id INT NOT NULL,
    PRIMARY KEY (borrower_id),
    FOREIGN KEY (loan_id) REFERENCES loan(loan_id),
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
);
```

Loan Payment

This query will create a table named loan_payment which will have the loan payment id as the primary key and the amount of the payment. The loan id will refer loan table and identify the loan for which payment is being done.

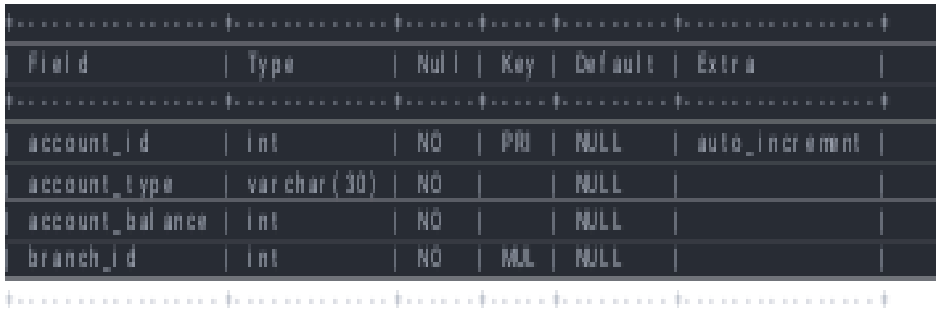
```
CREATE TABLE loan_payment(
    loan_payment_id INT NOT NULL AUTO_INCREMENT,
    amount INT NOT NULL,
    loan_id INT NOT NULL,
    PRIMARY KEY (loan_payment_id),
    FOREIGN KEY (loan_id) REFERENCES loan(loan_id)
);
```

After running all these statements, we can see our tables with the show tables; command. It will show you something like this,



Tables_in_banking_system
account
banker_info
borrower
branch
customer
customer_credit_card
loan
loan_payment
transaction

Also, we can view our schema by selecting any table, DESCRIBE banking_system.account;



Field	Type	Null	Key	Default	Extra
account_id	int	NO	PRI	NULL	auto_increment
account_type	varchar(30)	NO		NULL	
account_balance	int	NO		NULL	
branch_id	int	NO	MUL	NULL	

Perform Querying

- **Create Branches**

  In this section, we will start performing the sql queries for the banking database,

```
INSERT INTO branch (branch_name, branch_address, assets)
VALUES ('daye', 'daye city', 100000000);

INSERT INTO branch (branch_name, branch_address, assets)
VALUES ('bensa', 'bensa city', 230000000);
```

Output : If we select the all inserted data it will look like something this,

```
+-----------+-------------+-----------+-------------------------------------
----------------------------+
| branch_id | branch_name | assets    | branch_address
|
+-----------+-------------+-----------+-------------------------------------
----------------------------+
|         1 | daye        | 100000000 | daye city|
|         2 | bensa       | 230000000 | bensa city|
+-----------+-------------+-----------+-------------------------------------
----------------------------+
```

Create Accounts

To create an account we can insert data in the account table along with providing the necessary information.

```
INSERT INTO account (branch_id, account_balance, account_type)
VALUES (1, 0, 'savings');

INSERT INTO account (branch_id, account_balance, account_type)
VALUES (2, 80000, 'current');
```

Output:

```
+------------+--------------+-----------------+-----------+
| account_id | account_type | account_balance | branch_id |
+------------+--------------+-----------------+-----------+
|          1 | savings      | 0               |         1 |
|          2 | current      | 80000           |         2 |
+------------+--------------+-----------------+-----------+
```

Associate the Customer with the Account

After adding the accounts data we can create associated customers with them.

```
INSERT INTO customer (customer_name, mobileno, dob, account_id)
VALUES ('ABEBE', 988324122, '1998-01-31', 2);

INSERT INTO customer (customer_name, mobileno, dob, account_id)
VALUES ('KEBEDE', 898732112, '2000-05-14', 1);
```

Output :

```
+-------------+---------------+-----------+------------+------------+
| customer_id | customer_name | mobileno  | dob        | account_id |
+-------------+---------------+-----------+------------+------------+
|           1 | ABEBE         | 988324122 | 1998-01-31 |          2 |
|           2 | KEBEDE        | 898732112 | 2000-05-14 |          1 |
+-------------+---------------+-----------+------------+------------+
```

Perform Transactions on the Account

To perform any transaction we can insert the details of the transaction, associated account, and customer with that transaction.

After inserting data for the transaction, to keep the database consistent we will need to update the balance in the account.

```
INSERT INTO transaction(amount, transaction_type, customer_id, account_id)
VALUES (900, 'debit', 1, 2);

UPDATE account
SET account_balance = account_balance - 900
WHERE account_id=2;

INSERT INTO transaction(amount, transaction_type, customer_id, account_id)
VALUES (10000, 'credit', 1, 1);

UPDATE account
SET account_balance = account_balance + 10000
WHERE account_id=1;
```

Output :

```
Transaction Table
+----------------+--------+-------------+------------+------------------+
| transaction_id | amount | customer_id | account_id | transaction_type |
+----------------+--------+-------------+------------+------------------+
|              1 |    900 |           1 |          2 | debit            |
|              2 |  10000 |           1 |          1 | credit           |
+----------------+--------+-------------+------------+------------------+
Account Table
+------------+--------------+-----------------+-----------+
| account_id | account_type | account_balance | branch_id |
+------------+--------------+-----------------+-----------+
|          1 | savings      | 10000           |         1 |
|          2 | current      | 79100           |         1 |
+------------+--------------+-----------------+-----------+
```

Conclusion

- **Entity-relationship** diagram shows what are the entities, their attributes, and how they are interlinked.
- The schema is a logical structure that is used to analyze the organization of data in the database.
- **Schema Diagram** is a more technical structure than the ER Diagram.
- According to best practices to work with the database system, the implementation phase comes after the schema design.

THANK U!!!!!!!