Bank Management System Database

American University in Beirut – CMPS 244

Abstract

Our project's goal is to create a database management system for a fictitious bank that will store information about customers, accounts, transactions, loans, and other banking operations. The system will (if possible) provide a user-friendly interface for bank employees and customers to access and manage this information.

Entities

Our database will maintain information about various entities, including customers, accounts, transactions, loans, branches, and employees.

1. **Customer**: A person or organization who has a bank account. Name, address, contact information, and account details will be stored by the customer entity.
2. **Account**: A record of financial transactions involving a customer. Account information such as account type (e.g., savings, checking), balance, and transaction history will be stored by the account entity.
3. **Transaction**: A record of a financial transaction between a bank and a customer. The transaction entity will save information such as the transaction date, amount, and type (e.g. deposit, withdrawal, transfer).
4. **Loan**: A sum of money borrowed by a customer from a bank. Data such as the loan amount, interest rate, and repayment schedule will be saved by the loan entity.
5. **Branch**: A bank's physical location where customers can conduct transactions. The branch entity will keep information such as an address, phone number, and operating hours.

6. **Bank**: The main or parent bank oversees all branch locations. The Bank entity may include details such as the bank's name, logo, contact information, and other information about the bank's operations and management.
7. **Employee**: Tellers, customer service representatives, loan officers, and other employees work for the bank. Employee information, such as name, contact information, job title, and other details, will be stored in the Employee entity.

We might also have subclasses of Employee and a Dependent entity, depending on the specific requirements of the bank's database management system.

Relationships

Our database will also maintain information about the relationships that exist between the entities.

1. **Customer-Account**: A customer can have multiple accounts, and each account is associated with a single customer. This is a one-to-many relationship.
2. **Account-Transaction**: An account can have multiple transactions, and each transaction is associated with a single account. This is a one-to-many relationship.
3. **Customer-Loan**: A customer can have multiple loans, and each loan is associated with a single customer. This is a one-to-many relationship.
4. **Loan-Transaction**: A loan can have multiple transactions, and each transaction is associated with a single loan. This is a one-to-many relationship.

5. **Bank-Branch**: In a bank management system database project, the relationship between the Bank entity and the Branch entity is typically a one-to-many relationship, where a Bank can have multiple Branches, but each Branch is associated with only one Bank.
6. **Dependent-Employee**: The relationship between the Employee entity and the Dependent entity is a one-to-many relationship, where an Employee can have multiple Dependents, but each Dependent is associated with only one Employee.
7. **Specialized employees** (job type): Bank Teller, Technician, data analyst, branch manager.
8. **Specialized employees** (branch type): either working in main branch or secondary branches.

Constraints

We've set in place a set of constraints to ensure the integrity and validity of the data in our database.

1. **Primary Key Constraint**: A primary key constraint is used to ensure that each record in a table has a unique identifier. For example, each customer can be identified by a unique customer ID, each account by a unique account number, and each transaction by a unique transaction ID.
2. **Foreign Key Constraint**: A foreign key constraint is used to enforce referential integrity between two tables. For example, the account table can have a foreign key that references the customer ID in the customer table, ensuring that each account is associated with a valid customer.

3. **Not Null Constraint**: A not-null constraint is used to ensure that a column in a table cannot have a null value. For example, the customer's name column in the customer table cannot be null.
4. **Check Constraint**: A check constraint is used to enforce data integrity by restricting the values that can be entered into a column. For example, a check constraint can be used to ensure that the balance in an account cannot be negative.
5. **Unique Constraint**: A unique constraint is used to ensure that no duplicate values are entered into a column. For example, a unique constraint can be applied to the customer ID column in the customer table to ensure that each customer has a unique ID.
6. **Default Constraint**: A default constraint is used to provide a default value for a column in the event that a value is not specified when a new record is inserted into the table. For example, a default constraint can be used to set the status of a new account to "active" by default.

Sample Queries

- Retrieving customer information
- Retrieving account information
- Retrieving transaction history for a customer
- Retrieving loan information
- Retrieving transaction history for a loan
- Retrieving all customers who have a loan
- Updating the balance of an account
- Adding a new transaction

EER Model

Link

Relational Model

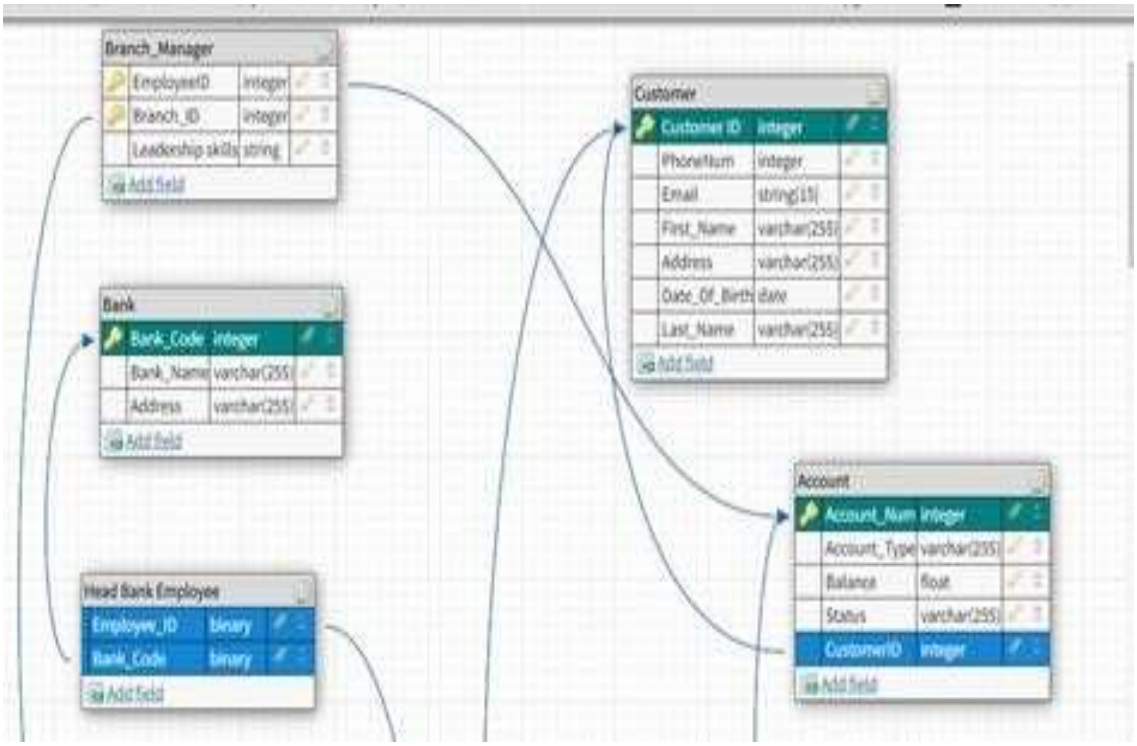
Schema:

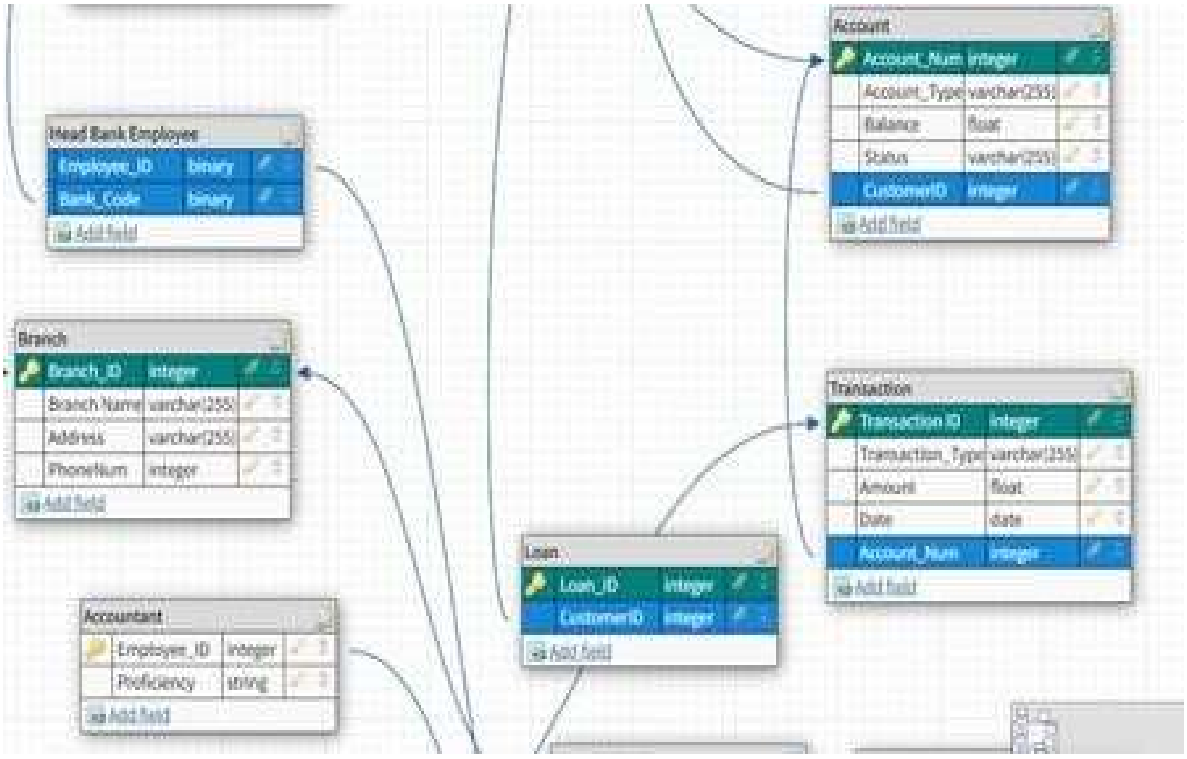

```
Employee(Employee_ID integer (primary key), Bdate date, Address varchar(255),
Jobtype varchar(255), Salary float, Name varchar(255), PhoneNum integer)

Dependent(Dependent_ID integer (primary key), D_Name varchar(255),
Relationship varchar(255), EmployeeID integer (foreign key references
Employee.Employee_ID))
```

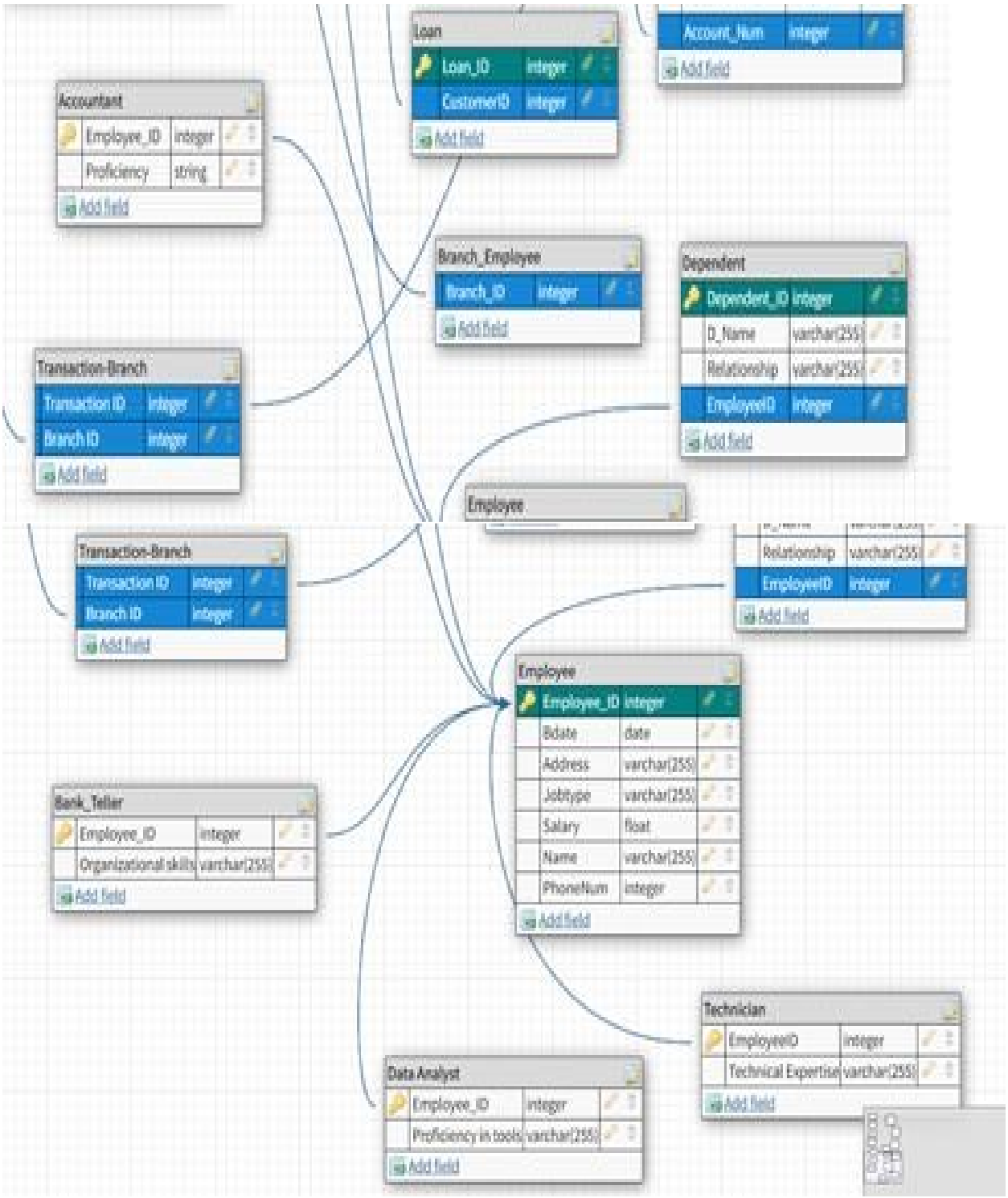
```
Accountant(Employee_ID integer (primary key, foreign key references
Employee.Employee_ID), Proficiency string)

Technician(EmployeeID integer (primary key, foreign key references
Employee.Employee_ID), Technical_Expertise varchar(255))

Data_Analyst(Employee_ID integer (primary key, foreign key references
Employee.Employee_ID), Proficiency_in_tools varchar(255))
Bank_Teller(Employee_ID integer (primary key, foreign key references
Employee.Employee_ID), Organizational_skills varchar(255))

Customer(Customer_ID integer (primary key), PhoneNum integer, Email
varchar(15), First_Name varchar(255), Last_Name varchar(255), Address
varchar(255), Date_Of_Birth date)

Bank(Bank_Code integer (primary key), Bank_Name varchar(255), Address
varchar(255))

Account(Account_Num integer (primary key), Account_Type varchar(255), Balance
float, Status varchar(255), CustomerID integer (foreign key references
Customer.Customer_ID))

Transaction(Transaction_ID integer (primary key), Transaction_Type
varchar(255), Amount float, Date date,
Account_Num integer (foreign key references Account.Account_Num))

Branch(Branch_ID integer (primary key), Branch_Name varchar(255), Address
varchar(255), PhoneNum integer)
```

```
Loan(Loan_ID integer (primary key), CustomerID integer (foreign key
references Customer.Customer_ID))

Transaction_Branch(Transaction_ID integer (foreign key references
Transaction.Transaction_ID), Branch_ID integer (foreign key references
Branch.Branch_ID))

Head_Bank_Employee(Employee_ID integer (foreign key references
Employee.Employee_ID), Bank_Code integer (foreign key references
Bank.Bank_Code))

Branch_Employee(Branch_ID integer (foreign key references Branch.Branch_ID),
Employee_ID integer (foreign key references Employee.Employee_ID))

Branch_Manager(EmployeeID integer (foreign key references
Employee.Employee_ID), Branch_ID integer (foreign key references
Branch.Branch_ID), Leadership_skills string)
```

Diagram:


Branch_Manager
EmployeeID integer
Branch_ID integer
Leadership skills string
Add field
Customer
Customer ID integer
PhoneNum integer
Email string(15)
First_Name varchar(255)
Address varchar(255)
Date_Of_Birth date
Last_Name varchar(255)
Add field
Bank
Bank_Code integer
Bank_Name varchar(255)
Address varchar(255)
Add field
Account
Account_Num integer
Account_Type varchar(255)
Balance float
Status varchar(255)
CustomerID integer
Add field
Head Bank Employee
Employee_ID binary
Bank_Code binary
Add field


Account
Account_Num integer
Account_Type varchar(255)
Balance float
Status varchar(255)
CustomerID integer
Add field
Head Bank Employee
Employee_ID binary
Bank_Code binary
Add field
Branch
Branch_ID integer
Branch Name varchar(255)
Address varchar(255)
PhoneNum integer
Add field
Transaction
Transaction ID integer
Transaction_Type varchar(255)
Amount float
Date date
Account_Num integer
Add field
Loan
Loan_ID integer
CustomerID integer
Add field
Accountant
Employee_ID integer
Proficiency string
Add field


Loan
Loan_ID integer
CustomerID integer
Add field
Account_Num integer
Add field
Accountant
Employee_ID integer
Proficiency string
Add field
Branch_Employee
Branch_ID integer
Add field
Dependent
Dependent_ID integer
D_Name varchar(255)
Relationship varchar(255)
EmployeeID integer
Add field
Transaction-Branch
Transaction ID integer
Branch ID integer
Add field
Employee
Transaction-Branch
Transaction ID integer
Branch ID integer
Add field
Relationship varchar(255)
EmployeeID integer
Add field
Employee
Employee_ID integer
Bdate date
Address varchar(255)
Jobtype varchar(255)
Salary float
Name varchar(255)
PhoneNum integer
Add field
Bank_Teller
Employee_ID integer
Organizational skills varchar(255)
Add field
Technician
EmployeeID integer
Technical Expertise varchar(255)
Add field
Data Analyst
Employee_ID integer
Proficiency in tools varchar(255)
Add field

Final Phase

Several indexes have been created to improve the performance of certain queries. Indexes are used to speed up the retrieval of data from tables by creating a separate data structure that allows for quick lookup of data based on specific columns. In this section, we will list all the indexes created and provide a justification for their inclusion.

The following indexes have been created in the script:

1. idx_CustomerID on the Account table for the CustomerID column
2. idx_AccountNum on the Transactions table for the Account_Num column
3. idx_TransactionID on the Transaction_Branch table for the Transaction_ID column
4. idx_BranchID on the Transaction_Branch table for the Branch_ID column
5. idx_EmployeeID on the Branch_Employee table for the Employee_ID column
6. idx_BranchID2 on the Branch_Employee table for the Branch_ID column

The justification for including each of these indexes is as follows:

1. idx_CustomerID: This index is created on the CustomerID column of the Account table. The purpose of this index is to speed up queries that involve filtering or sorting data based on the CustomerID column. For example, a query to retrieve all accounts belonging to a particular customer would benefit from this index.
2. idx_AccountNum: This index is created on the Account_Num column of the Transactions table. The purpose of this index is to speed up queries that involve filtering or sorting data based on the Account_Num column. For example, a query to retrieve all transactions for a particular account would benefit from this index.
3. idx_TransactionID: This index is created on the Transaction_ID column of the Transaction_Branch table. The purpose of this index is to speed up queries that involve joining the Transactions and Branch tables on the Transaction_ID column. For example, a query to retrieve all transactions that took place at a particular branch would benefit from this index.
4. idx_BranchID: This index is created on the Branch_ID column of the Transaction_Branch table. The purpose of this index is to speed up queries that involve filtering or sorting data based on the Branch_ID column. For example, a query to retrieve all transactions that took place at a particular branch would benefit from this index.
5. idx_EmployeeID: This index is created on the Employee_ID column of the Branch_Employee table. The purpose of this index is to speed up queries that involve joining the Employee and Branch tables on the Employee_ID column. For example, a query to retrieve all employees who work at a particular branch would benefit from this index.
6. idx_BranchID2: This index is created on the Branch_ID column of the Branch_Employee table. The purpose of this index is to speed up queries that involve filtering or sorting data based on the Branch_ID column. For example, a query to retrieve all employees who work at a particular branch would benefit from this index.

Views are an essential feature of SQL databases that allow users to create virtual tables based on the result set of a query. Views are used to simplify queries, improve security, and provide customized access to data for different users. In this report, I will list all the views that I have created and the justification for including these views.

Views:

1. View1: This view selects specific columns from the Customer and Account tables and joins them on the Customer_ID and CustomerID fields. The purpose of this view is to provide a list of all customers with their associated account details, including account type, balance, and status.
2. View2: This view selects specific columns from the Transactions, Account, and Customer tables and joins them on the Account_Num and Customer_ID fields. The purpose of this view is to provide a list of all transactions with the associated customer and account details, including transaction type, amount, and date.
3. v_Employee_Job_Details: This view selects specific columns from the Employee, Accountant, Technician, Data_Analyst, Bank_Teller, and Branch_Manager tables and joins them using left joins on the Employee_ID field. The purpose of this view is to provide a list of all employees and their job details, including their job type and specific proficiency or skills related to their job. The view uses a CASE statement to display the specific proficiency or skill for each job type.

Justification:

The views were created to simplify the queries for retrieving data from the database. The views allow for easier access to frequently used data and reduce the need for writing complex queries. Additionally, the views improve security by limiting the data that can be accessed by certain users. Finally, the views provide customized access to data for different users based on their specific needs.