DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By:

|  |  |
| --- | --- |
| PES2201800047 | Naik Bhavan Chandrashekhar |

|  |
| --- |
| **Summary:**  The Project is based on **Bank Management System using SQL**. It has a database which contains different tables that hold the values for Customers, Employees, Accounts and Transactions. These four tables are mapped to each other using the concept of primary and foreign keys. The customer table holds the information about the customer including a special number given to him (CIF\_No) which acts as a primary key. Using this key, we can find details about the customer and various types of accounts maintained by the person. This project is capable of throwing the details of employees who have put through the transaction and the one who has authorized it. By entering the Transaction\_ID, we can trace all the details of transaction, like the Account\_No, Amount, Balance, Type, Customer details and Employees involved in this process. |

**Table of Contents:**

|  |  |
| --- | --- |
| [Summary](#Summary) | 1 |
| [Table of Contents](#Table_of_Contents) | 2 |
| [Introduction](#Intro) | 3 |
| [Data Model](#Data_Model) | 4 |
| [Functional Dependencies and Normalization](#FD_Norm) | 5 |
| [Data Definition Language (DDL)](#DDL) | 7 |
| [Triggers](#Triggers) | 9 |
| [SQL Queries](#Query) | 15 |
| [Conclusion](#Conclusion) | 19 |
|  |  |

# **Introduction:**

As mentioned above, this project is based on ‘**Bank Management System**’. We all know that banking is one of the very important industries across the world. Maintaining confidentiality, integrity and accessibility are crucial in today’s digital world. We all come across several incidents of cyber crimes in financial institutes. Banks are also equipped with several robust integrated information security put in place to maintain the three qualities mentioned above. The amount involved in day-to-day transactions is exorbitant which warrants full proof system. I have only considered basic structure of banking in this project to understand it in a better way. Banks across the world have already connected through core banking permitting customers to put through the transaction anywhere. Our mini world is the basic bank system.

* The first entity is Customer: It contains attributes pertaining to general information about the customer, where CIF\_No is the primary key. CIF stands for ‘Customer Identification File’ which is a special ID given to a customer for the first time as he comes to a bank.
* The second entity is Employee: This also contains many different attributes related to information about an employee, where PF\_No is the primary key. PF stands for ‘Provident Fund’ which is an ID given to every employee.
* The third entity is Accounts: This entity contains all details about the accounts maintained in the bank. One customer may have more than one account. The account is mapped to customer using the CIF\_No. Here, Account\_No acts as the primary key.
* The last entity is Transactions: It contains all information related to a particular transaction. Its primary key is Transaction\_ID. We can figure out information regarding a transaction like the account involved, customer details and even the employee details who approved of this transaction.

# **Data Model:**

Bank Schema:

A screenshot of a social media post

Description automatically generated

ER Diagram:

**A close up of a map

Description automatically generated**

**Functional Dependencies and Normalization:**

There are many function dependencies considering the schema that we have chosen since each table has a primary key. We can consider a primary key or any combination of primary keys on the left side to be mapped to one column or any combination of other columns in the table.

Hence, let us consider the functional dependencies with candidate keys. One candidate key in ‘Customer’ Table is ‘Identification\_No’.

Here’s the functional dependency:

Identification\_No 🡪 {Customer\_Name, Address, Opening\_Date, Birth\_Date}

We have another candidate key in ‘Employee’ Table which is ‘Contact\_No’. i.e.

Contact\_No 🡪 {Employee\_Name, Birth\_Date, Address, Designation}

We choose Contact\_No in Employee table as candidate key but not in Customer table because a customer maybe minor and his/her contact number may match with their parents/guardians. But in the case of Employee, they are not minors and have their own unique contact number.

Considering these functional dependencies, we move on to normalization:

We check all the tables for the normalization test and check whether they satisfy it or not. The schema will be in the normal form satisfied by the least normalized table. We will find which normal form all the tables are in using their attributes.

* 1NF: Our Customer table had ‘Name’ as composite attribute but now it has been split to ‘First\_Name’ and ‘Last\_Name’. So, we do not have either composite or multi-values attributes. Also, all values belong to the domain and the column names are also unique. Hence, all our tables satisfy 1NF.
* 2NF: All our tables have a primary key which means that in every case, we can find all the other values in the tables using only one value of the primary key. Also, all our tables satisfy 1NF as mentioned above. Hence, the tables satisfy 2NF as well.
* 3NF: For two tables, Employee and Accounts, we can observe that ‘Capability’ depends on ‘Type’ and ‘Rate\_of\_Interest’ depends on ‘Acc\_Type’ respectively. i.e. a non-prime attribute depends on another non-prime attribute which says that transitive dependency exists in both of these tables. So, we say that these two tables are of 2NF form only but on the other hand, the other two tables, Customer and Transaction do not satisfy transitive dependency and satisfy the requirements of 3NF. They both satisfy BCNF (Boyce-Codd Normal Form) as well since the primary key is the only way we can get to all the other values of the tables and we cannot get other values using any other attribute.
* 4NF: We can observe that in the Customer and Transaction tables, we cannot find multi-valued dependency because in both tables, there is a primary key and there cannot be more than one rows having same value of primary key. Customer table has a candidate key as well. Hence, both satisfy 4NF as well.
* 5NF: Our customer table has a primary key and a candidate key. So, it can be decomposed into two parts. Hence, Customer Table is only in 4NF form. Now, coming to the last, Transaction table. We cannot decompose the table into two part (we can but we need to have primary key in both). Join dependency does not exist in Transaction table. So, Transaction is in 5NF.
* We come to the following conclusion based in Normalization:

1. Employee table is in 2NF
2. Accounts table is in 2NF
3. Customer table is in 4NF
4. Transactions table is in 5NF

* Therefore, we say that our schema is in 2NF i.e. The Second Normal Form.

# **DDL (Data Definition Language):**

**Creation of Tables in a schema:**

create schema Bank;

create database Bank; use Bank;

Create Table Bank.Customer(

CIF bigint not null primary key, First\_Name varchar(30) not null, Last\_Name varchar(30) not null , Address varchar(50) not null, Contact\_No bigint not null, Home\_Branch int not null ,Opening\_Date date not null, Gender char(3) not null, Birth\_Date date not null, Identification\_No varchar(15) not null unique, Constitution varchar(10) not null, check (CIF <= 999999999999), check (Contact\_No <= 99999999999), check (Home\_Branch <= 999999));

Create Table Bank.Employee(

PF\_No bigint not null primary key, Emp\_Name varchar(30) not null, Birth\_Date date not null, Designation varchar(25),

Type varchar(7) not null, Capability int not null,

Address varchar(50) not null, Contact\_No bigint not null,

Joining\_Date date not null, Gender char(1) not null,

check (PF\_No <= 99999999), check (Capability<10),

check (Contact\_No <= 99999999999));

Create Table Bank.Accounts(

Account\_No bigint not null primary key, CIF bigint not null,

Opening\_Date date not null, Acc\_Type varchar(7) not null,

Facility varchar(11) not null, Balance int not null,

Interest\_Rate float not null, check (Account\_No<=99999999999),

check (CIF<=99999999999), check(Interest\_Rate<=10.00));

Create Table Bank.Transactions(

Transaction\_ID varchar(10) not null primary key,

Account\_No bigint not null, Acc\_Type varchar(9) not null,

Amount int not null, Maker\_Id bigint not null,

Checker\_Id bigint not null, check (Account\_No <= 99999999999),

check (Maker\_Id <= 99999999), check (Checker\_Id <= 99999999));

**Definition of Foreign Keys:**

Alter Table Bank.Accounts

Add Foreign Key (CIF) References Bank.Customer(CIF);

Alter Table Bank.Transactions

Add Foreign Key (Account\_No) References Bank.Accounts(Account\_No);

Alter Table Bank.Transactions

Add Foreign Key (Maker\_Id) References Bank.Employee(PF\_No);

Alter Table Bank.Transactions

Add Foreign Key (Checker\_Id) References Bank.Employee(PF\_No);

**Some Insertion Statements (Part of DML):**

Insert Into Bank.Customer Values (12345678910,'Varun','Sharma','Mumbai',8769583768,10000,'2007-07-23','M','1987-04-30','99846572839','Individual')

Insert Into Bank.Employee Values ('5161746', 'Sahil Govekar', '1981-08-29', 'Clerk', 'Maker', '5', 'Mumbai', '9251478391', '2004-06-24', 'M');

# **Triggers:**

1. Trigger to cancel withdrawal if Amount > Balance (AFTER INSERT )

Go

create trigger transactionruleviolated

on Bank.Transactions

after insert

as

declare @Amount bigint

declare @Balance bigint

declare @Type varchar(10)

declare @ACC1 bigint

declare @Trans\_ID varchar(10)

declare @log\_action varchar(20)

set @log\_action='inserted record'

select @ACC1=t.Account\_No,@Amount=t.Amount,@Type=t.Trans\_type,@Trans\_ID=t.Transaction\_ID from inserted t

select @Balance=a.Balance from Bank.Accounts a where Account\_No=@ACC1

if(@Amount>@Balance AND @Type='Withdrawl')

insert into Bank.Withdrawal(message,Amount,Balance,Account\_No,log\_action,log\_timestamp) values ('Withdrawal Abort',@Amount,@Balance,@ACC1,@log\_action,getdate())

else if(@Type='Withdrawal')

insert into Bank.Withdrawal(message,Amount,Balance,Account\_No,log\_action,log\_timestamp) values ('Withdrawal Success',@Amount,@Balance,@ACC1,@log\_action,getdate());

else

insert into Bank.Withdrawl(message,Amount,Balance,Account\_No,log\_action,log\_timestamp) values ('Deposit Success',@Amount,@Balance,@ACC1,@log\_action,getdate());

print 'After Insert Trigger Fired'

A screenshot of a social media post

Description automatically generated

2. Trigger to update balance on transaction (INSERT TRIGGER)

Go

create trigger updatebalancetrigger

on Bank.Transactions

after insert

as

declare @Amount bigint

declare @Balance bigint

declare @Type varchar(10)

declare @ACC1 bigint

select @ACC1=t.Account\_No,@Amount=t.Amount,@Type=t.Trans\_type from inserted t

select @Balance=a.Balance from Bank.Accounts a where Account\_No=@ACC1

if(@Amount<@Balance AND @Type='Withdrawal')

set @Balance=@Balance-@Amount

if(@Type='Deposit')

set @Balance=@Balance+@Amount

update Bank.Accounts set Balance=@Balance where Account\_No=@ACC1

print 'Update Balance Trigger Fired'

**A screenshot of a social media post

Description automatically generated**

3. Trigger to select action on deletion of account (DELETE TRIGGER)

Go

create trigger removeaccount

on Bank.Accounts

instead of delete

as

declare @action varchar(20)

declare @Balance bigint

declare @Type varchar(10)

declare @ACC1 bigint

declare @message1 varchar(50)

declare @message2 varchar(50)

declare @log\_action varchar(50)

set @message1='Account Removed Successfully'

set @message2='Account Could Not Be Removed'

select @ACC1=a.Account\_No,@Type=a.Acc\_Type,@Balance=a.Balance from deleted a

if(@Type='Deposit')

begin

insert into Bank.AccountDeletion(Account\_No,log\_action,Balance,message,log\_timestamp)values(@ACC1,'Return Balance Money',@Balance,@message1,getdate());

update Accounts set Balance=0 where Account\_No=@ACC1

end

if(@Type='Loan')

insert into Bank.AccountDeletion(Account\_No,log\_action,Balance,message,log\_timestamp)values(@ACC1,'Loan Not Repaid',@Balance,@message2,getdate());

print 'Before Delete Trigger Fired'

A screenshot of a cell phone

Description automatically generated

4. Trigger to update interest (UPDATE TRIGGER)

Go

create trigger updateinterest

on Bank.Accounts

after update

as

declare @Type varchar(10)

declare @ACC bigint

declare @Facility varchar(20)

declare @Interest int

set @Interest=0

select @ACC=a.Account\_No,@Type=a.Acc\_Type,@Facility=a.Facility from inserted a

if(@Type='Deposit')

begin

if(@Facility='Current')

update Bank.Accounts set Accounts.Interest\_Rate='0' where Account\_No=@ACC

else

update Bank.Accounts set Accounts.Interest\_Rate='4.5' where Account\_No=@ACC

end

if(@Type='Loan')

begin

if(@Facility='Home Loan')

update Bank.Accounts set Accounts.Interest\_Rate='8.5' where Account\_No=@ACC

if(@Facility='Bike Loan')

update Bank.Accounts set Accounts.Interest\_Rate='7.5' where Account\_No=@ACC

else

update Bank.Accounts set Accounts.Interest\_Rate='9.5' where Account\_No=@ACC

end

print 'After Update Trigger Fired'

A screenshot of a cell phone

Description automatically generated

**SQL Queries:**

**Correlated-Nested Queries:**

1. Return Customer details given the Transaction ID

Select First\_Name,Last\_name,Contact\_No

From Bank.Customer

Where CIF =(

select CIF

from Bank.Accounts a

where Account\_No =(

select Account\_No

from Bank.Transactions where Transaction\_ID='MNP3726482'));

2. Return Transactions in which the amount transacted is greater than the average amount transacted

select t.Transaction\_ID,t.Checker\_Id

From Bank.Transactions t

where t.Amount > (

select avg(Amount)

from Bank.Transactions)

**A screenshot of a social media post

Description automatically generated**

**Aggregate Queries:**

1. Return the Account Numbers and CIF having transactions >1

select t.Account\_No,count(t.Account\_No) as count, c.CIF

from Bank.Transactions t,Bank.Accounts a, Bank.Customer c

where t.Account\_No=a.Account\_No and a.CIF=c.CIF

group by c.CIF,t.Account\_No

having count(t.Account\_No)>1

2. Return the number of employess who's birth date is in a particular year

select Year(Birth\_Date) as Year\_No, count(First\_Name) as Cust\_Count

from Bank.Customer c,Bank.Accounts a

where c.CIF=a.CIF

group by Year(Birth\_Date)

3. Return the total amount of money transferred during transactions

select sum(Amount) total\_amount

from Bank.Transactions

A screenshot of a cell phone

Description automatically generated

**Outer-Join Queries:**

1. Find Transaction\_ID, Account\_No, CIF of customers who have '1' in their account number

select t.Transaction\_ID as Transaction\_ID,a.Account\_No as Account\_No,c.CIF as CIF

from (Bank.Customer c left outer join Bank.Accounts a on c.CIF=a.CIF) join Bank.Transactions as t

on a.Account\_No=t.Account\_No

group by a.Account\_No,c.CIF,t.Transaction\_ID

having a.Account\_No like '%1%'

2. Find Transaction\_ID, Account\_No and Maker ID of transactions who have done transactions of amount > 200000

select t.Transaction\_ID as Trans\_ID,a.Account\_No as Account\_No,e.PF\_No as PF\_No

from (Bank.Transactions t right outer join Bank.Employee e on t.Maker\_Id=e.PF\_No) join Bank.Accounts as a

on a.Account\_No=t.Account\_No

where Amount>200000

A screenshot of a social media post

Description automatically generated

# **Conclusion:**

This project was done using ‘MSSQL’ (MicroSoft Structured Query Language).

**Capabilities of this System:**

* Updates Balance upon ‘Success’ of transaction using triggers. We need not give another separate query to do this.
* Updates the Rate of Interest for a specific account if there is change in ‘Type’ or ‘Facility’ of the account.
* Checks whether the transaction can be made or not, and if it cannot be made, marks it as a ‘Failure’ in the system
* Does not delete the account but does change the balance to ‘0’ once everything is paid off and settled.

**Limitations:**

* We cannot completely delete a particular row of a table because of many foreign keys attached to it.
* Some fields should have been automatically filled like ‘Transaction\_ID’ where the user is not supposed to enter data, but that is not the case here.

**Future Enhancements:**

* We can add more tables for storing more information so that we can move from a basic system to a pretty complex one like in a real bank.
* We can include more triggers so that this system functions very efficiently and we need write much queries to get a small work done.