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:

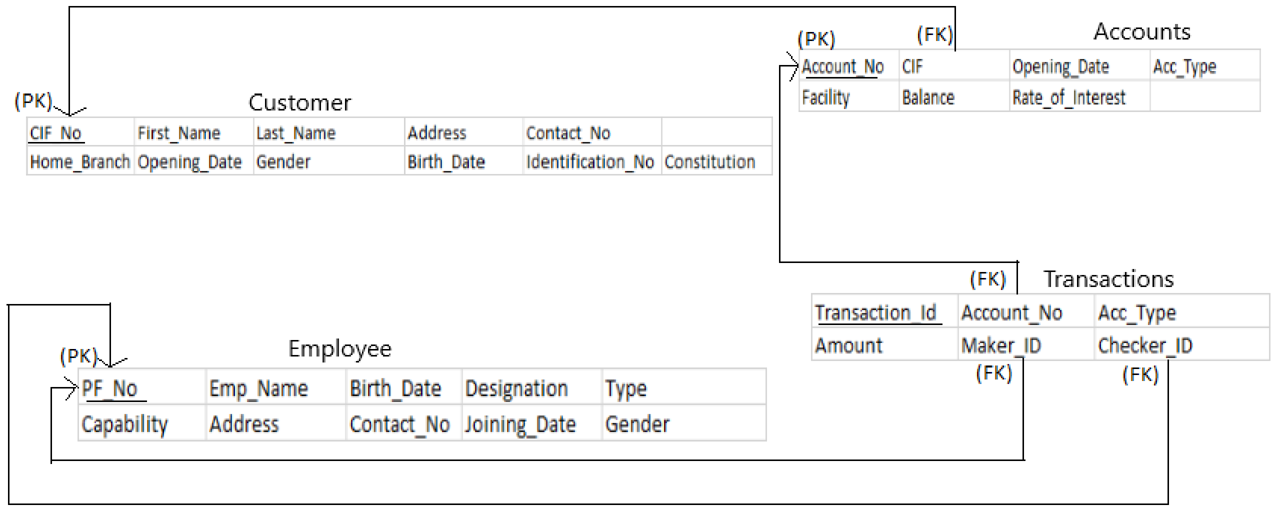
Introduction:

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.

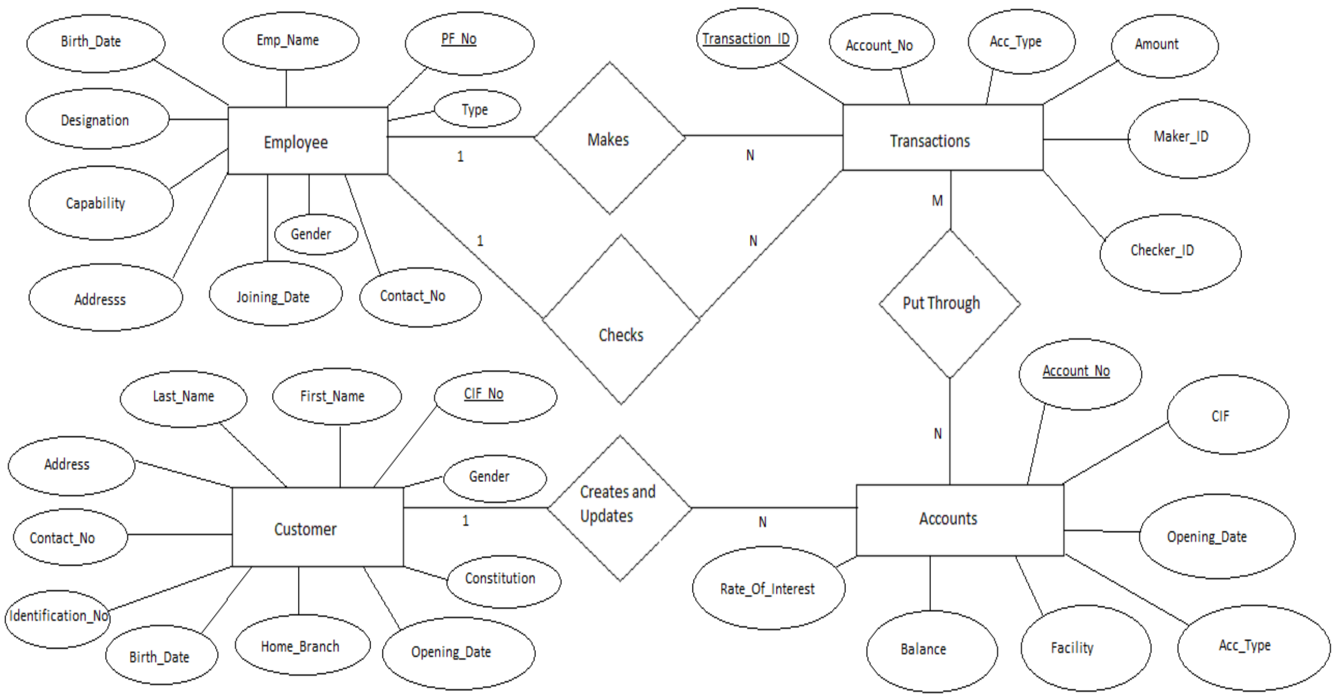
- The first entity is Customer: It contains attributes pertaining to general information about the customer, where CIF_No is the primary key. CIF stands for ‘Customer Identification File’ which is a special ID given to a customer for the first time as he comes to a bank.
- The second entity is Employee: 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:



ER Diagram:



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_I
D=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_timest
amp) 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_timest
```

amp) values ('Withdrawal

Success',@Amount,@Balance,@ACC1,@log_action,getdate());

else

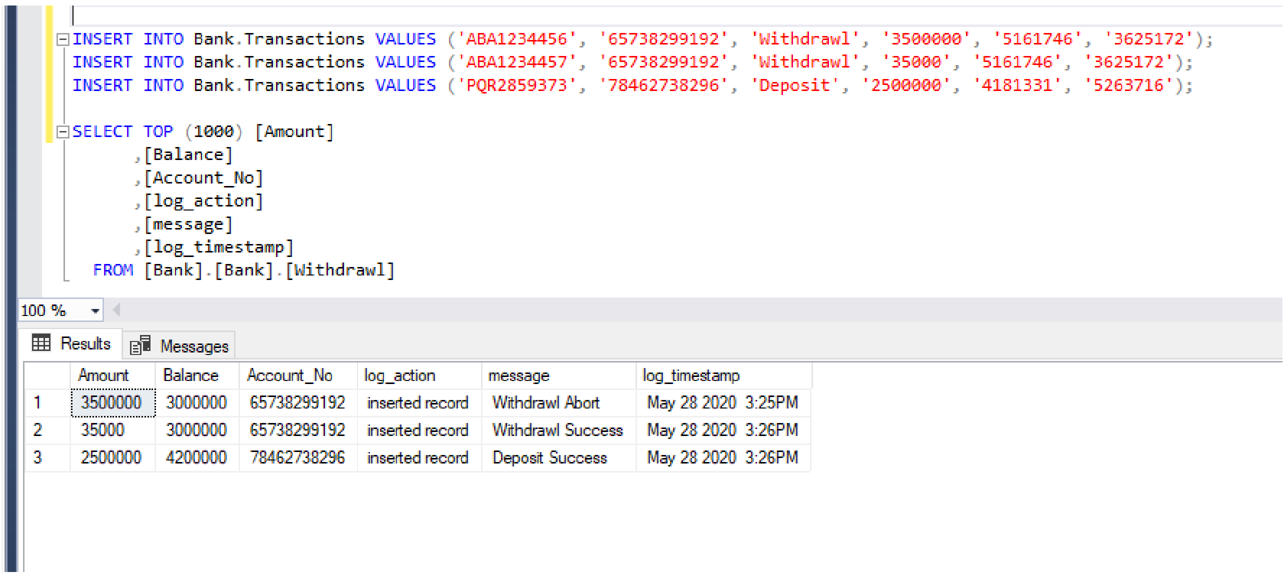
insert into

Bank.Withdrawl(message,Amount,Balance,Account_No,log_action,log_timesta

mp) values ('Deposit

Success',@Amount,@Balance,@ACC1,@log_action,getdate());

print 'After Insert Trigger Fired'



```
INSERT INTO Bank.Transactions VALUES ('ABA1234456', '65738299192', 'Withdrawl', '3500000', '5161746', '3625172');
INSERT INTO Bank.Transactions VALUES ('ABA1234457', '65738299192', 'Withdrawl', '35000', '5161746', '3625172');
INSERT INTO Bank.Transactions VALUES ('PQR2859373', '78462738296', 'Deposit', '2500000', '4181331', '5263716');

SELECT TOP (1000) [Amount]
      ,[Balance]
      ,[Account_No]
      ,[log_action]
      ,[message]
      ,[log_timestamp]
  FROM [Bank].[Bank].[Withdrawl]
```

100 %

Results | Messages

	Amount	Balance	Account_No	log_action	message	log_timestamp
1	3500000	3000000	65738299192	inserted record	Withdrawl Abort	May 28 2020 3:25PM
2	35000	3000000	65738299192	inserted record	Withdrawl Success	May 28 2020 3:26PM
3	2500000	4200000	78462738296	inserted record	Deposit Success	May 28 2020 3:26PM

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')

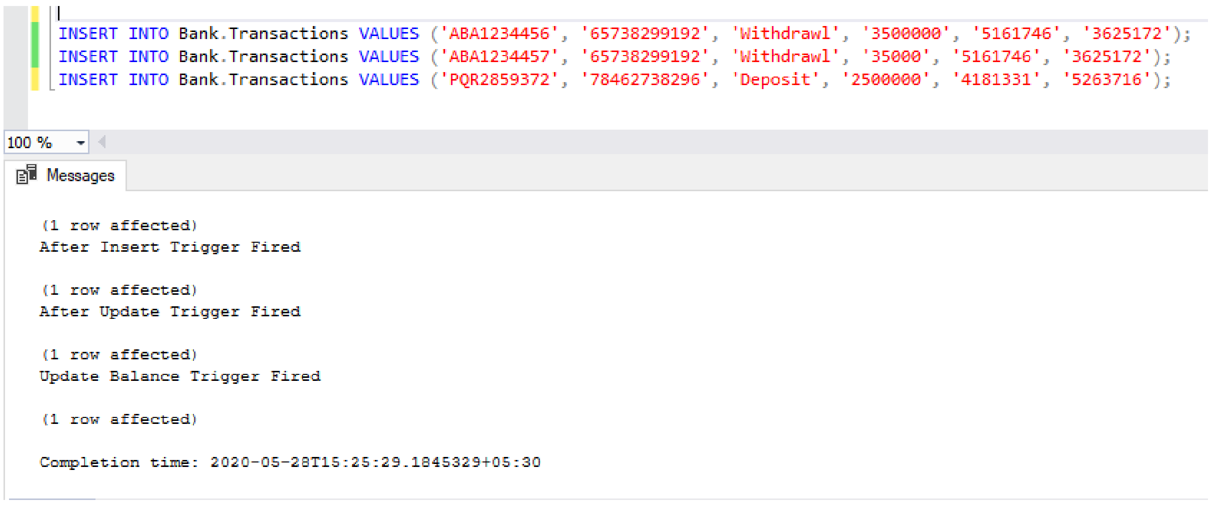
set @Balance=@Balance-@Amount

if(@Type='Deposit')

set @Balance=@Balance+@Amount

update Bank.Accounts set Balance=@Balance where Account_No=@ACC1

print 'Update Balance Trigger Fired'



```
INSERT INTO Bank.Transactions VALUES ('ABA1234456', '65738299192', 'Withdrawl', '3500000', '5161746', '3625172');
INSERT INTO Bank.Transactions VALUES ('ABA1234457', '65738299192', 'Withdrawl', '35000', '5161746', '3625172');
INSERT INTO Bank.Transactions VALUES ('PQR2859372', '78462738296', 'Deposit', '2500000', '4181331', '5263716');
```

100 %

Messages

```
(1 row affected)
After Insert Trigger Fired

(1 row affected)
After Update Trigger Fired

(1 row affected)
Update Balance Trigger Fired

(1 row affected)

Completion time: 2020-05-28T15:25:29.1845329+05:30
```

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());

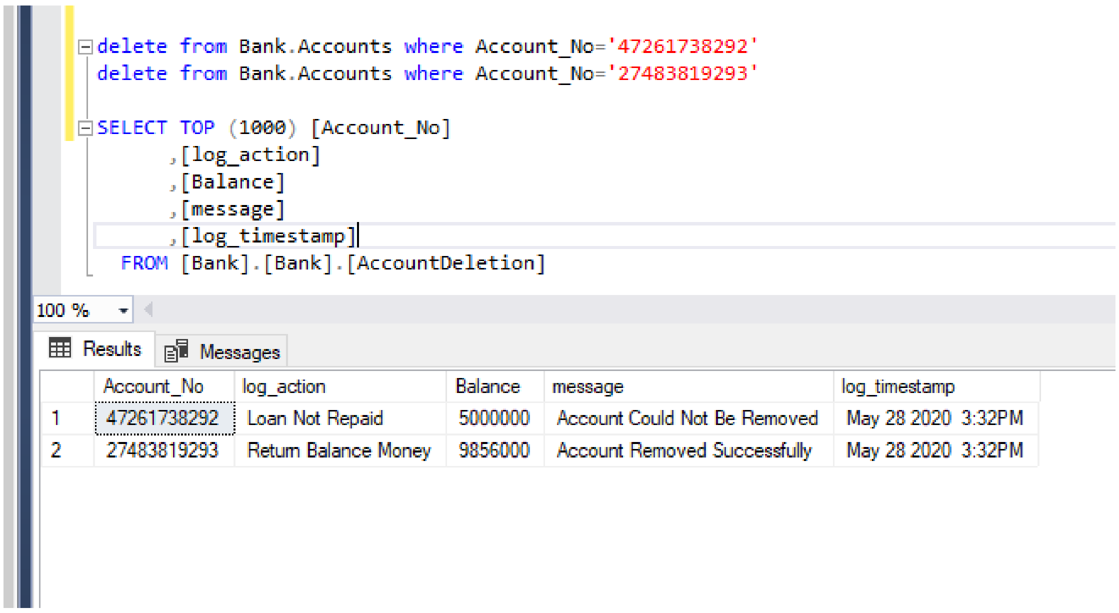
update Accounts set Balance=0 where Account_No=@ACC1

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'



```
delete from Bank.Accounts where Account_No='47261738292'
delete from Bank.Accounts where Account_No='27483819293'

SELECT TOP (1000) [Account_No]
      ,[log_action]
      ,[Balance]
      ,[message]
      ,[log_timestamp]
  FROM [Bank].[Bank].[AccountDeletion]
```

	Account_No	log_action	Balance	message	log_timestamp
1	47261738292	Loan Not Repaid	5000000	Account Could Not Be Removed	May 28 2020 3:32PM
2	27483819293	Return Balance Money	9856000	Account Removed Successfully	May 28 2020 3:32PM

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')
```

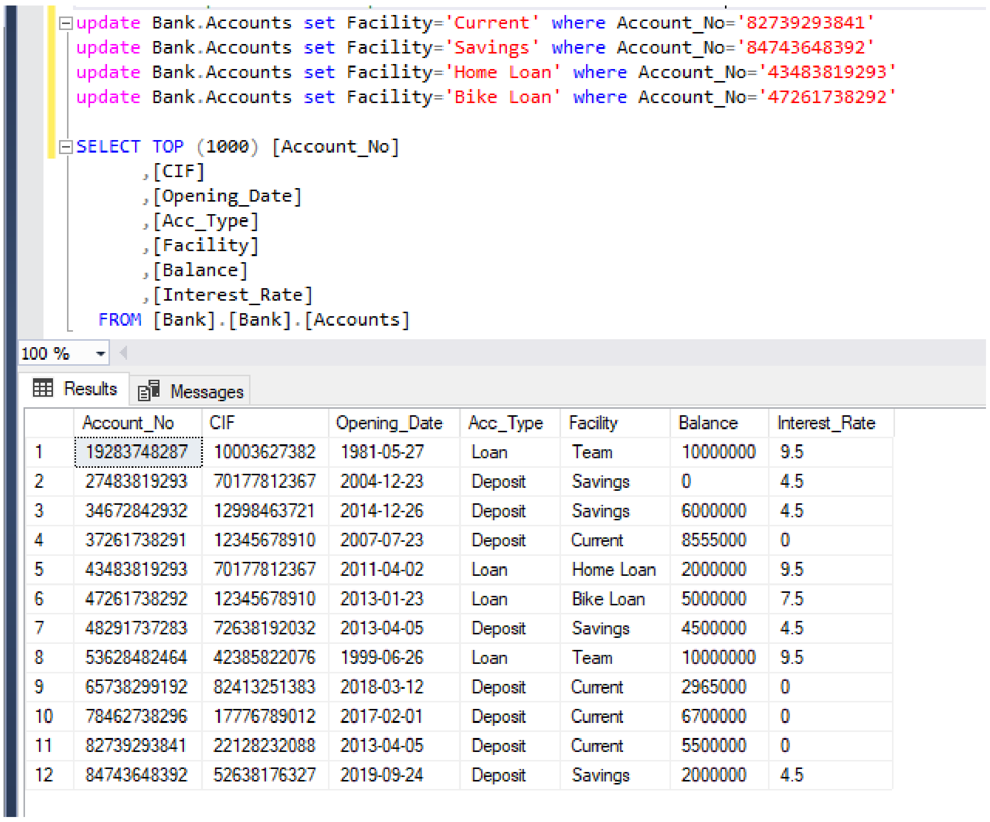
update Bank.Accounts set Accounts.Interest_Rate='7.5' where Account_No=@ACC

else

update Bank.Accounts set Accounts.Interest_Rate='9.5' where Account_No=@ACC

end

print 'After Update Trigger Fired'



```
update Bank.Accounts set Facility='Current' where Account_No='82739293841'
update Bank.Accounts set Facility='Savings' where Account_No='84743648392'
update Bank.Accounts set Facility='Home Loan' where Account_No='43483819293'
update Bank.Accounts set Facility='Bike Loan' where Account_No='47261738292'

SELECT TOP (1000) [Account_No]
      ,[CIF]
      ,[Opening_Date]
      ,[Acc_Type]
      ,[Facility]
      ,[Balance]
      ,[Interest_Rate]
  FROM [Bank].[Bank].[Accounts]
```

	Account_No	CIF	Opening_Date	Acc_Type	Facility	Balance	Interest_Rate
1	19283748287	10003627382	1981-05-27	Loan	Team	10000000	9.5
2	27483819293	70177812367	2004-12-23	Deposit	Savings	0	4.5
3	34672842932	12998463721	2014-12-26	Deposit	Savings	6000000	4.5
4	37261738291	12345678910	2007-07-23	Deposit	Current	8555000	0
5	43483819293	70177812367	2011-04-02	Loan	Home Loan	2000000	9.5
6	47261738292	12345678910	2013-01-23	Loan	Bike Loan	5000000	7.5
7	48291737283	72638192032	2013-04-05	Deposit	Savings	4500000	4.5
8	53628482464	42385822076	1999-06-26	Loan	Team	10000000	9.5
9	65738299192	82413251383	2018-03-12	Deposit	Current	2965000	0
10	78462738296	17776789012	2017-02-01	Deposit	Current	6700000	0
11	82739293841	22128232088	2013-04-05	Deposit	Current	5500000	0
12	84743648392	52638176327	2019-09-24	Deposit	Savings	2000000	4.5

SQL Queries:

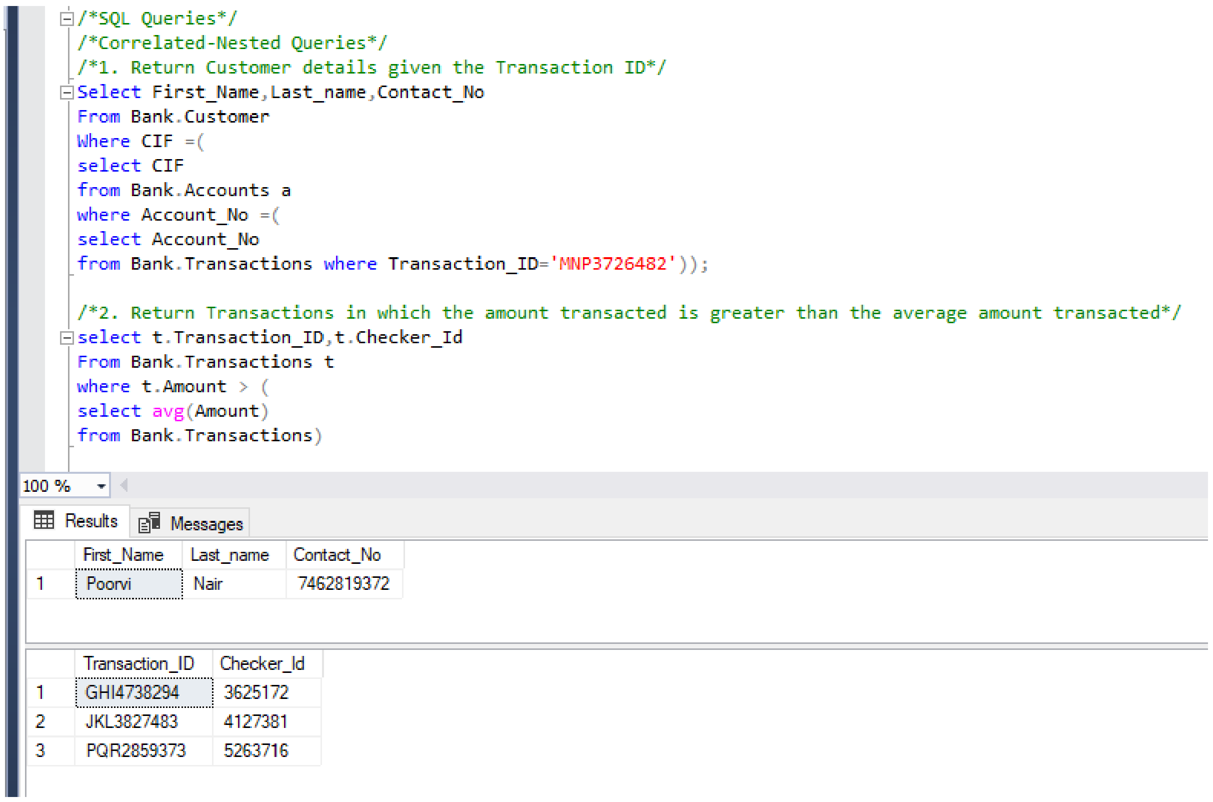
Correlated-Nested Queries:

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)
```



```
/*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)
```

	First_Name	Last_name	Contact_No
1	Poorvi	Nair	7462819372

	Transaction_ID	Checker_Id
1	GHI4738294	3625172
2	JKL3827483	4127381
3	PQR2859373	5263716

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

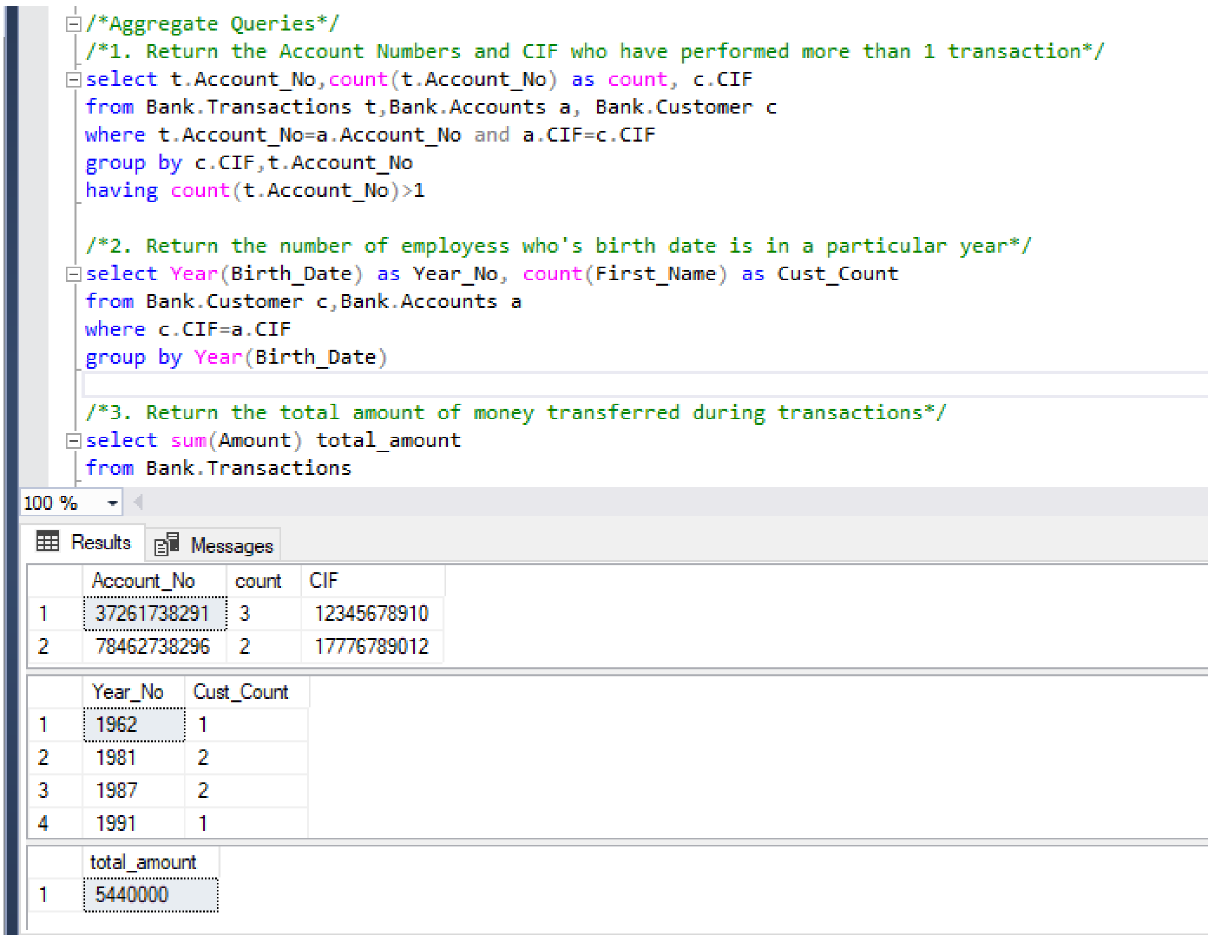
where c.CIF=a.CIF

group by Year(Birth_Date)

3. Return the total amount of money transferred during transactions

select sum(Amount) total_amount

from Bank.Transactions



```
/*Aggregate Queries*/
/*1. Return the Account Numbers and CIF who have performed more than 1 transaction*/
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
```

	Account_No	count	CIF
1	37261738291	3	12345678910
2	78462738296	2	17776789012

	Year_No	Cust_Count
1	1962	1
2	1981	2
3	1987	2
4	1991	1

	total_amount
1	5440000

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%'

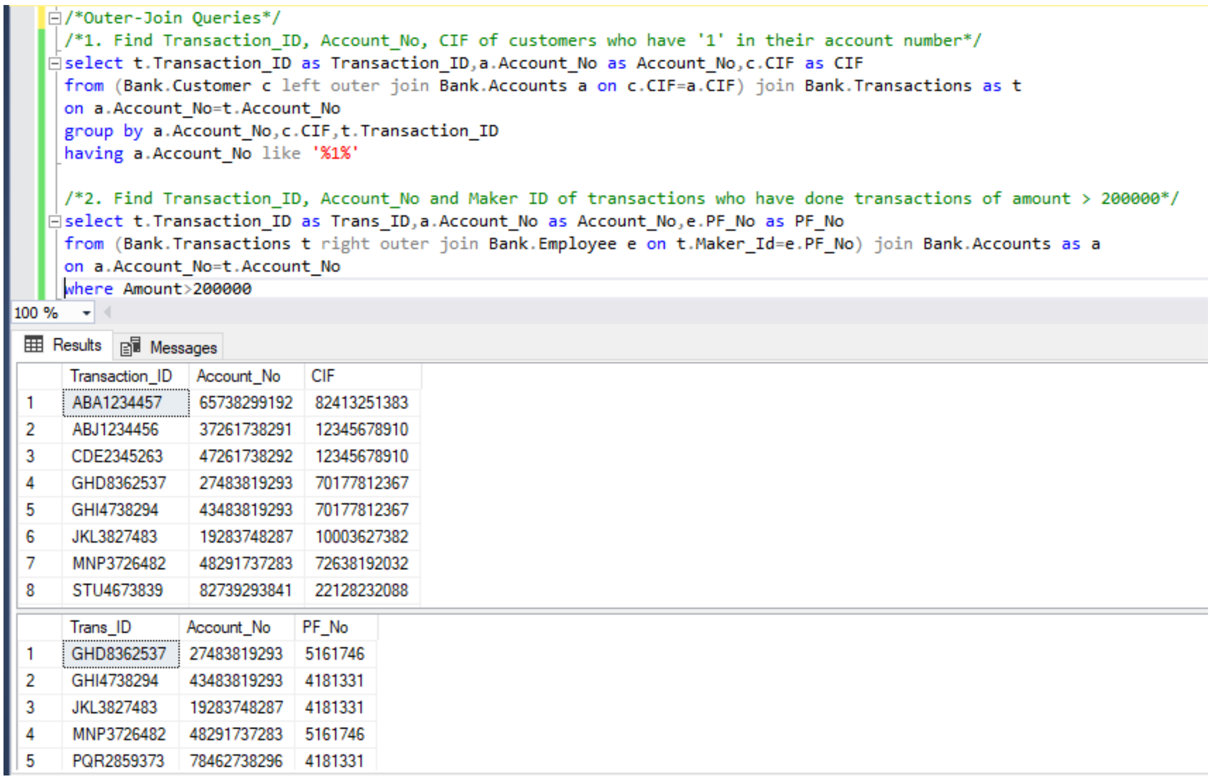
2. Find Transaction_ID, Account_No and Maker ID of transactions who have done transactions of amount > 200000

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



```
/*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
```

100 %

Results Messages

	Transaction_ID	Account_No	CIF
1	ABA1234457	65738299192	82413251383
2	ABJ1234456	37261738291	12345678910
3	CDE2345263	47261738292	12345678910
4	GHD8362537	27483819293	70177812367
5	GHI4738294	43483819293	70177812367
6	JKL3827483	19283748287	10003627382
7	MNP3726482	48291737283	72638192032
8	STU4673839	82739293841	22128232088

	Trans_ID	Account_No	PF_No
1	GHD8362537	27483819293	5161746
2	GHI4738294	43483819293	4181331
3	JKL3827483	19283748287	4181331
4	MNP3726482	48291737283	5161746
5	PQR2859373	78462738296	4181331

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.