

# **TERM PROJECT**

CS631-001 - Data Management System Design

Topic:  
**BANK PROJECT**

**LAKSHYA SAHARAN**  
[ls565@njit.edu](mailto:ls565@njit.edu)

## **GROUP-21**

*Lakshya Saharan ([ls565@njit.edu](mailto:ls565@njit.edu)),*

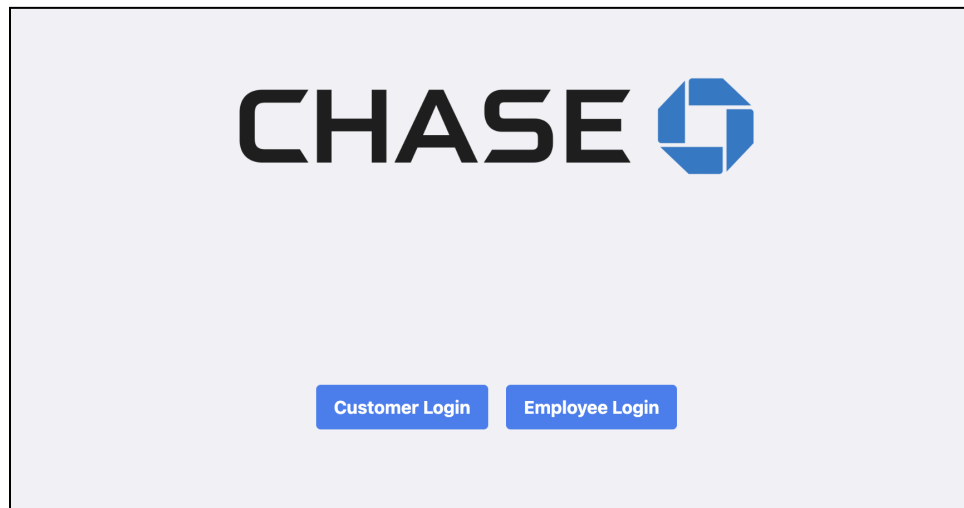
*Arjun Reddy ([ar2752@njit.edu](mailto:ar2752@njit.edu)),*

*Naga Sathwik Sangaraju ([ns2284@njit.edu](mailto:ns2284@njit.edu))*

# WELCOME TO OUR CHASE BANK

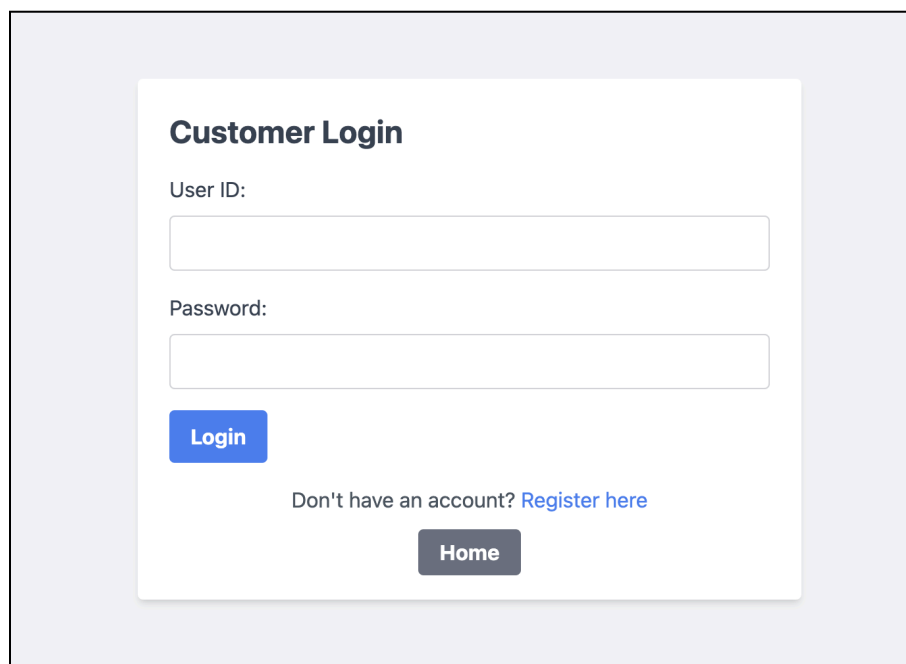
## Login Page:

- First will see our login page, where customer or employee either can login.



## Customer Login or Register:

- When the customer clicks on the "Customer Login" button, they will be redirected to the input of their UserId and Password.

A mockup of the customer login form. The form is a white rectangular box with a light gray border. At the top of the box, the title "Customer Login" is displayed in a bold, dark gray font. Below the title, there are two input fields: the first is labeled "User ID:" and the second is labeled "Password:". Each label is in a small, dark gray font, and the input fields are white rectangles with thin gray borders. Below the password field, there is a blue rectangular button with the word "Login" in white text. At the bottom of the form, there is a line of text that reads "Don't have an account? [Register here](#)", where "Register here" is a blue hyperlink. Below this text is a dark gray rectangular button with the word "Home" in white text. The entire form is centered within a larger light gray rectangular area.

- If the customer doesn't have an account, no problem, he/she can register.

**Register**

SSN:  
163456789

Name:  
Lakshya Saharan

Password:  
.....

Apt No:  
556

Street:  
155 Summit St

City:  
Newark

State:  
NJ

Zip:  
07103

**Register**

**Back**

- When the customer clicks register, then the data will be sent to backend in form of json. And the backend will hit two queries:
  - First one is to find the a personal banker for our customer. It is going to query the employees who are under the branch manager and assistant manager, then select the employee who has the least number of customers under him.

```
String query = "SELECT e.SSN AS EmployeeWithLeastCustomers, \n" +
    "    COUNT(c.SSN) AS CustomerCount\n" +
    "FROM EMPLOYEE e\n" +
    "LEFT JOIN CUSTOMER c ON e.SSN = c.Personal_banker\n" +
    "WHERE e.Super_SSN IN (\n" +
    "    SELECT SSN \n" +
    "    FROM EMPLOYEE \n" +
    "    WHERE Super_SSN = (\n" +
    "        SELECT Mgr_SSN \n" +
    "        FROM BRANCH \n" +
    "        WHERE BranchId = 1\n" +
    "    )\n" +
    ")\n" +
    "GROUP BY e.SSN, e.Name\n" +
    "ORDER BY CustomerCount ASC\n" +
    "LIMIT 1;";
```

- Second query will be to insert the customer and assign this personal banker.

```
query = "INSERT INTO CUSTOMER (SSN, Name, Password, Apt_no, Street, City, State, Zip, Personal_banker, Loan_no)\n" +
    "VALUES\n" +
    "(?, ?, ?, ?, ?, ?, ?, ?, ?, NULL)";
```

- Once the registration is done, the customer can now login using the credentials. Following query is used to verify the credentials from the backend.

```
String query = "SELECT COUNT(1) AS Is_Exist FROM CUSTOMER WHERE SSN=\""+ ssn +"\" AND Password=\""+password+"\"";
```

- In this we are verifying the value of count is 1 or 0.
- Additionally all the passwords are Base64 encoded. As you can see below from our customer table

SSN	Name	Password	Apt_no	Street	City	State	Zip	Personal_banker	Loan_no
111223333	John Doe	am9obnBhc3M=	101	Maple St	New York	NY	10001	234567890	NULL
163456789	Alex Ekeoch	YWxleHBhc3M=	123	155 Summit St	Newark	NJ	07103	345678901	1006
222334444	Jane Smith	amFuZXBhc3M=	202	Oak St	Jersey City	NJ	07305	987654321	NULL
333445555	Tom Brown	dG9tcGFzcw==	303	Pine St	Newark	NJ	07101	234567890	NULL
444556666	Emma Wilson	ZW1tYXBhc3M=	404	Birch St	Hoboken	NJ	07030	987654321	1007
555667777	Sophia Lee	c29waGlhcGFzcw==	505	Cedar St	Manhattan	NY	10002	987654321	NULL

## Customer Index Page:

- Now the customer lands on the index page which have multiple feature which he/she can use. Lets go through them one by one.

Make Transaction

Check Balance

Check Summary

Personal Banker

Log out

## Open a new account:

- So since our customer just registered, he doesn't have a bank account. In this case he visits our local branch of Chase Bank and the employee or personal banker will verify the details and open an account for the customer.
- So now the employee logs into the website using following form:

Employee Login

EmpID:

345678901

Password:

.....

Login

Home

- Following query will run in the backend to verify the employee. Its working in a similar way as we checked for customers.

```
String query = "SELECT COUNT(1) AS Is_Exist FROM EMPLOYEE " +
    "WHERE SSN=\""+employeeId+"\" AND Password=\""+password+"\"";
```

- Once an employee is logged in, he/she will see the following options.

Open Account

Deposit Money

Give Loan

Check My Employees

Log out

- Since our customer is waiting for the account to be opened, so the employee will go to open account option and ask the customer for saving or checking or both type of account.

### Open Account

SSN:

Account Type:

☒ Checking ☐ Saving

Submit

- Assuming our customer only wanted a checking account for regular usage. Following query will run in the backend:

```
String query = "INSERT INTO ACCOUNT (Acc_no, Acc_type, Balance, Recent_Access_Date, Customer_SSN)\n" +  
"VALUES (?, ?, ?, ?, ?)";
```

- We have added a bifurcation between the type of accounts based on the starting two digits of account number:
  - CHECKING account number → Starts with **10**
  - SAVING account number → Starts with **20**
- In our case the customer asked for a checking account so we gave this account number: **1010418690**. As you can see the first two digits are 10 and rest of the 8 digits are unique for every account.

Acc_no	Acc_type	Balance	Recent_Access_Da...	Customer_SSN
1010418690	CHECKING	0.00	2024-12-09	163456789

## Deposit money:

- Now the customer has a checking account, let's add some money to it. So the customer will provide the banker to deposit some money to his account number. The banker after verifying the details will create a deposit transaction and the balance will be updated.

The form is titled "Deposit Money" and is contained within a light gray border. It has three input fields: "Account Number:" with the value "1010418690", "SSN:" with the value "163456789", and "Amount:" with the value "25000". Below the input fields are two buttons: a blue "Submit" button and a gray "Back" button.

- So for this deposit transaction the following queries will run.
- First will fetch the current balance from the account.

```
// get current balance from account table
String balanceFromAccountQuery = "SELECT Balance FROM ACCOUNT WHERE Acc_no=?";
```

- Next will add the new amount to the current balance and create a transaction as "Customer Deposit".

```
String query = "INSERT INTO TRANSACTION (Tranc_code, Tranc_type, Charge, Acc_no, Amount, Date, Hour, Balance_Snap)\n" +
"VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
```

- And finally will put the updated balance into the account table.

```
// update the balance in account table
String accountUpdateQuery = "UPDATE ACCOUNT SET Balance=?, Recent_Access_Date=? WHERE Acc_no=?";
```

## Withdrawing money:

- Now that our customer has some money in his/her checking account. Lets with draw some amount using “Make Transaction” feature in customer page.

### Make Transaction

Account Number:

SSN:

Amount:

- Once clicked on submit, it will fire the withdraw api in the backend. Which in a similar way as deposit money, will call three queries. Only difference this time is of the transaction charge based on the type of account. So for checking will deduct \$1.00 and for saving \$10.00.

## Checking Balance:

- Our customer can check balance using our feature.
- So far he/she first deposited \$25000 and then withdrawn \$500 + \$1(charge for checking account). The balance should be \$24499.

Enter Account Number:

Account Type	Balance
Checking	24499

- Following is the query used:

```
String balanceFromAccountQuery = "SELECT BaLance FROM ACCOUNT WHERE Acc_no=?";
```



## Checking Bank Statement:

- Our customer has made some transactions so he/she can see the statement using our feature.

Enter Account Number:

Type	Charge	Account Number	Date	Hour	Balance
WD	\$1	1010418690	2024-12-09	19:14:50	24499
CD	\$0	1010418690	2024-12-09	19:01:25	25000

- As you can see, the customer has to input the account number and they will see all the transactions with their respective balance snapshot.
- We are using the following query which gets the transaction, then sort it in descending order of date and then hour. So the latest transaction is on the top.

```
String query = "SELECT * FROM TRANSACTION WHERE Acc_no=? ORDER BY Date DESC, Hour DESC";
```

## Personal Banker Info:

- Suppose the customer wants to take a loan or have a general query then he/she can contact their personal banker using our feature.
- When the customer clicks on the “Personal Banker” button they will see the name & phone no. of the banker along with their supervisor details.

### Personal Banker

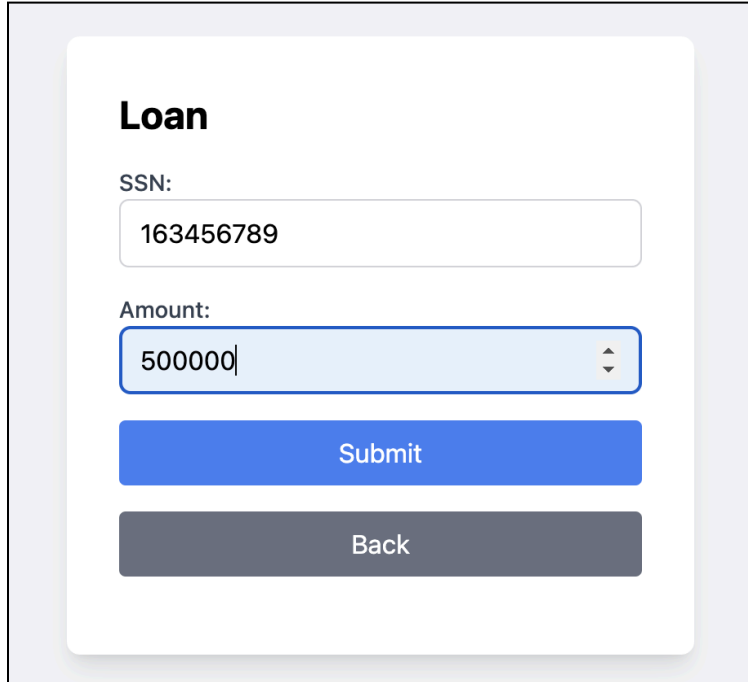
Detail	Information
Name	Eren Canan
Telephone	+799-214-9907
Manager Name	Donald Trump
Manager Telephone	+862-214-9922

- The query for this involves the customer table and employee table. First will get the SSN of personal banker from that customer. Then will use that to get the employee details along with the supervisor details.

```
String query = "SELECT \n" +  
    "    e.Name AS EmployeeName,\n" +  
    "    e.Telephone AS EmployeeTelephone,\n" +  
    "    m.Name AS ManagerName,\n" +  
    "    m.Telephone AS ManagerTelephone\n" +  
    "FROM \n" +  
    "    EMPLOYEE e\n" +  
    "LEFT JOIN \n" +  
    "    EMPLOYEE m ON e.Super_SSN = m.SSN\n" +  
    "WHERE \n" +  
    "    e.SSN = (SELECT Personal_banker FROM CUSTOMER WHERE SSN = ?)";
```

## Give Loan to Customer:

- Suppose our customer wants build a house or buy a car then they will probably need a loan.
- The customer has to go to our branch and provide verification to the banker. Then the banker will process the loan.



**Loan**

SSN:

Amount:

Submit

Back

- Here our customer is requesting a loan of \$50,000. That's a lot, he better be buying a nice car or suv maybe.

- In the backend will run following queries for properly storing the loan.
- First one will add this new loan into the LOAN table by incrementing the unique loan number.

```
ps = con.prepareStatement( sql: "SELECT MAX(Loan_Number) AS Loan_Number FROM LOAN")
```

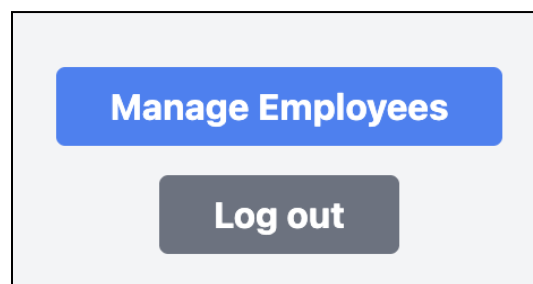
```
String loanQuery = "INSERT INTO LOAN (Loan_Number, Loan_Amount, Monthly_repay, BranchID) " +  
"VALUES (?, ?, 150, 1)";
```

- After that will add this new loan number in the customer table as well so that the it is attached with customer as a foreign key referencing to the loan table.

```
// update loan number in customer table  
String query = "UPDATE CUSTOMER SET Loan_no=? WHERE SSN=?";
```

## Managing Bank Employees:

- Let's jump into the main role of our branch manager. He/She can do all the things which a normal employee can do but additionally also edit or delete the employees.
- Whenever the manager logs in using the same employee login page, the backend will identify it whether its a manager or normal employee. If its a manager then will unlock manage employee feature exclusive to the branch manager.



- The manager will see this above feature when they login. For simplicity we are just showing this button on the manager's page although he/she has all the capabilities as normal employee.
- To identify as a manager, we are running following query in the backend:

```
String checkManagerQuery = "SELECT Mgr_SSN from BRANCH where BranchID=1";
ps = con.prepareStatement(checkManagerQuery);
rs = ps.executeQuery();
while(rs.next()) {
    String MgrSSN = rs.getString( columnLabel: "Mgr_SSN");
    if(MgrSSN.equals(employeeId)) {
        result.append("Success_isManager");
    }
}
}
```

- When the manager clicks on the “Manage Employee” button, then our backend will fetch details of all the employees.

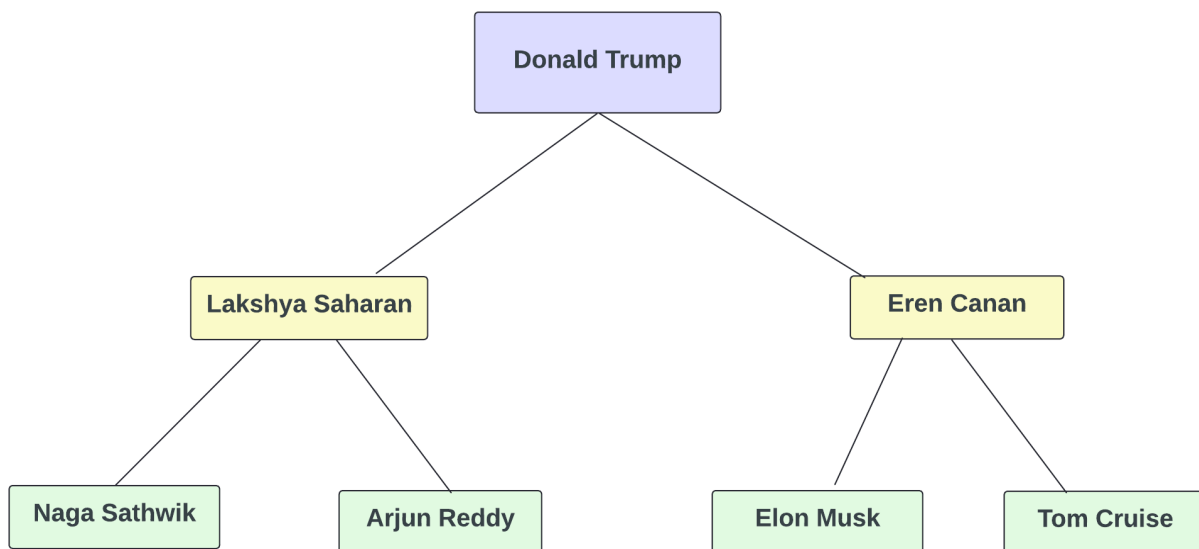
Manage Employees						
SSN	Name	Phone	Dependent Name	Start Date	Lenght of Employment	Actions
123456789	Lakshya Saharan	+862-214-9904	Jitin Saharan	2015-03-15	9	<a href="#">Edit</a> <a href="#">Delete</a>
234567890	Naga Sathwik	+432-214-9906	Prudhvi	2018-02-10	6	<a href="#">Edit</a> <a href="#">Delete</a>
345678901	Eren Canan	+799-214-9907	Scarlett Johansson	2019-05-25	5	<a href="#">Edit</a> <a href="#">Delete</a>
456789012	Donald Trump	+862-214-9922	Ivanka Trump	2020-08-30	4	
456789013	Tom Cruise	+851-214-9908	Conner Cruise	2020-08-30	4	<a href="#">Edit</a> <a href="#">Delete</a>
456789014	Elon Musk	+862-214-9909	Justine Musk	2020-08-30	4	<a href="#">Edit</a> <a href="#">Delete</a>
987654321	Arjun Reddy	+915-214-9925	Ramesh Reddy	2017-06-20	7	<a href="#">Edit</a> <a href="#">Delete</a>

[Back](#)

- As you can see above the manager can edit or delete his employees **except himself**.

## Cascade effect on deleting a assistant manager:

- Our bank has three layers of hierarchy among the employees. Top one is branch manager, then assistant manager and after that normal employees.
- **Branch Manager > Assistant Manager > Employee**
- Now suppose if the assistant manager is deleted then what happens to his/her employees?
- In that case the management is transferred to a default manager.
- Consider the following example:
  - **Donald Trump** → **Branch Manager**
  - **Lakshya Saharan** → **Assistant Manager**
  - **Eren Canan** → **Assistant Manager**
  - **Naga Sathwik** → **Employee**
  - **Arjun Reddy** → **Employee**
  - **Tom Cruise** → **Employee**
  - **Elon Musk** → **Employee**



- Here if Lakshya Saharan is deleted then his employees will be transferred to Eren Canan.
- To achieve this we first tried **ON DELETE SET DEFAULT** , but that doesn't work in MYSQL for certain reasons.
- Then we tried **TRIGGER** , but that also doesn't work because the table EMPLOYEE has circular reference and thus it can have infinite recursion.

- So the solution is to use **PROCEDURE**. So we created it in the database and called it from the backend.

```
DELIMITER $$
```

```
CREATE PROCEDURE delete_employee(IN employee_ssn INT)
```

```
BEGIN
```

```
-- Delete the specified manager
```

```
DELETE FROM EMPLOYEE WHERE SSN = employee_ssn;
```

```
-- Update rows where super_id is NULL to default to 1
```

```
UPDATE EMPLOYEE
```

```
SET Super_SSN = '345678901'
```

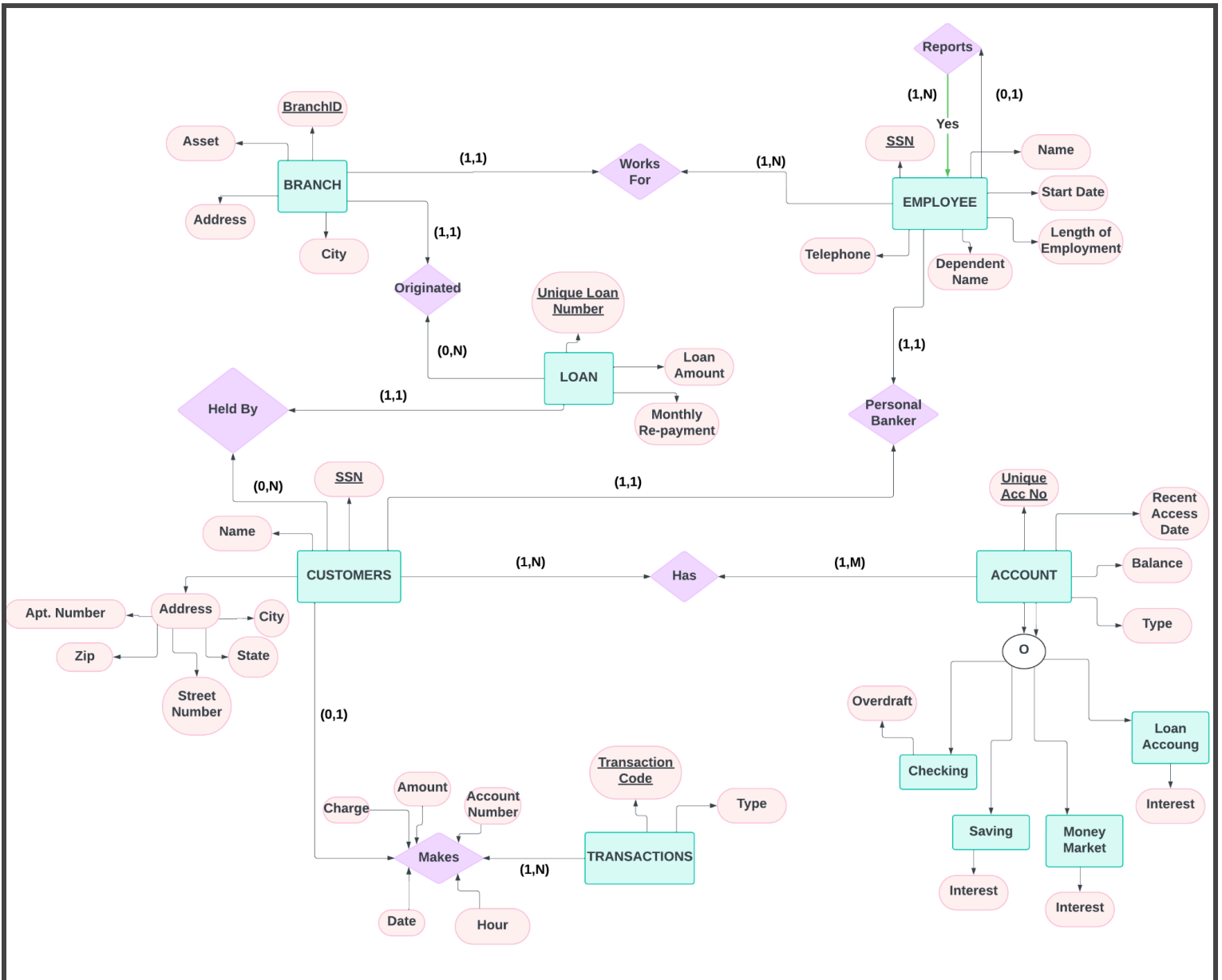
```
WHERE Super_SSN IS NULL;
```

```
END$$
```

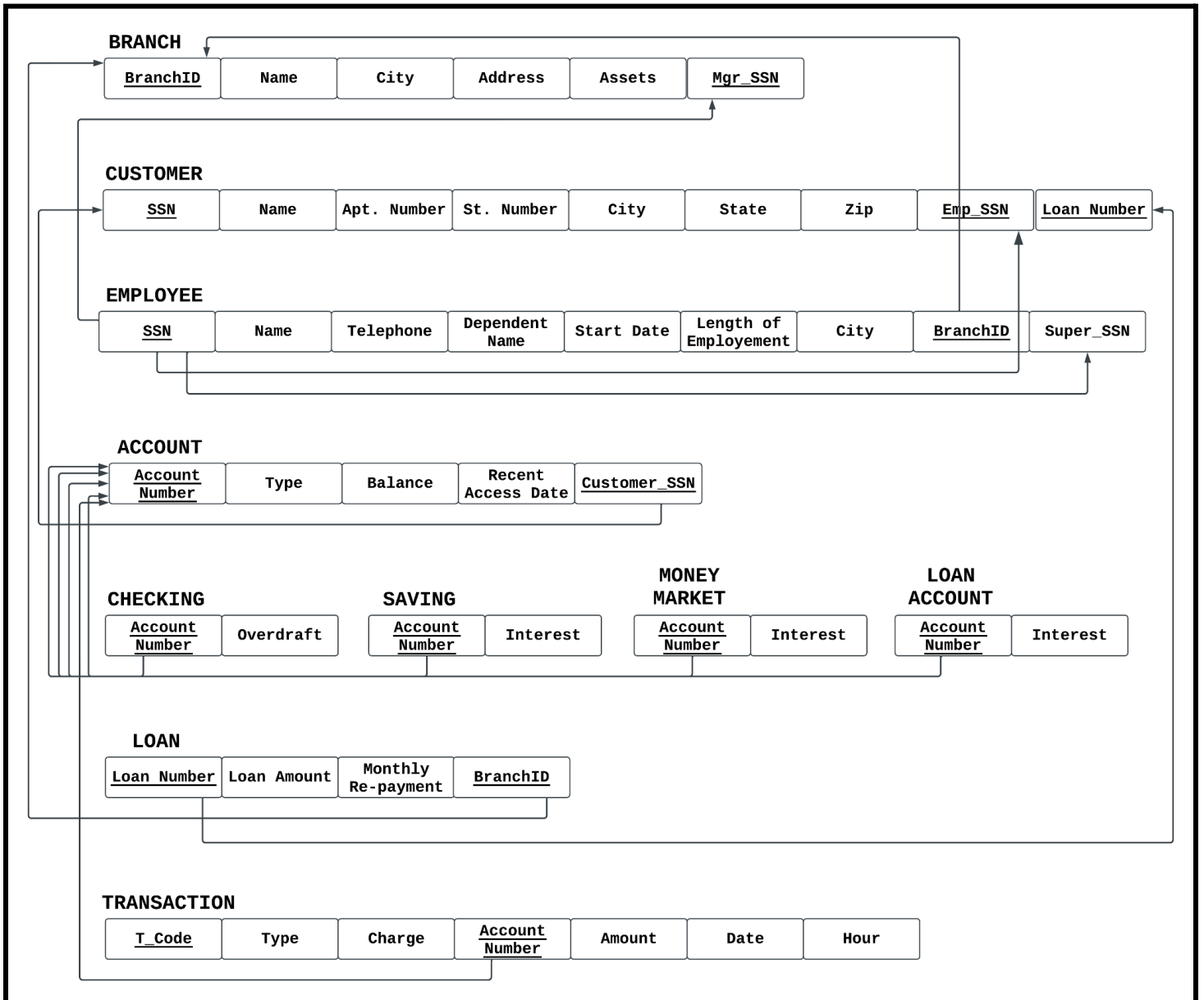
```
DELIMITER ;
```

- This will first delete the employee based on the SSN.
- The delete query will put Super\_SSN of the affected employees as NULL
- Then the update query will assign a default manager.

## ER Diagram



# Relational Schema





# DATABASE DESIGN AND QUERIES

## ★ Creating database BANK

```
CREATE DATABASE BANK;  
USE BANK;
```

Action Output				
		Time	Action	Response
✓	1	01:41:21	CREATE DATABASE BANK	1 row(s) affected
✓	2	01:41:21	USE BANK	0 row(s) affected

## ★ Creating table BRANCH

```
CREATE TABLE BRANCH(  
    BranchID      INTEGER          NOT NULL,  
    Name          VARCHAR(20)      NOT NULL,  
    City          VARCHAR(20),  
    Address       VARCHAR(50),  
    Asset         VARCHAR(20),  
    Mgr_SSN       CHAR(9),  
    PRIMARY KEY (BranchID)  
);
```

## ★ Creating table EMPLOYEE

```
CREATE TABLE EMPLOYEE(  
    SSN           CHAR(9)          NOT NULL,  
    Name          VARCHAR(20)      NOT NULL,  
    Telephone     VARCHAR(10),  
    Dependent_name VARCHAR(20),  
    Start_date    DATE,  
    Length_of_emp INTEGER,  
    Super_SSN     CHAR(9)          DEFAULT '456789012',  
    BranchID      INTEGER,  
    PRIMARY KEY (SSN),  
    FOREIGN KEY (Super_SSN) REFERENCES EMPLOYEE(SSN)  
        ON DELETE CASCADE      ON UPDATE CASCADE,  
    FOREIGN KEY (BranchID) REFERENCES BRANCH(BranchID)  
        ON DELETE SET NULL     ON UPDATE CASCADE
```

```
);
```

★ Now altering Branch table to add foreign key constraint referencing to Employee table

```
ALTER TABLE BRANCH
ADD CONSTRAINT fk_mgr_ssn
FOREIGN KEY (Mgr_SSN) REFERENCES EMPLOYEE(SSN)
ON DELETE SET NULL ON UPDATE CASCADE;
```

★ Inserting 5 rows in both these tables

```
-- Inserting branches with NULL managers
INSERT INTO BRANCH (BranchID, Name, City, Address, Asset, Mgr_SSN)
VALUES
(1, 'Chase Bank', 'New York', '123 Broadway St', 'Financial', NULL),
(2, 'TD Bank', 'Jersey City', '456 Market St', 'Financial', NULL),
(3, 'Wells Fargo', 'Newark', '789 Main St', 'Financial', NULL),
(4, 'Bank of America', 'Hoboken', '101 River Rd', 'Financial', NULL),
(5, 'Citibank', 'Manhattan', '202 Park Ave', 'Financial', NULL);

-----
-- Inserting employees with valid BranchID references
INSERT INTO EMPLOYEE (SSN, Name, Telephone, Dependent_name, Start_date,
Length_of_emp, Super_SSN, BranchID)
VALUES
('123456789', 'Lakshya Saharan', '2015551234', 'Jitin Saharan',
'2015-03-15', 9, NULL, 1), -- Employee for Chase Bank
('987654321', 'Arjun Reddy', '2015555678', 'Ramesh Reddy', '2017-06-20', 7,
NULL, 2), -- Employee for TDBank
('234567890', 'Naga Sathwik', '2015559876', 'Prudhvi', '2018-02-10', 6,
NULL, 3), -- Employee for Wells Fargo
('345678901', 'Eren Canan', '2015553456', 'Scarlett Johansson',
'2019-05-25', 5, NULL, 4), -- Employee for Bank of America
('456789012', 'Donald Trump', '2015556789', 'Ivanka Trump', '2020-08-30',
4, NULL, 5); -- Employee for Citibank
```

★ Because of the circular dependency of these two tables we have added NULL values to foreign keys, but now will add data to them.

```
-- Updating branches with managers' SSNs
```

```
UPDATE BRANCH
SET Mgr_SSN = '123456789' -- Lakshya Saharan is the manager of Chase Bank
WHERE BranchID = 1;
```

```
UPDATE BRANCH
SET Mgr_SSN = '987654321' -- Arjun Reddy is the manager of TD Bank
WHERE BranchID = 2;
```

```
UPDATE BRANCH
SET Mgr_SSN = '234567890' -- Naga Sathwik is the manager of Wells Fargo
WHERE BranchID = 3;
```

```
UPDATE BRANCH
SET Mgr_SSN = '345678901' -- Eren Canan is the manager of Bank of America
WHERE BranchID = 4;
```

```
UPDATE BRANCH
SET Mgr_SSN = '456789012' -- Donald Trump is the manager of Citibank
WHERE BranchID = 5;
```

-----

-- Eren Canan manages Lakshya Saharan, Naga Sathwik, and Arjun Reddy

```
UPDATE EMPLOYEE
SET Super_SSN = '345678901' -- Eren Canan's SSN
WHERE SSN = '123456789'; -- Lakshya Saharan's SSN
```

```
UPDATE EMPLOYEE
SET Super_SSN = '345678901' -- Eren Canan's SSN
WHERE SSN = '234567890'; -- Naga Sathwik's SSN
```






```
UPDATE EMPLOYEE
SET Super_SSN = '345678901' -- Eren Canan's SSN
WHERE SSN = '987654321'; -- Arjun Reddy's SSN
```

-- Donald Trump manages Eren Canan

```
UPDATE EMPLOYEE
SET Super_SSN = '456789012' -- Donald Trump's SSN
WHERE SSN = '345678901'; -- Eren Canan's SSN
```

## ★ BRANCH table display






```
SELECT * FROM BRANCH;
```

Result Grid						
Filter Rows: <input type="text" value="Search"/>						
Edit:   						
Export/Import:  						
BranchID	Name	City	Address	Asset	Mgr_SSN	
1	Chase Bank	New York	123 Broadway St	Financial	123456789	
2	TD Bank	Jersey City	456 Market St	Financial	987654321	
3	Wells Fargo	Newark	789 Main St	Financial	234567890	
4	Bank of America	Hoboken	101 River Rd	Financial	345678901	
5	Citibank	Manhattan	202 Park Ave	Financial	456789012	
NULL	NULL	NULL	NULL	NULL	NULL	

### ★ EMPLOYEE table display

- Eren is manager of Lakshya, Sathwik and Arjun
- Donald Trump is manager of Eren

```
SELECT * FROM EMPLOYEE;
```

Result Grid							
Filter Rows: <input type="text" value="Search"/>							
Edit:   							
Export/Import:  							
SSN	Name	Telephone	Dependent_name	Start_date	Length_of_emp	Super_SSN	BranchID
123456789	Lakshya Saharan	2015551234	Jitin Saharan	2015-03-15	9	345678901	1
234567890	Naga Sathwik	2015559876	Prudhvi	2018-02-10	6	345678901	3
345678901	Eren Canan	2015553456	Scarlett Johansson	2019-05-25	5	456789012	4
456789012	Donald Trump	2015556789	Ivanka Trump	2020-08-30	4	NULL	5
987654321	Arjun Reddy	2015555678	Ramesh Reddy	2017-06-20	7	345678901	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

### ★ Creating LOAN table

- Added cascade effect on update and on delete it will set to NULL

```
CREATE TABLE LOAN(
    Loan_Number          INTEGER          NOT NULL,
    Loan_Amount          INTEGER          NOT NULL,
    Monthly_repay        INTEGER          NOT NULL,
    BranchID             INTEGER,
    PRIMARY KEY(Loan_Number),
    FOREIGN KEY(BranchID) REFERENCES BRANCH(BranchID)
                        ON DELETE SET NULL      ON UPDATE CASCADE
);
```

### ★ Inserting 5 rows in Loan table

```
-- Inserting 5 rows into the LOAN table
INSERT INTO LOAN (Loan_Number, Loan_Amount, Monthly_repay, BranchID)
VALUES
(1001, 50000, 1500, 1), -- Loan for Chase Bank (BranchID = 1)
(1002, 30000, 900, 2), -- Loan for TD Bank (BranchID = 2)
(1003, 25000, 750, 3), -- Loan for Wells Fargo (BranchID = 3)
(1004, 100000, 2500, 4), -- Loan for Bank of America (BranchID = 4)
(1005, 15000, 450, 5); -- Loan for Citibank (BranchID = 5)
```






Result Grid				
Filter Rows: <input type="text" value="Search"/>				
	Loan_Number	Loan_Amount	Monthly_repay	BranchID
▶	1001	50000	1500	1
	1002	30000	900	2
	1003	25000	750	3
	1004	100000	2500	4
	1005	15000	450	5
	NULL	NULL	NULL	NULL

## ★ Creating CUSTOMER table

```
CREATE TABLE CUSTOMER(
    SSN                CHAR(9)                NOT NULL,
    Name               VARCHAR(20)            NOT NULL,
    Apt_no             INTEGER,
    Street             VARCHAR(20),
    City               VARCHAR(20),
    State              VARCHAR(20),
    Zip                CHAR(9),
    Personal_banker     CHAR(9),
    Loan_no             INTEGER,
    PRIMARY KEY(SSN),
    FOREIGN KEY(Personal_banker) REFERENCES EMPLOYEE(SSN)
        ON DELETE SET NULL                ON UPDATE CASCADE,
    FOREIGN KEY(Loan_no) REFERENCES LOAN(Loan_number)
        ON DELETE SET NULL                ON UPDATE CASCADE
);
```

## ★ Inserting 5 rows into CUSTOMER table

```
-- Inserting 5 rows into the CUSTOMER table
INSERT INTO CUSTOMER (SSN, Name, Apt_no, Street, City, State, Zip,
Personal_banker, Loan_no)
VALUES
('111223333', 'John Doe', 101, 'Maple St', 'New York', 'NY', '10001',
'123456789', 1001), -- Personal banker: Lakshya Saharan
('222334444', 'Jane Smith', 202, 'Oak St', 'Jersey City', 'NJ', '07305',
'987654321', 1002), -- Personal banker: Arjun Reddy
('333445555', 'Tom Brown', 303, 'Pine St', 'Newark', 'NJ', '07101',
'234567890', 1003), -- Personal banker: Naga Sathwik
('444556666', 'Emma Wilson', 404, 'Birch St', 'Hoboken', 'NJ', '07030',
'345678901', 1004), -- Personal banker: Eren Canan
('555667777', 'Sophia Lee', 505, 'Cedar St', 'Manhattan', 'NY', '10002',
'456789012', 1005); -- Personal banker: Donald Trump
```

Result Grid   Filter Rows: <input type="text"/> Search <input type="text"/> Edit:    Export/Import:									
	SSN	Name	Apt_no	Street	City	State	Zip	Personal_banker	Loan_no
▶	111223333	John Doe	101	Maple St	New York	NY	10001	123456789	1001
	222334444	Jane Smith	202	Oak St	Jersey City	NJ	07305	987654321	1002
	333445555	Tom Brown	303	Pine St	Newark	NJ	07101	234567890	1003
	444556666	Emma Wilson	404	Birch St	Hoboken	NJ	07030	345678901	1004
	555667777	Sophia Lee	505	Cedar St	Manhattan	NY	10002	456789012	1005
	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

## ★ Creating ACCOUNT table

```
CREATE TABLE ACCOUNT (
    Acc_no                INTEGER                NOT NULL,
    Acc_type              VARCHAR(20)            NOT NULL,
    Balance               DECIMAL(10, 2)        NOT NULL,
    Recent_Access_Date    DATE                  NOT NULL,
    Customer_SSN          CHAR(9),
    PRIMARY KEY (Acc_no),
    FOREIGN KEY (Customer_SSN) REFERENCES CUSTOMER(SSN)
        ON DELETE SET NULL        ON UPDATE CASCADE
);
```

## ★ Inserting values into ACCOUNT table

```
-- Insert 5 records into the ACCOUNT table
```

```

INSERT INTO ACCOUNT (Acc_no, Acc_type, Balance, Recent_Access_Date,
Customer_SSN)
VALUES
(101, 'CHECKING', 2000.00, '2024-01-10', '111223333'),    -- John Doe
(102, 'SAVING', 5000.00, '2024-02-01', '222334444'),      -- Jane Smith
(103, 'MONEY MARKET', 10000.00, '2024-03-15', '333445555'), -- Alice
Johnson
(104, 'LOAN ACCOUNT', 15000.00, '2024-04-01', '444556666'), -- Bob Williams
(105, 'CHECKING', 3000.00, '2024-05-10', '555667777');    -- Charlie Brown

```

Result Grid					
Filter Rows:		Search		Edit:	
Acc_no	Acc_type	Balance	Recent_Access_Date	Customer_SSN	
101	CHECKING	2000.00	2024-01-10	111223333	
102	SAVING	5000.00	2024-02-01	222334444	
103	MONEY MARKET	10000.00	2024-03-15	333445555	
104	LOAN ACCOUNT	15000.00	2024-04-01	444556666	
105	CHECKING	3000.00	2024-05-10	555667777	
NULL	NULL	NULL	NULL	NULL	

## ★ Now creating subclasses of ACCOUNT table

```

CREATE TABLE CHECKING (
    Acc_no          INTEGER      NOT NULL,
    Overdraft       DECIMAL(10, 2) NOT NULL, -- Overdraft limit for
checking accounts
    PRIMARY KEY (Acc_no),
    FOREIGN KEY (Acc_no) REFERENCES ACCOUNT(Acc_no) ON DELETE CASCADE
);

CREATE TABLE SAVING (
    Acc_no          INTEGER      NOT NULL,
    Interest        DECIMAL(5, 2) NOT NULL, -- Interest rate for saving
accounts
    PRIMARY KEY (Acc_no),
    FOREIGN KEY (Acc_no) REFERENCES ACCOUNT(Acc_no) ON DELETE CASCADE
);

CREATE TABLE MONEY_MARKET (
    Acc_no          INTEGER      NOT NULL,
    Interest        DECIMAL(5, 2) NOT NULL, -- Interest rate for money
market accounts
    PRIMARY KEY (Acc_no),
    FOREIGN KEY (Acc_no) REFERENCES ACCOUNT(Acc_no) ON DELETE CASCADE
);

```

```

CREATE TABLE LOAN_ACCOUNT (
    Acc_no          INTEGER      NOT NULL,
    Interest        DECIMAL(5, 2) NOT NULL, -- Interest rate for loan
accounts
    PRIMARY KEY (Acc_no),
    FOREIGN KEY (Acc_no) REFERENCES ACCOUNT(Acc_no) ON DELETE CASCADE
);

```

## ★ Inserting values into all these account types

```

-- Inserting 5 records into the CHECKING table
INSERT INTO CHECKING (Acc_no, Overdraft)
VALUES
(101, 500.00), -- Checking account for John Doe
(102, 1000.00), -- Checking account for Jane Smith
(103, 1500.00), -- Checking account for Alice Johnson
(104, 200.00), -- Checking account for Bob Williams
(105, 300.00); -- Checking account for Charlie Brown

-- Inserting 5 records into the SAVING table
INSERT INTO SAVING (Acc_no, Interest)
VALUES
(101, 3.50), -- Saving account for Jane Smith
(102, 2.75), -- Saving account for John Doe
(103, 4.00), -- Saving account for Alice Johnson
(104, 3.25), -- Saving account for Bob Williams
(105, 2.80); -- Saving account for Charlie Brown

-- Inserting 5 records into the MONEY_MARKET table
INSERT INTO MONEY_MARKET (Acc_no, Interest)
VALUES
(101, 2.25), -- Money Market account for Alice Johnson
(102, 2.50), -- Money Market account for John Doe
(103, 3.00), -- Money Market account for Jane Smith
(104, 2.75), -- Money Market account for Tom Brown
(105, 3.10); -- Money Market account for Emma Wilson

-- Inserting 5 records into the LOAN_ACCOUNT table
INSERT INTO LOAN_ACCOUNT (Acc_no, Interest)
VALUES
(101, 5.00), -- Loan account for Bob Williams
(102, 4.50), -- Loan account for John Doe
(103, 6.00), -- Loan account for Jane Smith
(104, 5.50), -- Loan account for Tom Brown
(105, 4.25); -- Loan account for Emma Wilson

```



★ Each subtype of ACCOUNT

○ CHECKING ACCOUNT

Acc_no	Overdraft	
101	500.00	
102	1000.00	
103	1500.00	
104	200.00	
105	300.00	
NULL	NULL	

○ SAVING ACCOUNT

Acc_no	Interest	
101	3.50	
102	2.75	
103	4.00	
104	3.25	
105	2.80	

○ MONEY-MARKET

Acc_no	Interest	
101	2.25	
102	2.50	
103	3.00	
104	2.75	
105	3.10	
NULL	NULL	

○ LOAN ACCOUNT

Acc_no	Interest	
101	5.00	
102	4.50	
103	6.00	
104	5.50	
105	4.25	

## ★ TRANSACTION TABLE

```
CREATE TABLE TRANSACTION (
    Tranc_code      INTEGER          NOT NULL, -- Transaction code (Primary
Key)
    Tranc_type      VARCHAR(20)      NOT NULL, -- Type of transaction
(e.g., deposit, withdrawal, transfer)
    Charge          DECIMAL(10, 2),    -- Any transaction charges
    Acc_no          INTEGER          NOT NULL, -- Foreign Key referencing
ACCOUNT table
    Amount          DECIMAL(10, 2) NOT NULL, -- Amount involved in the
transaction
    Date            DATE              NOT NULL, -- Date of transaction
    Hour            TIME              NOT NULL, -- Hour of transaction
    PRIMARY KEY (Tranc_code),
    FOREIGN KEY (Acc_no) REFERENCES ACCOUNT(Acc_no)
        ON DELETE CASCADE      ON UPDATE CASCADE
);
```

```
-- Insert 5 records into the TRANSACTION table
INSERT INTO TRANSACTION (Tranc_code, Tranc_type, Charge, Acc_no, Amount,
Date, Hour)
VALUES
(1, 'WD', 0.00, 101, 500.00, '2024-01-10', '10:30:00'),    -- John Doe's
withdrawal of 500 at 10:30 AM
(2, 'CD', 1.50, 102, 200.00, '2024-02-01', '12:00:00'), -- Jane Smith's
deposit of 200 at 12:00 PM
(3, 'CD', 5.00, 103, 1500.00, '2024-03-15', '14:15:00'), -- Alice Johnson's
deposit of 1500 at 2:15 PM
(4, 'WD', 0.00, 104, 1000.00, '2024-04-01', '09:00:00'), -- Bob Williams's
withdrawal of 1000 at 9:00 AM
(5, 'WD', 2.00, 105, 300.00, '2024-05-10', '16:45:00'); -- Charlie Brown's
withdrawal of 300 at 4:45 PM
```

[illegible]

# UPDATE AND DELETE OPERATIONS

## BRANCH TABLE

BranchID	Name	City	Address	Asset	Mgr_SSN
1	Chase Bank	New York	123 Broadway St	Financial	123456789
2	TD Bank	Jersey City	456 Market St	Financial	987654321
3	Wells Fargo	Newark	789 Main St	Financial	234567890
4	Bank of America	Hoboken	101 River Rd	Financial	345678901
5	Citibank	Manhattan	202 Park Ave	Financial	456789012
NULL	NULL	NULL	NULL	NULL	NULL

## **UPDATE**

This query updates the City and Address for the branch with BranchID 3 (Wells Fargo).

```
UPDATE BRANCH
SET City = 'Elizabeth', Address = '456 New Rd'
WHERE BranchID = 3;
```

BranchID	Name	City	Address	Asset	Mgr_SSN
1	Chase Bank	New York	123 Broadway St	Financial	123456789
2	TD Bank	Jersey City	456 Market St	Financial	987654321
3	Wells Fargo	Elizabeth	456 New Rd	Financial	234567890
4	Bank of America	Hoboken	101 River Rd	Financial	345678901
5	Citibank	Manhattan	202 Park Ave	Financial	456789012
NULL	NULL	NULL	NULL	NULL	NULL

## **DELETE**

This query deletes branchID 2 and has a cascade effect on the referencing table.

```
DELETE FROM BRANCH
WHERE BranchID = 2;
```

BranchID	Name	City	Address	Asset	Mgr_SSN
1	Chase Bank	New York	123 Broadway St	Financial	123456789
2	TD Bank	Jersey City	456 Market St	Financial	987654321
3	Wells Fargo	Elizabeth	456 New Rd	Financial	234567890
4	Bank of America	Hoboken	101 River Rd	Financial	345678901
5	Citibank	Manhattan	202 Park Ave	Financial	456789012
NULL	NULL	NULL	NULL	NULL	NULL

---

## EMPLOYEE TABLE

[illegible]**UPDATE**

```
UPDATE EMPLOYEE
SET Telephone = '8622149904'
WHERE Name = 'Lakshya Saharan';
```

[illegible]

**DELETE**

```
DELETE FROM EMPLOYEE
WHERE BranchID = 3;
```

[illegible]

## LOAN TABLE

Loan_Number	Loan_Amount	Monthly_repay	BranchID
1001	50000	1500	1
1002	30000	900	2
1003	25000	750	3
1004	100000	2500	4
1005	15000	450	5
NULL	NULL	NULL	NULL

## UPDATE

This query updates the Loan\_Amount and Monthly\_repay for all loans that have a Loan\_Amount greater than 30,000 and belong to the 'Bank of America' branch (BranchID = 4). It increases the Loan\_Amount by 10% and the Monthly\_repay by 5%

```
UPDATE LOAN
SET
    Loan_Amount = Loan_Amount * 1.10,      -- Increase Loan_Amount by 10%
    Monthly_repay = Monthly_repay * 1.05  -- Increase Monthly_repay by 5%
WHERE
    Loan_Amount > 30000
    AND BranchID = 4;  -- Bank of America (BranchID = 4)
```

Loan_Number	Loan_Amount	Monthly_repay	BranchID
1001	50000	1500	1
1002	30000	900	2
1003	25000	750	3
1004	110000	2625	4
1005	15000	450	5
NULL	NULL	NULL	NULL

## DELETE

```
DELETE FROM LOAN
WHERE
    Loan_Amount <= 20000
    AND BranchID > 3;
```

Loan_Number	Loan_Amount	Monthly_repay	BranchID
1001	50000	1500	1
1002	30000	900	2
1003	25000	750	3
1004	110000	2625	4
NULL	NULL	NULL	NULL

---

## CUSTOMER TABLE

[illegible]**UPDATE**

Update the City of "John Doe" from "New York" to "Brooklyn."

```
UPDATE CUSTOMER
SET City = 'Brooklyn'
WHERE Name = 'John Doe';
```

[illegible]

**DELETE**

The record for "Emma Wilson" is deleted from the CUSTOMER table

```
DELETE FROM CUSTOMER
WHERE Name = 'Emma Wilson';
```

[illegible]

## ACCOUNT TABLE

Acc_no	Acc_type	Balance	Recent_Access_Date	Customer_SSN
101	CHECKING	2000.00	2024-01-10	111223333
102	SAVING	5000.00	2024-02-01	222334444
103	MONEY MARKET	10000.00	2024-03-15	333445555
104	LOAN ACCOUNT	15000.00	2024-04-01	NULL
105	CHECKING	3000.00	2024-05-10	555667777
NULL	NULL	NULL	NULL	NULL

### UPDATE

The Balance for Jane Smith's account is increased by \$500

```
UPDATE ACCOUNT
```

```
SET Balance = Balance + 500
```

```
WHERE Customer_SSN = '222334444';
```

Acc_no	Acc_type	Balance	Recent_Access_Date	Customer_SSN
101	CHECKING	2000.00	2024-01-10	111223333
102	SAVING	5500.00	2024-02-01	222334444
103	MONEY MARKET	10000.00	2024-03-15	333445555
104	LOAN ACCOUNT	15000.00	2024-04-01	NULL
105	CHECKING	3000.00	2024-05-10	555667777
NULL	NULL	NULL	NULL	NULL

### DELETE

The record for Alice Johnson's account is deleted from the ACCOUNT table based on her Customer\_SSN.

```
DELETE FROM ACCOUNT
```

```
WHERE Customer_SSN = '333445555';
```

Acc_no	Acc_type	Balance	Recent_Access_Date	Customer_SSN
101	CHECKING	2000.00	2024-01-10	111223333
102	SAVING	5500.00	2024-02-01	222334444
104	LOAN ACCOUNT	15000.00	2024-04-01	NULL
105	CHECKING	3000.00	2024-05-10	555667777
NULL	NULL	NULL	NULL	NULL

---





# NORMALISATION

## BRANCH Table

**Primary Key:** BranchID

### Functional Dependencies:

1. BranchID → Name, City, Address, Asset, Mgr\_SSN

This relation satisfies **1NF** as all values are atomic, and there are no multivalued or repeating groups. The key BranchID uniquely identifies all other attributes.

There are no partial dependencies or transitive dependencies in BRANCH. Each non-key attribute depends fully on BranchID. Hence, the BRANCH table is already in **3NF**.

BranchID	Name	City	Address	Asset	Mgr_SSN
1	Downtown	New York	123 Elm St.	\$5,000,000	123456789
2	Midtown	Chicago	456 Oak St.	\$3,000,000	987654321
3	Uptown	Los Angeles	789 Pine St.	\$2,500,000	654987321
4	East End	Miami	101 Maple Ave.	\$4,000,000	567890123
5	West End	Seattle	202 Birch Dr.	\$1,200,000	345678901

## CUSTOMER Table

**Key:**

- **Primary Key:** SSN (Customer's Social Security Number)

### Functional Dependencies:

1. SSN → Name, Password, Apt\_no, Street, City, State, Zip, Personal\_banker, Loan\_no
  - Each customer's information is uniquely identified by their SSN.
2. **Transitive Dependencies:**
  - Loan\_no → Loan\_Amount, Monthly\_repay (via the LOAN table).
  - Personal\_banker → Name (of the employee) (via the EMPLOYEE table).

These transitive dependencies violate 3NF because attributes like Loan\_Amount and Monthly\_repay or the name of the Personal\_banker are indirectly dependent on the customer's SSN through Loan\_no or Personal\_banker.

### 1. Eliminating Transitive Dependency on Loan\_no:

- Separate loan-related details into a new table or depend on the existing LOAN table to handle Loan\_Amount, Monthly\_repay, etc.
- Retain only Loan\_no in the CUSTOMER table to establish a foreign key relationship.

### 2. Eliminating Transitive Dependency on Personal\_banker:

- The Personal\_banker attribute (foreign key referencing EMPLOYEE) creates a dependency on employee details. Rely on the existing EMPLOYEE table for additional information about the banker.
- Keep only Personal\_banker as a foreign key in the CUSTOMER table.

### 3. Breaking Down Address Details:

- If required, you could create a separate table for Address (normalization of address components). However, if the address is specific to the customer and does not repeat across customers, it can remain part of the CUSTOMER table without violating 3NF.

```
CREATE TABLE CUSTOMER (  
    SSN          CHAR(9)   NOT NULL,  
    Name         VARCHAR(20) NOT NULL,  
    Password     VARCHAR(200),  
    Apt_no       INTEGER,  
    Street       VARCHAR(20),  
    City         VARCHAR(20),  
    State        VARCHAR(20),  
    Zip          CHAR(9),  
    Personal_banker CHAR(9),  
    Loan_no      INTEGER,  
    PRIMARY KEY (SSN),  
    FOREIGN KEY (Personal_banker) REFERENCES EMPLOYEE(SSN)  
        ON DELETE SET NULL ON UPDATE CASCADE,  
    FOREIGN KEY (Loan_no) REFERENCES LOAN(Loan_number)  
        ON DELETE SET NULL ON UPDATE CASCADE  
);
```

SSN	Name	Password	Apt_no	Street	City	State	Zip	Personal_banker	Loan_no
123456789	John Customer	Cust@123	101	Elm Street	New York	NY	123456789	654987321	1
987654321	Alice Walker	W@lk3r!	202	Oak Avenue	Chicago	IL	234567890	123456789	2
654987321	Bob Johnson	J@hns0n	303	Maple Blvd	Los Angeles	CA	345678901	987654321	NULL
567890123	Sarah Smith	S@r@h2023	404	Pine Lane	Miami	FL	456789012	654987321	3
345678901	Mike Brown	B1rown@2023	505	Birch Drive	Seattle	WA	567890123	567890123	4

1. **No Partial Dependencies:**

- Every non-key attribute is fully dependent on the primary key SSN.

2. **No Transitive Dependencies:**

- Details about loans (Loan\_Amount, Monthly\_repay) are stored in the LOAN table.
- Information about the personal banker (Name, etc.) is stored in the EMPLOYEE table.

This structure ensures that the CUSTOMER table is in **3NF**.

## ACCOUNT Table

**Primary Key:** Acc\_no (Account Number)

**Functional Dependencies:**

1.  $\text{Acc\_no} \rightarrow \text{Acc\_type}, \text{Balance}, \text{Recent\_Access\_Date}, \text{Customer\_SSN}$ 
  - Each account's details are uniquely identified by Acc\_no.
2. **Transitive Dependency:**
  - $\text{Customer\_SSN} \rightarrow \text{Name}, \text{Address}, \text{etc.}$  (via the CUSTOMER table).

## CHECKING Table

**Primary Key:** Acc\_no (Foreign key referencing ACCOUNT)

**Functional Dependencies:**

1.  $\text{Acc\_no} \rightarrow \text{Overdraft}$ 
  - Each checking account's overdraft limit is uniquely identified by Acc\_no.

## SAVING Table

**Primary Key:** Acc\_no (Foreign key referencing ACCOUNT)

**Functional Dependencies:**

1. Acc\_no  $\rightarrow$  Interest
    - Each saving account's interest rate is uniquely identified by Acc\_no.
- 

## Step 2: Normalization and Checking for 3NF

### ACCOUNT Table

The table is in **1NF** because all values are atomic. To ensure compliance with **3NF**, we analyze dependencies:

1. Acc\_no is the primary key, and all non-key attributes (Acc\_type, Balance, Recent\_Access\_Date, Customer\_SSN) are fully dependent on it.
2. There is no partial dependency because Acc\_no is a single-column key.
3. There is a transitive dependency via Customer\_SSN, which references the CUSTOMER table. Customer details (Name, Address, etc.) are not stored here, so no transitive dependency remains.

### CHECKING Table

- The table satisfies **1NF**, **2NF**, and **3NF** because all attributes (Acc\_no and Overdraft) are fully functionally dependent on the primary key Acc\_no.

### SAVING Table

- Similar to CHECKING, the table satisfies **1NF**, **2NF**, and **3NF** because all attributes (Acc\_no and Interest) are fully functionally dependent on the primary key Acc\_no.
- 

## Final Normalized Schema

The tables are already normalized and satisfy **3NF**. No further decomposition is needed.

**ACCOUNT Table**

Acc_no	Acc_type	Balance	Recent_Access_Date	Customer_SSN
1001	Checking	5000.00	2024-12-01	123456789
1002	Saving	2000.00	2024-11-20	987654321
1003	Checking	7500.00	2024-12-05	654987321
1004	Saving	12000.00	2024-11-30	567890123
1005	Checking	3000.00	2024-12-08	345678901

**CHECKING Table**

Acc_no	Overdraft
1001	1000.00
1003	1500.00
1005	500.00

**SAVING Table**

Acc_no	Interest
1002	2.50
1004	3.00

# GROUP, HAVING AND NESTED QUERIES

## GROUP BY

We want to find out the total **Loan\_Amount** issued by each branch, and the average monthly repayment for those loans. The **LOAN** table contains the data on loans, and the **BRANCH** table contains data on branches.

```
SELECT
    B.Name AS Branch_Name,
    SUM(L.Loan_Amount) AS Total_Loan_Amount,
    AVG(L.Monthly_repay) AS Average_Monthly_Repayment
FROM
    LOAN L
JOIN
    BRANCH B ON L.BranchID = B.BranchID
GROUP BY
    B.Name;
```




Branch_Name	Total_Loan_Amount	Average_Monthly_Repayment
Chase Bank	50000	1500.0000
TD Bank	30000	900.0000
Wells Fargo	25000	750.0000
Bank of America	100000	2500.0000
Citibank	15000	450.0000

## GROUP BY and HAVING

The total balance for each customer, but only show customers with a total balance greater than \$10,000

- SUM(ACCOUNT.Balance): Calculates the total balance for each customer.
- HAVING SUM(ACCOUNT.Balance) > 10000: Filters the results to include only customers whose total balance is greater than \$10,000.

```
SELECT
    CUSTOMER.SSN,
    CUSTOMER.Name,
    SUM(ACCOUNT.Balance) AS Total_Balance
FROM
    CUSTOMER
JOIN
    ACCOUNT ON CUSTOMER.SSN = ACCOUNT.Customer_SSN
GROUP BY
    CUSTOMER.SSN, CUSTOMER.Name
HAVING
    SUM(ACCOUNT.Balance) > 10000;
```

Result Grid   Filter Rows: <input type="text" value="Search"/>				Export: 
SSN	Name	Total_Balance		
444556666	Emma Wilson	15000.00		

## NESTED QUERY WITH ALL

This query selects employee names with dependent names which have work experience less than **“Lakshya Saharan”**

```
SELECT E.Name, E.Dependent_name
FROM EMPLOYEE E
WHERE E.Length_of_emp < ALL (
    SELECT E2.Length_of_emp
    FROM EMPLOYEE E2
    WHERE Name = 'Lakshya Saharan'
);
```

Name	Dependent_name
Naga Sathwik	Prudhvi
Eren Canan	Scarlett Johansson
Donald Trump	Ivanka Trump
Arjun Reddy	Ramesh Reddy

## NESTED QUERY WITH IN

This query gets employee which for branch which are located in “**New York**” or “**Hoboken**”

```
SELECT *
FROM EMPLOYEE E
WHERE E.BranchID IN (
    SELECT B.BranchID
    FROM BRANCH B
    WHERE B.City = 'New York' OR B.City = 'Hoboken'
);
```

[illegible]



# CONCLUSION

The **most difficult step** was ensuring **normalization of relations**. Identifying functional dependencies and resolving transitive dependencies to achieve 3NF required careful analysis of the schema and the business requirements.

Integrating **foreign key constraints** with cascading rules for updates and deletions was tricky, especially for circular dependencies like EMPLOYEE managing branches and being supervised by other employees.

Writing **complex SQL queries** involving nested subqueries and aggregate functions (GROUP BY and HAVING) was challenging but rewarding.

- **Easiest Steps:**

- Designing the **EER diagram** was relatively straightforward because the business rules provided a clear structure for entity relationships.
- Populating tables with sample data was simple once the schema was finalized and tested.

- **Unexpected Learnings:**

- The **importance of cascading rules** for referential integrity became evident. Cascades simplify data management but require careful planning to avoid unintended data loss.
- Learning how to optimize queries for better performance, especially when dealing with large datasets.

- **What We Would Do Differently:**

- Start with a **simpler schema** and iteratively add complexity instead of tackling all business rules at once.
- Dedicate more time to the **normalization process** early to avoid redesigns during later stages.
- Automate the testing of SQL queries and schema validation using scripts to save time during debugging.

## **Final Comments and Conclusions**

This project provided an excellent hands-on experience in database design, implementation, and querying. It strengthened our understanding of:

- Translating real-world business rules into a database schema.
- The significance of normalization for reducing redundancy and ensuring data integrity.
- Writing and testing SQL queries to meet specific business requirements.

Overall, the project was a rewarding experience that enhanced our technical and collaborative skills. We now feel more confident in tackling real-world database management challenges.