BSA BANK

Team: AMIGOS

Akhil kumar Bachu
abachu

Briyana Benjamin Rana
brana

Shankarram Saravanan
saravan4

Abstract

This document is a proposal to address one of the major challenges in the real world - Bank Management System. Using Database Management concepts, we propose a solution for efficiently managing a bank. Finally, we provide a demo application to demonstrate how it might be used by an end user.

I.PROBLEM STATEMENT

The ever-growing technology made things ease for man over the past years, and it would be hectic work and includes more time for customers to visit a bank and stand in line for hours just to check their transaction details, personal details linked to the account, loan balance, or to have a look at their loan summary. To aid the customers and the bank employees, we have proposed a DB that is agile, scalable, and up-to-date and will make it easier for the customers and bank employees to check the loan account summary.

This will also help managers as they need to check transaction details or update the personal details of a customer. So, it is necessary to find and formulate the relations in such a way that it can effectively retrieve everything about a customer and the customer's account. Furthermore, we also must check the speed and memory of the DB too. Considering all the above constraints, we have decided to have 7 relations, which we felt would be the best way possible to comprehend a customer's details.

II. WHY NOT EXCEL?

Normally, the database is scalable. The maximum number of rows that Spreadsheet tools frequently allow is 10,048,567 which limits the product's potential to scale. For instance, if our product could only be used by 10,048,567 it would appear unattractive and lose market share.

Cross-table relationships can be created easily, however creating relationships between two sheets in programs like Excel is nearly impossible.

Constraints can be applied to table schemas to prevent duplicate entries from appearing in a table. There is no method to set any restrictions on the data that is contributed to a spreadsheet.

Databases have API that can interact with the application, making this work easier, as the table is made to talk with a human through an application. Spreadsheets don't have any CRUD operation to support APIs.

A small team of five persons or for personal usage may use a spreadsheet. However, using a spreadsheet simultaneously by hundreds of users might lead to chaos. Databases frequently offer application developers high-performance APIs that can support thousands of concurrent users by using multithreading and concurrency.

III. TARGET USERS:

Bank Employees (Admin):

They are the bank employees, who will have access to the customer id and can check the transaction details of a customer, every time they login into they need to give their emp_id, password, and the account_no whose loan summary they want to look into. **Operations Allowed**: Read, update.

Customers(User):
They are bank customers, who have access to their loan details, personal details, account details, and loan transaction details. **Operation Allowed**: Read

Real-Life Scenario:
A customer can open his account through his login credentials and account number and check his transactional details, personal details associated with his account, and account summary. A bank employee can check a customer's account by login in with his login credentials and account number of the customer, and check transactional details, checking account balance, and checking or updating personal details.

IV. DESIGN

A. ER DIAGRAM

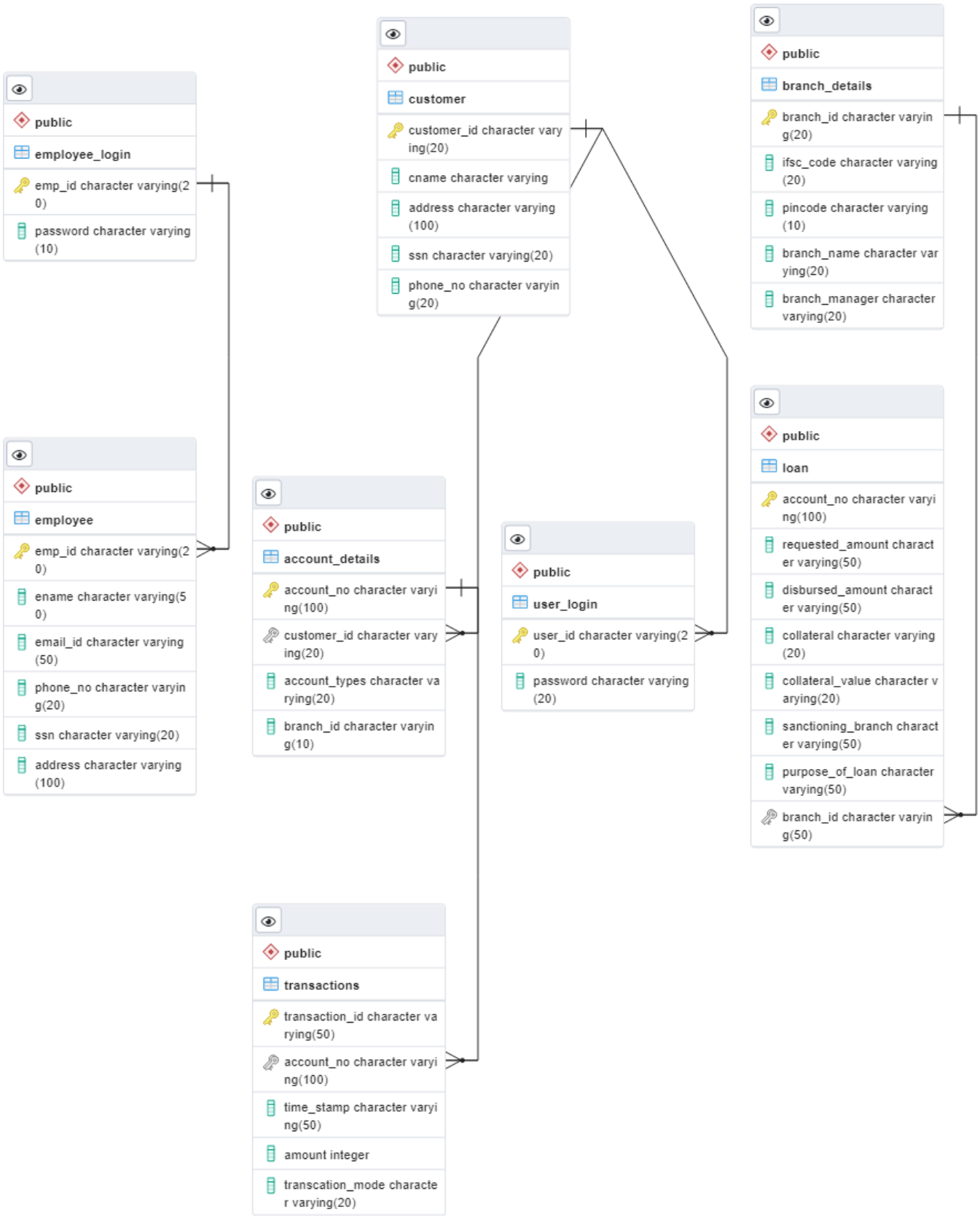


Figure 1: ER MODEL FOR BANK MANAGEMENT SYSTEM

B. Database schema and description

Relation 1: Customer
Schema: Customer (Customer_id varchar(20) primary key, Name varchar(20), Email_id varchar(30), Address varchar(100), SSN varchar(20), Phone_no varchar(20))

Attributes:
1) Customer_id
a) It is used for uniquely identify all the user.
b) It is the Primary Key. It will be autogenerated.
c) It acts as a foreign key for the login table.
d) Datatype – Varchar(10)

2) Cname
a) Name of the customer (contains both First and last name)
b) Datatype – Varchar(10)

3) Address
a) This contains the address of the Customer.
b) Datatype – Varchar(100)

4) SSN
a) This contains the Social Security number of the customer.
b) It should be a Unique, NOT NULL type.
c) Datatype – Varchar(10)

5) Phone_no
a) Phone number of the customer.
b) It should be a Not null Type.
C) Datatype – Varchar(15)

Relation 2: Employee
Schema: Employee(Emp_id varchar(20) Primary Key, Ename varchar(20), Email_id varchar(30), Phone varchar(20), Branch_name varchar(20), Branch_id varchar(10) Foreign key references Branch_details(Branch_id)).

Attributes:
1) Employee_id
a) It is a Primary Key that helps in identifying all the customers uniquely.
b) It acts as a foreign key for the Login relation.
b) Datatype – varchar(20)

2) Ename
a) It contains the name of the employee
b) Datatype – varchar(20)

3) Email_id
a) It is necessary for the employee to have an email, so it has to be a Not NULL and unique value.
b) Datatype – varchar(30)

4) SSN
a) This contains the Social Security number of the customer.
b) It should be a Unique, NOT NULL type.
c) Datatype – Varchar(20)

5) Phone_no
a) It contains the contact number of the employee.
b) It must be a Not null and unique value for every employee.
c) Datatype – Varchar(20)

5) Address
a) This contains the address of the Customer.
b) Datatype – Varchar(100)

Relation 3: Customer_login
Schema: Employee(User_id varchar(10) Primary Key, Password varchar(10)).

Attributes:
1) User_id
a) It is a Primary Key that helps in identifying all the customers(users) uniquely.
b) Datatype – varchar(10)

2) Password
a) It contains the password of the user
b) Datatype – varchar(10)

Relation 4: Employee_login
Schema: Employee(Emp_id varchar(10) Primary Key, Password varchar(10)).

Attributes:
1) Emp_id
a) It is a Primary Key that helps in identifying all the Employees uniquely.
b) Datatype – varchar(10)

2) Password
a) It contains the password of the Employee who works in the bank
b) Datatype – varchar(10)

Relation 5: Account_details
Schema: Account_details(Account_no int Primary key, Customer_id int(17) Foreign key references Customer(Customer_id) , Account_type varchar(20), Branch_id varchar(20))

Attributes:
1) Account_no
a) It contains the account number of the customer.
b) Datatype – int(17)

2) Customer_Id
a) It contains the customer_id and acts as a foreign key that references the relation customer
b) Datatype – Varchar(10)

3) Account_type
a) It says the type of account that the user holds.
b) Datatype – varchar(20)

4)Branch_name
a) It contains the id of the branch and acts as a foreign key that references the Branch table
b) Datatype – varchar(20)

Relation 6: Transaction
Schema: Transaction(Transaction_id varchar(15) Primary key , Account_no int foreign key references Account_details(Account_no), Time_stamp Date, Amount int, Transaction_mode varchar(20))

Attributes:
1)Transaction_Id
a) It is used to uniquely identify all the users.
b) Transaction is a unique code that gets generated for each transaction.
c) It is a primary key
d) Datatype - varchar(15)

2)Account_no
a) It contains the account number of the user.
b) It is a foreign key that references relation Account_details
c) Datatype – int

3) Time_stamp
a) It contains the time at which the transaction occurred between accounts.
b) Datatype - Date

4) Amount
a) It contains the amount which either credited or debited using the sign(+,-).
b) Datatype – Int

5)Transaction_mode
a) It contains the type of mode in which the transaction occurred (eg: Deposit, bank transfer, etc)
b)Datatype : varchar(20)

Relation 7: Loan
Schema: Loan(Loan_id int primary key, Account_no int Foreign key references Account_details(Account_no), Sanctioned_amount int, Requested_amount int, Disbursed amount int, Collateral varchar(20), Collateral_value int, Sanctioning_branch varchar(20), Purpose_of_loan varchar(20), Branch_id int Foreign key references Branch_details(Branch_id))

Attributes:
1)Loan_Id
a) It is unique for all the customers.
b) It acts as a primary key
c) Datatype - Int

2) Account_no
a) It contains the account number of the customer who applied for the loan.
b) It is a foreign key that references relation Account_details.
c) Datatype – int

3)Sanctioned_amount
a)It contains the amount sanctioned by the amount to the customer who applied for the loan.
b) Datatype - Int

4) Requested_amount
a) It contains the loan amount requested by the customer.
b) Datatype -Int

5)Collateral_type
a) It contains the amount disbursed by the customer to date.
d) Datatype - Int

6)Sanctioned_amount
a)It contains the amount sanctioned by the amount to the customer who applied for the loan.
b) Datatype - Int

7) Collateral
a) It contains the value of the collateral (eg: collateral: property value)
b) Datatype - Int

8)Purpose_of_loan
a) It contains the purpose for which the loan has been taken.
b) Datatype – varchar(20)

9) Branch_Id
a)Required to map with relation branch_details
b) It contains the id of the branch
c) Datatype – Varchar(10)

Relation 8: Branch_details
Schema: Branch_details(Branch_id int Primary key, IFSC code varchar(20), Pincode int, Branch_name varchar(20), Branch_manager varchar(20))

Attributes:
1)Branch_Id

a) Required to map relationship with Employee
b) It contains the id of the branch
c) Datatype – Varchar(10)

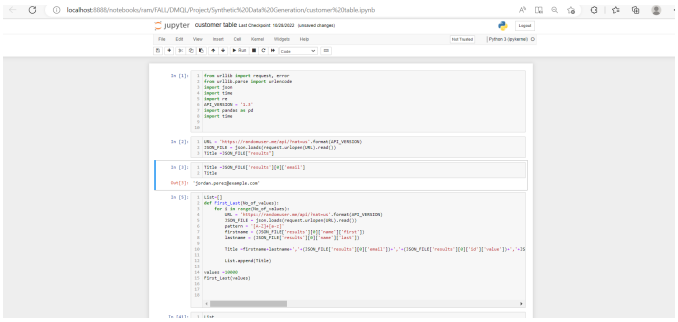
2) IFSC_code
a) It refers to the unique code which refers the locality of the bank.
b) Datatype – varchar(10)
c) Datatype – int

3)Pincode
a) It contains the Pincode of the bank.
b) Datatype – int

4) Branch_name
a) It is required to map with relation Employee and Account_Details.
b) Datatype – varchar(10)

5)Branch_manager
a) It contains the name of each bank manager
b) Datatype – varchar(20)

DATA SOURCE:'
We have generated synthetic data using python, and the API we had used to generate data is mentioned below .
https://randomuser.me/



V. FUNCTIONAL DEPENDENCY and BCNF

FD'S for Relation: **Customer**:

Customer_id → cname is in BCNF.
Customer_id → address is in BCNF.
Customer_id → SSN → Violates BCNF.
Customer_id → phone_no is in BCNF.

Customer relation will be decomposed to R1(customer_id, SSN) , R2(customer_id, address, cname, phone_no)
As SSN is a prime attribute it violates the FD customer_id → SSN. Now the table is in BCNF form.

FD's for Relation **account_details table**:

Account_no → customer_id.
Account_no → account_type.
Account_no → branch_id.

As all the R.H.S attributes are non-prime attributes and L.H.S is the only candidate key of the table. It is already in the form of BCNF.

FD's for Relation: **branch_details** :

Branch_id → ifsc_code,pincode, branch_name,branch_manager.
Here all the R.H.S attributes are non-prime attributes. And the table has only one candidate key which is Branch_id hence the table is having all functional dependencies in BCNF form.

FD's for Relation: **Transaction table**:

transaction_id → account_no, time_stamp, amount, transaction_mode.
Here all the R.H.S attributes are non-prime attributes. And the table has only one candidate key which is transaction_id. In this table account_no is a foreign key taking reference from the account_details, thus it has duplicates. Hence the table is having all functional dependencies in BCNF form.

FD's for Relation : **Loan table**:

account_no → requested_amount, disbursed_amount, collateral, collateral_value, sanctioning_branch, purpose_of_loan, branch_id.
Here all the R.H.S attributes are non-prime attributes. And the table has only one candidate key which is account_no. In this table, branch_id is a foreign key taking reference from the branch_details table. Hence the table is having all functional dependencies in BCNF form.

FD's for Relation: **employee table**:

emp_id → ename is in BCNF.
emp_id → email_id is in BCNF.
emp_id → ssn violates BCNF.
emp_id → phone_no is in BCNF.
emp_id → address is in BCNF.

Here as ssn is not a non-prime attribute it will violate the BCNF. Thus decomposition of relations will take place. Hence the table employee will be decomposed into 2 relations employee1(emp_id,ssn) and employee2(emp_id, ename, email_id, address, phone_no) Now the relations are in BCNF form

FD's for Relation: **employee_login**:

Emp_id → password.
Here the table is already in the form of BCNF as there is only one candidate key emp_id. and the password is a non-prime attribute.

FD's for Relation : **user_login**:

user_id → password.
Here the table is already in the form of BCNF as there is only one candidate key user_id and the password is a non-prime attribute.

Instead of importing all the data manually, we have written a python script to insert all the data into the SQL database.

PYTHON CODE TO INSERT RECORDS INTO THE DATABASE

```
def Customer(conn):
    #fields = ['Customer_Id', 'Name', 'Email', 'date', 'SSN', 'Phone', 'Address']
    df = pd.read_csv('Customer.csv', skipinitialspace=True)


    #df = df[['Customer_Id', 'Name', 'Email', 'date', 'SSN', 'Phone', 'Address']]

    execute_many_sql_statement("INSERT INTO Customer VALUES(?,?,?,?,?,?);", conn, df.to_numpy())

    conn.commit()

Customer(conn)
```

```
def Branch_details(conn):
    df = pd.read_csv('Branch_Details.csv', skipinitialspace=True)

    execute_many_sql_statement("INSERT INTO Branch_details VALUES(?,?,?,?,?);", conn, df.to_numpy())

    conn.commit()

Branch_details(conn)
```

```
def Account_Details(conn):
    df = pd.read_csv('Account_Details.csv', skipinitialspace=True)



    execute_many_sql_statement("INSERT INTO Acount_Details VALUES(?,?,?,?);", conn, df.to_numpy())

    conn.commit()

Account_Details(conn)
```

VI. SQL QUERY ON DATABASE

A. SELECT with GROUP BY:

```
SELECT  transaction_id, min(amount) FROM Transactions GROUP BY transaction_id
```

Transaction_id	min(amount)
0001f8b9da40c2d	-INR 18,100.00
0005190f0b74954	-INR 34,750.00
0007e3b8e2c6d5c	INR 55,225.00
000959b0661a4b6	-INR 5,725.00
0009e3c48669319	-INR 8,200.00
000d8448b935c87	-INR 39,250.00
0010f2778116861	INR 3,925.00

B. SELECT with INNER JOIN:

```
select a.branch_id, b.ifsc_code, a.account_no from Account_Details AS a
inner join Branch_Details AS b on a.Branch_id  = b.Branch_id
```

Output Messages Notifications

branch_id character varying (10)	ifsc_code character varying (20)	account_no character varying (100)
BSABUF05	BSA10000081	1032665555259110.00
BSABUF07	BSA10000021	1022457127705150.00
BSABUF02	BSA10000076	1034377768495020.00
BSABUF15	BSA10000073	1030517720272910.00
BSABUF05	BSA10000081	1018800232477030.00
BSABUF19	BSA10000037	1042911798015270.00
BSABUF16	BSA10000062	1054901254984870.00
BSABUF14	BSA10000075	1065593631272280.00

C. Insert Query:

```
INSERT INTO Customer (customer_id, cname, email_id,SSN, Phone_no, address) VALUES ( "C224495", "Akhil Kumar", "akhil@gmail.com", "716-416-8
SELECT * FROM Customer WHERE email_id = "akhil@gmail.com"
```

Customer_id	cname	Email_id	SSN	Phone_no	Address
C224495	Akhil Kumar	akhil@gmail.com	716-416-8797	+1(716)4168797	30 West Northrp

D. Update Query:

```
UPDATE Customer
SET cname = "Akhil Bachu"
WHERE customer_id = "C224495";
SELECT * FROM Customer WHERE email_id = "akhil@gmail.com"
```

Customer_id	cname	Email_id	SSN	Phone_no	Address
C224495	Akhil Bachu	akhil@gmail.com	716-416-8797	+1(716)4168797	30 West Northrp

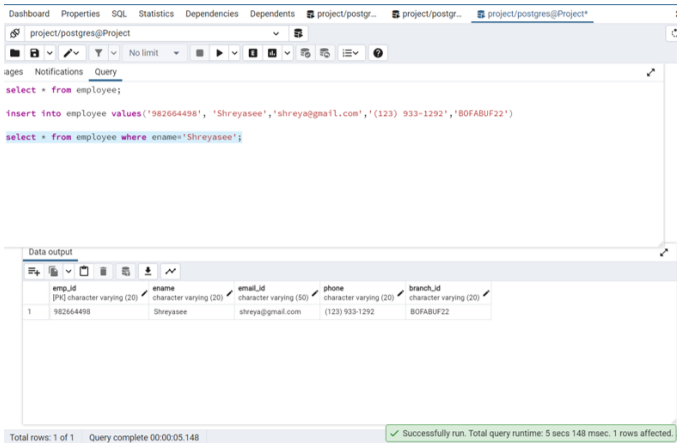
E. Delete Query:

```
DELETE FROM Customer WHERE cname = "Akhil Bachu";

SELECT * FROM Customer WHERE email_id = "akhil@gmail.com"
```

Customer.db

```
DELETE FROM Customer WHERE cname = "Akhil Bachu";

SELECT * FROM Customer WHERE email_id = "akhil@gm
```

F. INSERTION IN EMPLOYEE TABLE:

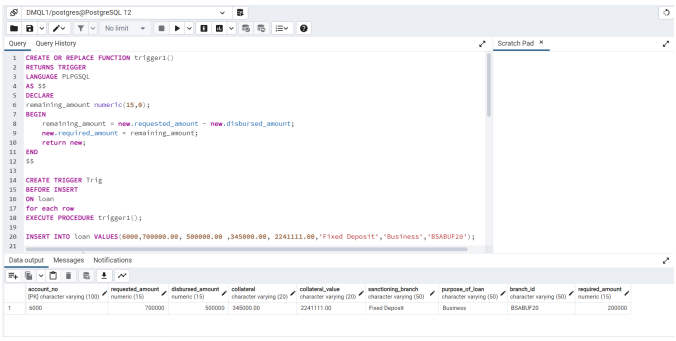


G. TRIGGER IMPLEMENTATION:

As requested_amount and disbursed_amount having values as 'INR 780,000', we changed the rows to numeric by applying the following preprocessing steps. later, a trigger was implemented on these columns.

```
UPDATE loan
SET requested_amount = SUBSTRING(requested_amount,4,100);

UPDATE loan
SET requested_amount = REPLACE(requested_amount,',','');
```

We have implemented a trigger to find the remaining amount of the loan that a customer has after a particular disbursement.



VII. QUERY ANALYSIS

For an inner join between branch_details and account_details it taking more than 5 ms,

	QUERY PLAN text
1	Hash Join (cost=16.07..236.74 rows=10000 width=87) (actual time=0.055..4.909 rows=10000 loops=1)
2	Hash Cond: ((a.branch_id)::text = (b.branch_id)::text)
3	-> Seq Scan on account_details a (cost=0.00..194.00 rows=10000 width=29) (actual time=0.022..1.060 rows=10000 loops=1)
4	-> Hash (cost=12.70..12.70 rows=270 width=116) (actual time=0.024..0.024 rows=26 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 10kB
6	-> Seq Scan on branch_details b (cost=0.00..12.70 rows=270 width=116) (actual time=0.009..0.013 rows=26 loops=1)
7	Planning Time: 0.273 ms
8	Execution Time: 5.358 ms

So we have introduced indexing to reduce the cost of the query.

Creating an index on branch_id of account_details

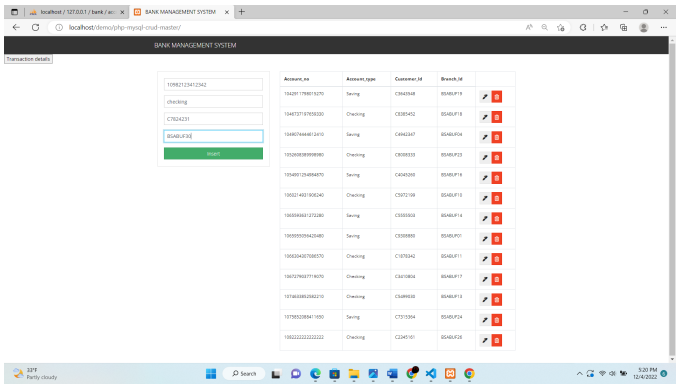
```
CREATE INDEX id
ON Account_details(branch_id)

EXPLAIN ANALYZE select a.branch_id, b.ifsc_code, a.account_no from Account_Details a
inner join Branch_Details b on a.Branch_id  = b.Branch_id
```

Now we could see that the execution time has been reduced to 4.8 ms

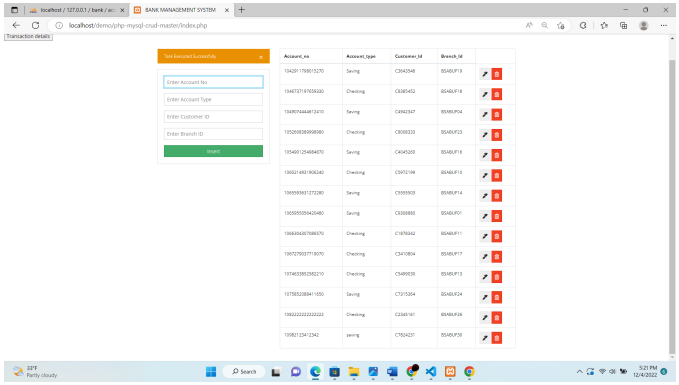
	QUERY PLAN text
1	Hash Join (cost=16.07..236.74 rows=10000 width=87) (actual time=0.079..4.418 rows=10000 loops=1)
2	Hash Cond: ((a.branch_id)::text = (b.branch_id)::text)
3	-> Seq Scan on account_details a (cost=0.00..194.00 rows=10000 width=29) (actual time=0.023..0.917 rows=10000 lo...
4	-> Hash (cost=12.70..12.70 rows=270 width=116) (actual time=0.045..0.046 rows=26 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 10kB
6	-> Seq Scan on branch_details b (cost=0.00..12.70 rows=270 width=116) (actual time=0.011..0.019 rows=26 loops=1)
7	Planning Time: 2.068 ms
8	Execution Time: 4.857 ms

VIII. APPLICATION SCREENSHOTS

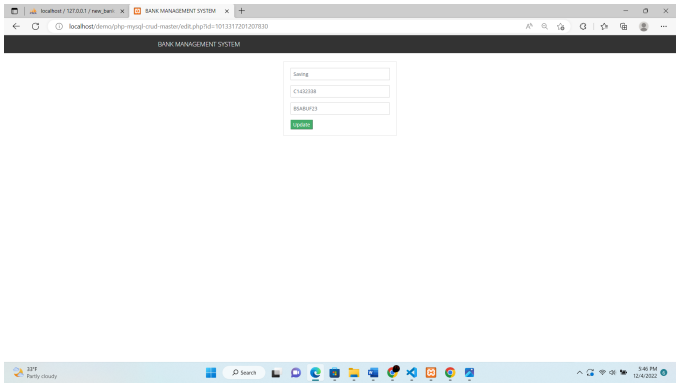
Insert Value into account_details table:



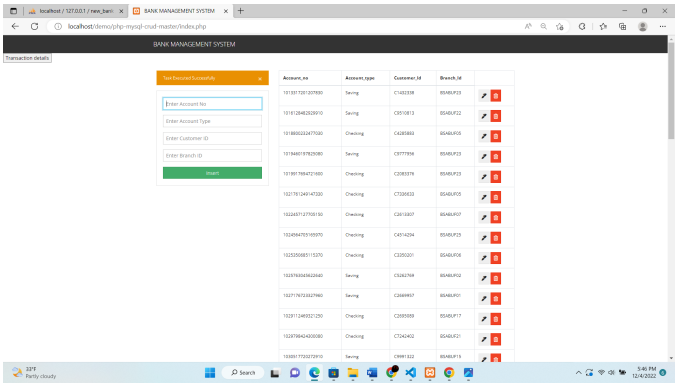
After inserting:



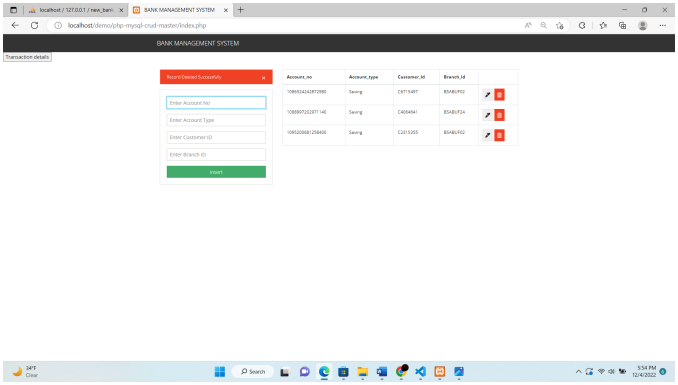
Updating details :



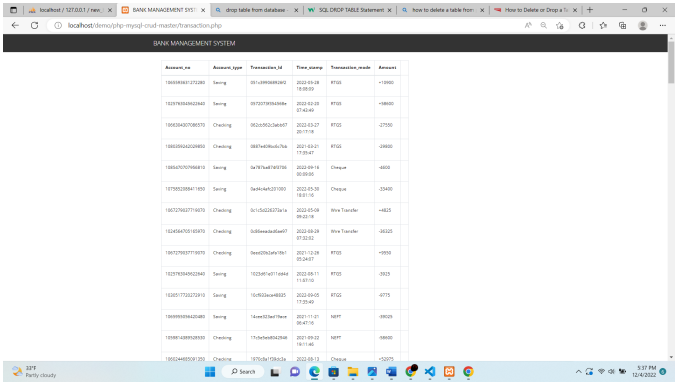
After Updating:



After deletion of few records:



INNER JOIN BETWEEN TRANSACTION AND ACCOUNT DETAILS:



IX. REFERENCES

https://www.lucidchart.com/pages/er-diagrams
https://www.studytonight.com/dbms/boyce-codd-normal-form.php