

CIS 552 DATABASE DESIGN

FINAL PROJECT

Project Title	:	Bank Management System
Submission Date	:	12/15/2023
Submitted by	:	Group 12
Group Members	:	Jameera Mahima Gujarlapudi (02117881), Sahithi Keshireddy (02084037)

Abstract

A bank management system, which oversees customer and account information as well as the second-by-second transactions it processes, will primarily address two issues. The first step is to have a third party confirm the customer's identification. After that, the loan with a set amount of money is approved depending on the customer's profile. The second is that many consumers find it difficult to withdraw cash from any ATM in any of a certain bank's branches.

Problem Statement

A bank management system is the project idea we are putting forth. We can manage customer data, account information, and second-by-second transactions with this database system. It keeps additional information and the specifics of the transaction safe. Working with millions of consumers is made easier by databases, which automate time-consuming everyday operations.

Specifically, our bank management system will address two issues. The first step is to have a third party confirm the customer's identity. After that, the loan with a set amount of money can be sanctioned based on the customer's profile validation. The second is that many users struggle to withdraw cash from any ATM in any of a bank's outlets. The money is deposited in a particular branch based on demand, and the database tracks all transactions made for that branch. After that, customers can easily withdraw their money from any ATM.

DATABASE VS EXCEL

Databases are excellent at maintaining the ACID (atomicity, consistency, isolation, and durability) qualities while easing the retrieval of information. Compared to spreadsheets, databases allow us to store different sorts of information (i.e., data types) in separate columns, which reduces the possibility of errors. Spreadsheet use raises a major concern: data storage. In our hypothetical scenario, a bank might have millions of clients, and that number would increase daily. Overall, such broad information cannot be accommodated by a spreadsheet. The search gets frantic. Databases are then useful in the situation. The capacity to store an infinite amount of data makes it simple to store a range of client data and provides speedy data update or retrieval. Ensuring that bank managers have consistent information across branches is crucial when managing numerous locations worldwide. Databases ensure reliable and clear information while also enhancing the user experience. All branches instantly see changes made in any branch. Additionally, database searching saves time and reduces the possibility of human error by automating tasks with a single command as opposed to manual labor.

The bank handles financial transactions, thus protecting client assets and data is important. The goal of this initiative is to increase users' and customers' sense of security and trust in banks. Customers may readily examine all of the information, and bank staff can conveniently maintain the database.

Target user

There are two categories of users of the database: 1. Customers 2. Bank Employees

User and Administrator

Bank staff administers the database. They can add, edit, or remove every piece of data in the database.

Only the customers' own unique information, such as loan data, transaction details, profile details, and account details, can be viewed.

Real-life scenario

A customer's information is gathered and kept in a database when they create an account with a bank. The consumer is verified by the bank in the background. The customer may wish to open a savings or checking account in accordance with his interests. The bank determines the account's limit following verification. Additionally, clients may wish to take advantage of the bank's benefits, loans, or merchandise. Banks can then search the database, provide product recommendations, and approve loans with specific loan amounts depending on the individual's profile.

Banks save a lot of data about their customers, including account types, transaction volume, number of accounts held by each customer, credit score, loan details, etc. In actuality, however, a consumer just needs to be aware of their personal information. The customer must have access to a bank management database to examine their personal information.

Banks can verify customers and impose credit and debit card limits. Employees of banks can also enhance the security of the client information by confirming the information provided and granting loans in accordance with the borrower's profile.

Database design:

Database Schema:

- Bank.
- Customer.
- Account.
- Doc Verification.
- Transactions.
- Loan.
- Atm details.

Attributes and their description with constraints

Table name bank

Table fields and Description

- branchname : Name of the branch
- branch id: ID of the branch
- routingnumber : routing number for bank
- address : Bank Address Constraints

- Primary Key : BranchId
- Unique key: routing number

Table name : Customer

Table fields and description

- customer id: Id of the customer
- firstname : First Name of customer
- lastname : Last Name of customer
- gender : Gender of customer
- age : Age of customer tenure : tenure of customer in bank
- num of products : No of products customer has
- isverified: Whether customer is verified
- address : Address of the customer
- branch id: ID of the branch
- credit score : Credit score the customer has overall
- customer status : show the customer is active or not Constraints
- Primary Key : Customerid
- Foreign key: Branchid
- NOT NULL :Age

Table name: account

Table fields and description

- account id : Account ID unique to each customer
- currency: Currency type
- customer id : Customer ID referencing customer table
- working bal : Balance of account
- category: Type of account
- branch id : ID of branch referencing Branch table
- account status : Active / Inactive Constraints
- Primary Key : account id
- Foreign key :Branch id, customer id
- NOT NULL :working bal

Table name : doc verification

Table fields and Description

- customer id: ID of customer referencing customer table
- address : Address of customer to give access to third party.

- verification status : has verification started?
- validation status : VALID/NOT VALID
- comments : Describes about the customer verification status by third party

Constraints

- Primary Key : Customer id
- Foreign key :customer id

Table fields and Description

- loanID: Id of the loan
- account ID: ID of the account referencing account table
- loan amount : amount taken as loan
- funded amt : how much is funded
- term: tenure of the loan
- int rate : Interest rate
- Installment : Amount paid
- home ownership : what kind of home customer has
- annual inc : income of employee
- verification status : verified or not
- issued date : date of loan issued
- loan status : paid or not Constraints
- Primary Key : loan id
- Foreign key : account id

Table name: atm details

Table fields and Description

- atm name : Name of the ATM
- transaction date : date of transaction
- no of withdrawals : How many withdrawals done
- total amt withdrawn: Amount drawn
- week day: When is the weekday
- working day : Holiday/weekday
- branch id: Id referencing branch table Constraints
- Primary Key : atm name, transaction date
- Foreign key : branch id

Table name: routing

Table fields and Description

- routing number
- branchname

Constraints

- Primary Key : routingnumber
- Foreign key: branch id atm details

Description: Bank is referenced for customer, account and atm details tables to refer branch id column

Referenced table Customer

Referencing table

- account
- doc verification

description: Customer is referred in account and document verification to retrieved customer id to know about customer details

Referenced table Account

Referencing table

- loan
- transaction

description: Account is referenced for account id in loan and debit account and credit account column to retrieve the account number in transaction table.

SQL Queries:

INSERT/UPDATE/DELETE for Dataset

```
INSERT QUERY

Query Query History
1 INSERT INTO customer(
2   customerid, firstname, lastname, gender, age, tenure,
3   noofproducts, is_verified, branch, address, creditscore, customer_status)
4 VALUES (178909, 'Rohith', 'Ramchettl', 'M', 24, 2, 0, 'Y', '80B8982346',
5   '34 Merrimac, 14241 ', 789, 'ACTIVE');

Data output Messages Notifications
INSERT 0 1
Query returned successfully in 183 msec.
```

UPDATE QUERY

Query Query History

```
1  
2  
3 update doc_verification set verification_status = 'completed', validation_status = 'VALID',  
4 "comments" = 'Verification Done' where customerId = 100287
```

Data output Messages Notifications

NOTICE: customer record updated
UPDATE 1
Query returned successfully in 139 msec.

DELETE QUERY

Query Query History

```
1  
2  
3 delete from account where accountid = 13919
```

Data output Messages Notifications

DELETE 1
Query returned successfully in 850 msec.

There are mainly three types of relationships used:

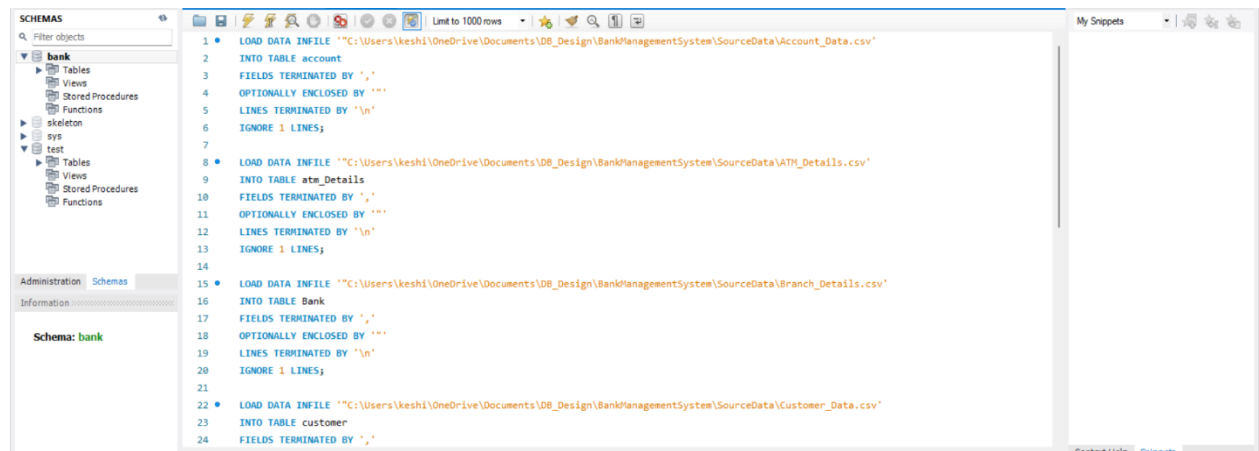
- one-to-one relationships
- one-to-many relationships
- many-to-many relationships

By establishing and maintaining these relationships using keys and SQL commands, databases organize data efficiently and ensure data consistency across multiple tables.

Record Insertion:

We have referred our data sets from

- <https://www.kaggle.com/datasets/mathchikhurn-forbank-customers>
- <https://www.kaggle.com/datasets/mrferozi/loan-datafor-dummy-bank>
- <https://www.kaggle.com/datasets/nitsbat/data-of-atmtransaction-of-xyz-bank>
- We have also generated data using <https://www.onlinedatagenerator.com/>

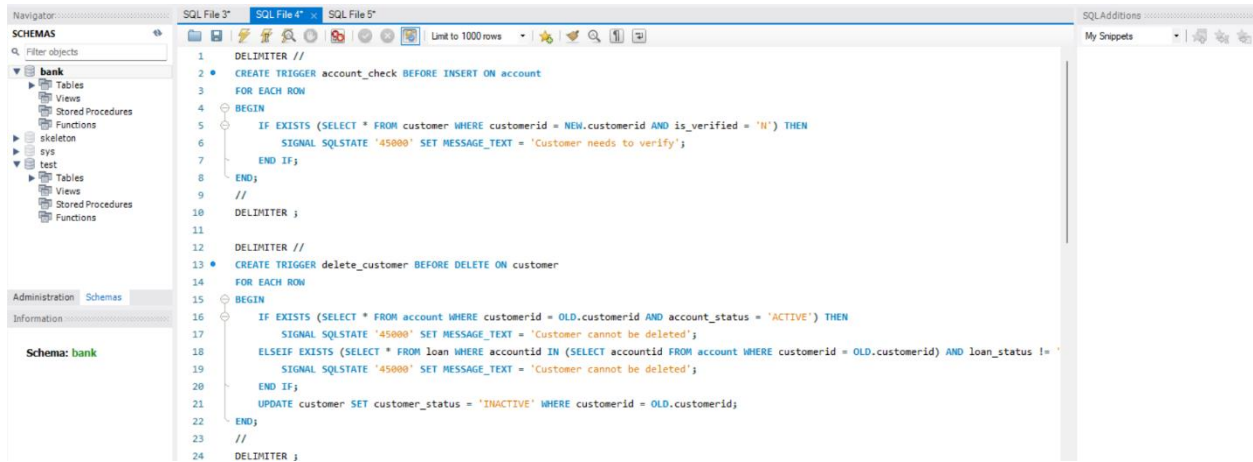


Data Integrity:

These MySQL triggers are designed to enforce various rules in a bank management system:

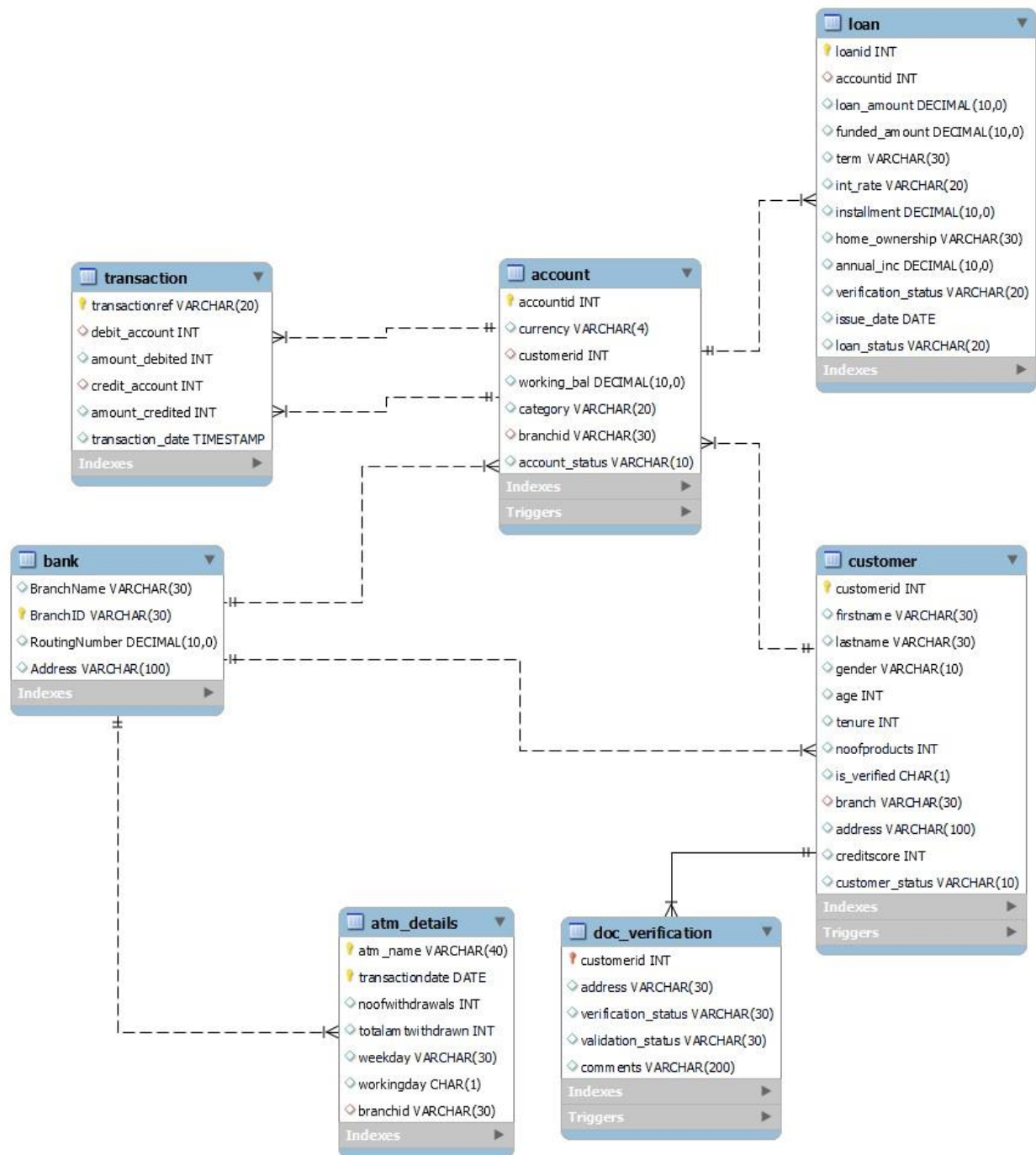
1. ``account_check``: This trigger activates before a new record is inserted into the ``account`` table. It checks if the customer associated with the new account is verified (`is_verified = 'N'`). If the customer is not verified, the trigger prevents the account creation and returns a message stating "Customer needs to verify."`
2. ``delete_customer``: This trigger is invoked before a record is deleted from the ``customer`` table. It checks two conditions: if the customer has any active accounts (`account_status = 'ACTIVE'`) or outstanding loans (loan_status != 'Fully Paid'`). If either condition is met, the trigger prevents the deletion of the customer and returns an appropriate message. Additionally, it sets the `customer_status` to 'INACTIVE' in the `customer` table.`
3. ``update_customer``: Triggered before an update in the ``doc_verification`` table. If the verification status is marked as 'completed', this trigger updates the ``is_verified`` status of the corresponding customer in the ``customer`` table to 'Y'.
4. ``insert_verification``: Activated after a new record is inserted into the ``customer`` table. It automatically inserts a new record into the ``doc_verification`` table with the initial verification status as 'Yet to Start', indicating that the verification process for the new customer is pending.

These triggers enhance data integrity and automate critical aspects of the system's workflow.



```
1 DELIMITER //
2 CREATE TRIGGER account_check BEFORE INSERT ON account
3 FOR EACH ROW
4 BEGIN
5     IF EXISTS (SELECT * FROM customer WHERE customerid = NEW.customerid AND is_verified = 'N') THEN
6         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Customer needs to verify';
7     END IF;
8 END;
9 //
10 DELIMITER ;
11
12 DELIMITER //
13 CREATE TRIGGER delete_customer BEFORE DELETE ON customer
14 FOR EACH ROW
15 BEGIN
16     IF EXISTS (SELECT * FROM account WHERE customerid = OLD.customerid AND account_status = 'ACTIVE') THEN
17         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Customer cannot be deleted';
18     ELSEIF EXISTS (SELECT * FROM loan WHERE accountid IN (SELECT accountid FROM account WHERE customerid = OLD.customerid) AND loan_status != 'PAID') THEN
19         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Customer cannot be deleted';
20     END IF;
21     UPDATE customer SET customer_status = 'INACTIVE' WHERE customerid = OLD.customerid;
22 END;
23 //
24 DELIMITER ;
```

UML Diagram:



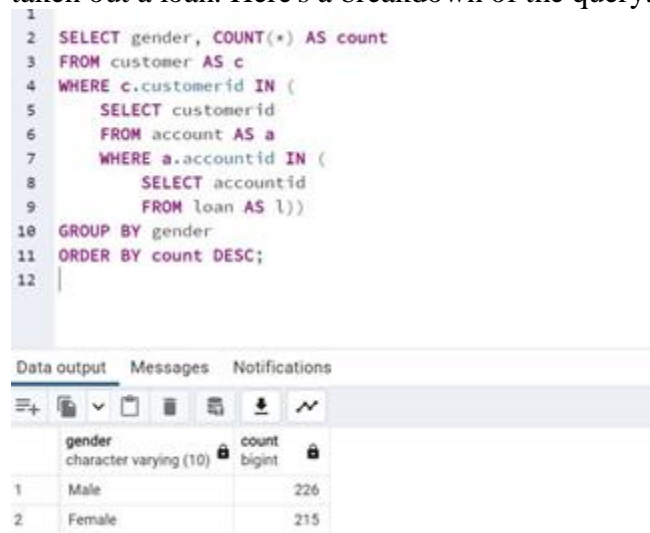
Performance Tuning:

The document outlines a database project for a bank management system, with an emphasis on transaction processing and customer and account information. Important topics including ATM cash withdrawals and customer identity verification are covered. When managing large amounts of client data, the database's ACID properties and effective data retrieval make it superior to Excel. With features tailored specifically for clients and bank staff, the system is intended to improve security and confidence. There are tables for bank information, client details, accounts, document verification, transactions, loans, ATM details, and routing in the comprehensive database schema. The document also describes database query optimization strategies, such as indexing frequently asked columns, using efficient joins, and using particular properties in queries to shorten query execution times.

The performance and user experience of the system are intended to be improved by these tactics.

The first SQL query is designed to count the number of male and female customers who have taken out a loan. Here's a breakdown of the query:

```
1
2 SELECT gender, COUNT(*) AS count
3 FROM customer AS c
4 WHERE c.customerid IN (
5     SELECT customerid
6     FROM account AS a
7     WHERE a.accountid IN (
8         SELECT accountid
9         FROM loan AS l))
10 GROUP BY gender
11 ORDER BY count DESC;
12
```



gender	count
Male	226
Female	215

- SELECT gender, COUNT(*) AS count: This selects two columns, one showing the gender and the other showing the count of rows for each gender.
- FROM customer AS c: This specifies that the data is being selected from the 'customer' table, which is aliased as 'c' for convenience in the query.
- WHERE c.customerid IN (...): This is a nested query that filters the customers based on whether they have an account ID that is linked to a loan.
 - SELECT customerid FROM account AS a WHERE a.accountid IN (...): This nested subquery selects customer IDs from the 'account' table where the account ID is present in the 'loan' table.
 - SELECT accountid FROM loan AS l: This innermost subquery selects account IDs from the 'loan' table.
- GROUP BY gender: This groups the results by gender so that the COUNT function can count the number of rows for each gender.

- ORDER BY count DESC: This orders the results by the count in descending order, so the gender with the most customers appears first.

The below SQL query selects customer IDs from customers with a credit score over 800 who have not had a loan charged off:

```
14 SELECT customer.customerid, count(*) as no_of_accounts
15 FROM customer
16 INNER JOIN account
17 ON customer.customerid = account.customerid
18 group by customer.customerid having count(*) > 2 order by no_of_accounts desc;
19
20
21
22
```

Data output		
Messages		
Notifications		
	customerid [PK] numeric	no_of_accounts bigint
1	191272	23
2	190052	20
3	196895	20
4	111661	19
5	100345	18
6	100410	17

-SELECT customerid FROM customer WHERE creditscore > 800: This selects customer IDs from the 'customer' table where the credit score is greater than 800.

- AND customerid IN (...): This further filters those customers to only include those who have accounts linked to loans that have not been charged off.

- SELECT accountid FROM account WHERE accountid IN (...): This nested subquery selects account IDs from the 'account' table that are also present in the 'loan' table.

- SELECT accountid FROM loan WHERE loan_status != 'Charged Off': This innermost subquery selects account IDs from the 'loan' table where the loan status is not 'Charged Off'.

The belowSQL query is designed to count the number of accounts each customer has and display them in descending order:

```
1 select atm_name, SUM(totalamtwithdrawn) as amount_wintdrawn
2 FROM atm_details group by atm_name order by amount_wintdrawn desc
3
4
```

Data output		
Messages		
Notifications		
	atm_name character varying (40)	amount_wintdrawn bigint
1	Hollywood Avenue AT...	1854299300
2	Kingsley Street ATM	1311309000
3	Main Street ATM	1161463900
4	Genesee Street ATM	999511100
5	Georgia Street ATM	726419500

- SELECT customer.customerid, count(*) as no_of_accounts: This selects the customer ID from the 'customer' table and counts the number of accounts associated with each customer ID, labeling the count as 'no_of_accounts'.

- FROM customer: This specifies the 'customer' table as the main source of data.

- INNER JOIN account ON customer.customerid = account.customerid: This joins the 'customer' table with the 'account' table where the customer IDs match, meaning it only includes customers who have accounts.
- GROUP BY customer.customerid: This groups the results by customer ID so that the COUNT function operates on this grouping.
- HAVING count(*) > 2: This filters the groups to only include those where the customer has more than two accounts.
- ORDER BY no_of_accounts desc: This orders the results by the number of accounts in descending order.

These queries are used to extract and organize information from a database, specifically related to customers, their accounts, and their loans.

Question and answers:

1. Count of Male and Female Customers with Loans

In TRC, the first query might be expressed as:

$$\{ c.\text{gender}, \text{COUNT}(t) \mid c \in \text{Customer} \text{ AND } t \in \text{Loan} \text{ AND } t.\text{accountid} = a.\text{accountid} \text{ AND } a.\text{customerid} = c.\text{customerid} \}$$

This expression states that we want to retrieve a set of tuples with gender and a count where c is a tuple in the Customer relation, t is a tuple in the Loan relation, and a is a tuple in the Account relation, such that the accountid from Loan matches with accountid from Account, and the customerid from Account matches with customerid from Customer.

2. Customer IDs with Credit Score > 800 and Active Loans.

In TRC, the second query might be expressed as:

$$\{ c.\text{customerid} \mid c \in \text{Customer} \text{ AND } c.\text{creditscore} > 800 \text{ AND } \exists a \in \text{Account} (\exists l \in \text{Loan} (l.\text{accountid} = a.\text{accountid} \text{ AND } l.\text{loan_status} \neq \text{'Charged Off'} \text{ AND } a.\text{customerid} = c.\text{customerid})) \}$$

This expression states that we want to retrieve a set of customer IDs where c is a tuple in the Customer relation with a credit score greater than 800, and there exists a tuple a in the Account relation and a tuple l in the Loan relation such that the accountid in Loan is equal to accountid in Account, the loan_status is not 'Charged Off', and the customerid in Account is equal to the customerid in Customer.

3. Customer IDs with Number of Accounts.

In TRC, the third query might be expressed as:

```
{ c.customerid, COUNT(a) | c ∈ Customer AND a ∈ Account AND c.customerid = a.customerid  
GROUP BY c.customerid HAVING COUNT(a) > 2 }
```

This expression states that we want to retrieve a set of tuples with customerid and a count of accounts where c is a tuple in the Customer relation and a is a tuple in the Account relation, such that customerid in Customer matches with customerid in Account. The results are grouped by customerid, and only those groups with more than two accounts are included.

Query optimization:

There are some query optimization techniques which reduce the query run time. They are:

- Instead of using “select” we can use the attributes list which we want to return.
- Using joins will reduce the complexity of the query and makes it simple to run.
- Indexing the columns which are used most of the time in where clause.

Limitations:

The following are the few limitations:

First, scalability issues Performance may suffer when the client base expands if the present database structure is unable to manage a noticeably higher volume of data.

2. Complex Query Execution: Some queries may require a lot of resources to execute, particularly if they involve numerous joins or subqueries.

3. Maintenance Challenges: Keeping the database updated and maintained on a regular basis can be resource-intensive, particularly when handling intricate relationships and dependencies.

4. Problems with Security: Even if the system has security elements, regular attention and changes to security protocols are necessary due to the continuing evolution of cyber threats.

5. Requirement for Precise Data Input: The correctness and dependability of the data entered, which can be harmed by data corruption or human error, determine how effective the system is.

6. Restricted Latitude in Addressing Exceptions: The system may not be able to effectively manage unusual or unexpected situations that don't fit within its preset routines and regulations.

Conclusion:

This small program provides a rich user experience while working with bank-released products such as credit cards, loan schemes (home/education/collateral, et cetera). Any user who has the required permissions can use the application to learn more about his profile details, participate in schemes and products, and seek advantages. Applications are available and accessible at all times. A vetted, trustworthy, and legitimate application.

Easy to get along with and offers a tendency towards friendliness.

Better design integrates practicality and aesthetics.

We have two roles in our database for the two types of users that are present in our system: bank and customers. The bank can access all features and has administrator rights. The complete database is not accessible to the customer. The customer can access his personal information, such as account and profile details, among other things. Due to the login feature, only users with permission can access the database or user interface by using their profile.