Bank Account Management System

Scenario Analysis:

This banking database manages all essential bank operations through connected tables. Customers can open accounts at different branches, with each account having a specific type of savings, checking or both. The system handles daily transactions – deposits, withdrawals, and transfer between accounts – while keeping balances updated.

For loans, customers can apply and make regular payments tracked in the system. Security is built in with username/password verification and login monitoring. Banks can manage their staff across branches and collect customer feedback for service improvement.

The entire system works together, when a customer makes a transaction if updated their balance; when they take a loan, it links to their account; and all activities are properly recorded. The integrated approach ensures everything from basic banking to security and customer service works smoothly while keeping all data accurate and secure.

Category 1: Core Banking Entities

Table	Primary key	Foreign Key
Branches:		
Branch_id	Branch_id	
Branch_name		
Branch_address		
Branch_phone		

Funtions:

Add new branches and assign managers. Update branch details and phone numbers. Find branches by location and view manager information.

Category 1: Core Banking Entities		
Table	**Primary key**	**Foreign Key**
Customer		
Customer_id	Customer_id	
first_name		
last_name		
Email		
Phone		
Address		
Age		

Functions:

Register new customers and verify their IDs. Update customer addresses and phone numbers.
View all customer accounts and personal details.

Category 1: Core Banking Entities		
Table	**Primary key**	**Foreign Key**
Account_types		
account_type_id	account_type_id	
Account_type		
interest_rate		
overdraft_limit		
minimum_balance		

Funtions:

Define savings and checking account rules. Set interest rates and fees. Update account requirements and limits.

Category 2: Financial Operations		
Table	**Primary key**	**Foreign Key**
Account		
Account_id	Account_id	Customer_id
Customer_id		Branch_id
Branch_id		Account_type_id
Account_type_id		
Current_Balance		
Account_open_date		
account_status		

Funtions:

Open new accounts for customers. Check balances and update amounts. Close accounts or change status when needed.

Category 2: Financial Operations		
Table	**Primary key**	**Foreign Key**
Transaction:		
Transaction_id	Transaction_id	account_id
Transaction_type		
current_balance		
account_id		
transaction_reason		
receiver_account		
sender_name		

Transaction_date		
Transaction_amount		

Functions:

Record deposits, withdrawals, and transfers. Check transaction history for any account. Verify transactions are valid and secure.

Category 2: Financial Operations		
Table	**Primary key**	**Foreign Key**
Loans		
Loan_id	Loan_id	customer_id
customer_id		account_id
account_id		
loan_amount		
interest_rate		
loan_term		
Monthly_minimum_payment		
remaining_balance		
loan_issue_date		
loan_end_date		
loan_status		

Functions:

Create new loans with amounts and terms. Record loan payments and update balances. Check which loans are active or paid.

Category 3: Security & Support		
Table	**Primary key**	**Foreign Key**
Customer_credentials		
Customer_id	Customer_id	Customer_id
Username		
password		

Functions:

Set up username and password when customers register for online banking.

Category 3: Security & Support		
Table	**Primary key**	**Foreign Key**
Login_Information		
login_id	login_id	Customer_id
customer_id		
Login_time		
device_information		
login_geo_location		
success_status		

Functions:

Verify customer login credentials. Track login attempts and locations. Change passwords when requested.

Category 3: Security & Support		
Table	**Primary key**	**Foreign Key**
Loan_payment:		
payment_id	payment_id	loan_id
loan_id		
payment_date		
payment_amount		
remaining_amount		
interest_amount		

Functions:

Create new loans with amounts and terms. Record loan payments and update balances. Check which loans are active or paid.

Category 3: Security & Support		
Table	**Primary key**	**Foreign Key**
Branch_employee:		
Employee_id	Employee_id	branch_id
branch_id		
Employee_first_name		
Employee_last_name		
Employee_DOB		
employee_status		
current_position		

Functions:

Add new staff to branches. Update employee positions and status. Assign managers to lead branches.

Category 3: Security & Support		
Table	**Primary key**	**Foreign Key**
Customer_feedback:		
feedback_id	feedback_id	customer_id
customer_id		
Rating		
feedback_type		
submission_date		
follow_up_required		

Functions:

Save customer comments and ratings. Check which feedback needs follow-up. View customer satisfaction scores.

Constraints: -

- A branch must have a valid manager assigned before being activated. Each branch can have multiple employees, but each employee belongs to only one branch.
- A customer can hold multiple accounts (e.g., savings and current), while each account belongs to only one customer.
- Customers can provide multiple pieces of feedback over time, but each feedback record is associated with one customer. Feedback ratings must be within the range of 1 to 5.

- A customer must be 18 years or older to open an account or apply for a loan. Each email, username, and government ID must be unique across all customers. A customer record must exist before any related account, loan, or transaction can be created.
- Every account can have multiple transactions, such as deposits, withdrawals, or transfers. Each transaction is tied to one account. Transaction amounts must always be greater than zero.
- Some accounts may be linked to loans. A customer can have several loans associated with their account. Each loan can have several payments over its term, but every payment belongs to one specific loan.
- The account balance cannot fall below the minimum balance defined by its account type. Account balances cannot become negative after any withdrawal or transfer.
- Loan amounts, interest rates, and monthly payments must be positive values. Loan payments cannot exceed the remaining loan balance.
- A loan cannot be marked “Closed” until the total payments are equal to the full loan amount.
- Login attempts must include the date, time, and status (‘Success’ or ‘Failed’).
- A customer cannot be deleted while having active accounts or unpaid loans.
- An account cannot be deleted if it has existing transactions or active loans.
- Transaction types are limited to ‘Deposit’, ‘Withdrawal’, or ‘Transfer’.
- Loan status is limited to ‘Active’, ‘Closed’, or ‘Defaulted’.
- Account status is limited to ‘Active’, ‘Closed’, or ‘Frozen’.
- All foreign key references must exist before a related record can be added.
- All users login id should be four digits and password should be eight digits long with the mix of symbols, upper case, lower case & numerical digits.

Phase 3:

Table creation code

```
Use Bank_Account_Management_System;


Create table
        Account_type( Account_type_id INT
        Primary Key,
    account_type varchar(50),
    interest_rate decimal(5,2) check(interest_rate >= 0),
    overdraft_limit decimal(10,2) check(overdraft_limit >= 0),
    minimum_balance decimal(10,2) check(minimum_balance >= 0)
);
Create table Branches(
        Branch_id INT PRIMARY KEY,
    Branch_name varchar(50) NOT NULL,
    Branch_address varchar(255),
    Branch_phone varchar(15)
);
Create table Branch_employee(
```

```
        Employee_id INT Primary key,
    Branch_id int,
    Employee_first_name varchar(50),
    Employee_last_name varchar(50),
    Employee_date_of_birth date,
        employee_status VARCHAR(20) check (employee_status in ('Active','Inactive')),
    Current_position varchar(50),
    foreign key (Branch_id) references Branches(Branch_id)
);
Create table Customers(
        Customer_id INT PRIMARY KEY,
    first_name varchar(50),
    last_name varchar(50),
    Email varchar(100) unique,
    phone varchar(15),
    address varchar(255),
    Age int check (Age >= 18)
);

Create table Account(
        Account_id INT Primary Key,
        Customer_id int,
```

```
	Branch_id int,
	Account_type_id int,
	Current_balance decimal(12,2) check(Current_balance >= 0),
	Account_open_date date,
	account_status VARCHAR(20) check (account_status in ('Active','Closed','Frozen')),
	foreign key (Customer_id) references Customers(Customer_id),
	foreign key (Branch_id) references Branches(Branch_id),
	foreign key (Account_type_id) references Account_type(Account_type_id)
);
Create table
	Transactions( Transaction_id INT
	Primary Key,
	transaction_type VARCHAR(20) check (transaction_type in ('Deposit','Withdrawal','Transfer')),
   Current_balance decimal(12,2),
   Account_id int,
   Transaction_reason varchar(255),
   Receiver_account int,
   Sender_name varchar(50),
   Transaction_date datetime,
   Transaction_amount Decimal(12,2) check(Transaction_amount >= 0),
   foreign key (account_id) references Account(Account_id)
);
Create table Loans(
```

```
        Loan_id INT Primary Key,
    Customer_id int,
    Account_id int,
    Loan_amount decimal(12,2) check (Loan_amount >= 0),
    interest_rate decimal(5,2) check (interest_rate >= 0),
    Loan_term int,
    Monthly_minimum_payment decimal(12,2),
    Remaining_balance decimal(12,2),
    Loan_issue_date date,
    Loan_end_date date,
        loan_status VARCHAR(20) check (loan_status in ('Active','Closed','Defaulted')),
    foreign key(Customer_id) references Customers(Customer_id),
    foreign key(Account_id) references Account(Account_id)
);
Create table
        Customers_credentials( Customer_i
        d INT Primary Key, Username
        varchar(50) unique, password
        varchar(50),
        foreign key (Customer_id) references Customers(Customer_id)
);
Create table
        Login_information( Login_id INT
        Primary Key,
```

```
    Customer_id int NOT NULL,
    Login_time datetime,
    Device_information varchar(100),
    Login_geo_location  varchar(100),
    sucess_status varchar(20) check (sucess_status in('Success','Failed')),
    foreign key(Customer_id) references Customers(Customer_id)
);
Create table
        Loan_payment( Payment_id INT
        Primary Key,
    Loan_id int,
    Payment_date datetime,
    Payment_amount decimal(12,5),
    Remaining_amount decimal(12,5),
    Interest_amount decimal(12,5),
    foreign key(Loan_id) references Loans(Loan_id)
);

Create table
        Customer_feedback( Feedback_id
        INT Primary Key,
    Customer_id int,
    Rating int check (Rating between 1 and 5),
    feedback_type varchar(50),
```

```
    submission_date date,
    Follow_up_required boolean,
        foreign key (Customer_id) references Customers(Customer_id)
);
```

INSERT INTO code

```
INSERT INTO Account_type (account_type_id, account_type, interest_rate,
overdraft_limit, minimum_balance) VALUES
(1,'Saving',0.50,0.00,100.00),
(2,'Saving',1.25,0.00,1000.00),
(3,'Checking',0.00,1000.00,0.00),
(4,'Checking',0.10,5000.00,500.00),
(5,'Student',0.75,500.00,50.00),
(6,'Business',0.05,20000.00,2500.00),
(7,'Fixed Deposit',3.50,0.00,5000.00),
(8,'Fixed Deposit',5.00,0.00,10000.00),
(9,'Student',0.40,2000.00,200.00),
(10,'Business',1.75,10000.00,3000.00),
(11,'Business',2.25,20000.00,5000.00),
(12,'Business',0.20,3000.00,0.00);

INSERT INTO Branches(Branch_id, Branch_name, Branch_address, Branch_phone)
VALUES
(101,'Branch1','branch1_address','+1-202-555-0101'),
(102,'Branch2','branch2_address','+1-202-555-0102'),
(103,'Branch3','branch3_address','+1-202-555-0103'),
(104,'Branch4','branch4_address','+1-202-555-0104'),
(105,'Branch5','branch5_address','+1-202-555-0105'),
```

(106,'Branch6','branch6_address','+1-202-555-0106'),

(107,'Branch7','branch7_address','+1-202-555-0107'),

(108,'Branch8','branch8_address','+1-202-555-0108'),

(109,'Branch9','branch9_address','+1-202-555-0109'),

(110,'Branch10','branch10_address','+1-202-555-0110'),

(111,'Branch11','branch11_address','+1-202-555-0111'),

(112,'Branch12','branch12_address','+1-202-555-0112');

INSERT INTO Branch_employee(Employee_id, Branch_id, Employee_first_name, Employee_last_name, Employee_date_of_birth, Employee_status, Current_position) VALUES

(10001,101,'EF1','EL1','1985-03-12','Active','branch Manager'),

(10002,102,'EF2','EL2','1990-07-22','Active','Teller'),

(10003,103,'EF3','EL3','1982-01-19','Inactive','Loan Officer'),

(10004,104,'EF4','EL4','1979-11-05','Active','CSR'),

(10005,105,'EF5','EL5','1992-04-30','Active','Auditor'),

(10006,106,'EF6','EL6','1988-06-15','Active','Teller'),

(10007,107,'EF7','EL7','1995-09-01','Inactive','CSR'),

(10008,108,'EF8','EL8','1986-12-23','Active','Operations Lead'),

(10009,109,'EF9','EL9','1991-10-10','Active','Branch Manager'),

(10010,110,'EF10','EL10','1983-02-02','Active','Loan Specialist'),

(10011,111,'EF11','EL11','1994-08-08','Active','Teller'),

(10012,112,'EF12','EL12','1985-03-12','Inactive','CSR');

INSERT INTO Customers(Customer_id, first_name, last_name, Email, phone, address, Age)VALUES

(1001,'Noah','Adams','noah.adams@example.com','+1-202-555-1001','Tehran',28),

(1002,'Emma','Baker','emma.baker@example.com','+1-202-555-1002','Istanbul',34),

(1003,'Liam','Clark','liam.clark@example.com','+1-202-555-1003','Toronto',41),

(1004,'Olivia','Davis','olivia.davis@example.com','+1-202-555-1004','London',25),

(1005,'Ava','Evans','ava.evans@example.com','+1-202-555-1005','Bandar-e- khamir',38),

(1006,'Mason','Foster','mason.foster@example.com','+1-202-555-1006','Karachi',23),

(1007,'Sophia','Garcia','sophia.garcia@example.com','+1-202-555-1007','Toronto',30),

(1008,'James','Hernandez','james.hernandez@example.com','+1-202-555-1008','Istnabul',45),

(1009,'Isabella','Ivanov','isabella.ivanov@example.com','+1-202-555-1009','Tehran',29),

(1010,'Ethan','Johnson','ethan.johnson@example.com','+1-202-555-1010','London',52),

(1011,'Mia','Kim','mia.kim@example.com','+1-202-555-1011','Tehran',27),

(1012,'Alexander','Lopez','Alex.lopez@example.com','+1-202-555-1012','Istanbul',36);

INSERT INTO Account(Account_id, Customer_id, Branch_id, Account_type_id, Current_balance, Account_open_date, account_status) VALUES

(2001,1001,101,1,1500.00,'2023-01-10','Active'),

(2002,1002,102,2,25000.00,'2022-06-15','Active'),

(2003,1003,103,3,800.50,'2024-02-20','Active'),

(2004,1004,104,4,5200.00,'2025-03-05','Frozen'),

(2005,1005,105,5,950.75,'2023-11-30','Active'),

(2006,1006,106,6,125000.00,'2021-09-12','Active'),

(2007,1007,107,7,30000.00,'2024-07-01','Active'),

(2008,1008,108,8,50000.00,'2022-04-18','Closed'),

(2009,1009,109,9,2800.00,'2025-01-22','Active'),

(2010,1010,110,10,45000.00,'2020-12-01','Active'),

(2011,1011,111,11,98000.00,'2023-05-14','Active'),

(2012,1012,112,12,1200.00,'2025-06-10','Active');

INSERT INTO Transactions(Transaction_id,transaction_type,Current_balance,Account_id,Transaction_reason,Receiver_account,Sender_name,Transaction_date,Transaction_amount)VALUES

(7001,'Deposit',1700.00,2001,'Initial funding',NULL,'Noah Adams','2025-01-05',200.00),

(7002,'Withdrawal',24500.00,2002,'ATM cash',NULL,'Emma Baker','2025-02-10',500.00),

(7003,'Transfer',600.50,2003,'Rent payment',2009,'Liam Clark','2025-03-01',200.00),

(7004,'Deposit',5400.00,2004,'Salary credit',NULL,'Payroll','2025-03-30',200.00),

(7005,'Withdrawal',850.75,2005,'Utility bill',NULL,'Ava Evans','2025-04-12',100.00),

(7006,'Transfer',120000.00,2006,'Supplier payment',2010,'Mason Foster','2025-05-05',5000.00),

(7007,'Deposit',30500.00,2007,'FD interest credit',NULL,'System Interest','2025-06-15',500.00),

(7008,'Transfer',45000.00,2008,'Close C move funds',2011,'James Hernandez','2025-07-01',5000.00),

(7009,'Deposit',3000.00,2009,'Gift',NULL,'Isabella Ivanov','2025-08-03',200.00),

(7010,'Withdrawal',43000.00,2010,'Loan payment',NULL,'Ethan Johnson','2025-09-09',1000.00),

(7011,'Transfer',97000.00,2011,'Investment transfer',2007,'Mia Kim','2025-10-21',1000.00),

(7012,'Deposit',1400.00,2012,'Bonus credit',NULL,'Alexander Lopez','2025-11-02',200.00);

INSERT INTO Loans(Loan_id,Customer_id,Account_id,Loan_amount,interest_rate,Loan_term,Monthly_minimum_payment,Remaining_balance,Loan_issue_date,Loan_end_date,loan_status) VALUES

(3001,1001,2001,5000.00,4.50,24,220.00,3600.00,'2024-01-15','2026-01-15','Active'),

(3002,1002,2002,20000.00,5.25,36,650.00,16000.00,'2023-06-01','2026-06-01','Active'),

(3003,1003,2003,10000.00,6.00,48,250.00,9800.00,'2024-03-20','2028-03-20','Active'),

(3004,1004,2004,8000.00,7.00,18,500.00,6000.00,'2025-04-01','2026-10-01','Active'),

(3005,1005,2005,3000.00,4.00,12,260.00,1200.00,'2023-12-10','2024-12-10','Closed'),

(3006,1006,2006,50000.00,8.50,60,1200.00,48000.00,'2021-10-01','2026-10-01','Active'),

(3007,1007,2007,15000.00,5.75,36,500.00,14500.00,'2024-07-15','2027-07-15','Active'),

(3008,1008,2008,25000.00,6.25,48,700.00,20000.00,'2022-05-01','2026-05-01','Defaulted'),

(3009,1009,2009,6000.00,4.50,24,260.00,5800.00,'2025-02-10','2027-02-10','Active'),

(3010,1010,2010,40000.00,7.00,48,1100.00,39500.00,'2020-12-15','2024-12-15','Closed'),

(3011,1011,2011,70000.00,9.00,72,1300.00,69000.00,'2023-05-20','2029-05-20','Active'),

(3012,1012,2012,9000.00,5.00,24,400.00,8800.00,'2025-06-20','2027-06-20','Active');

INSERT INTO Customers_credentials(Customer_id,Username,password)VALUES

(1001,'noah.adams','Pass@1001'),

(1002,'emma.baker','Pass@1002'),

(1003,'liam.clark','Pass@1003'),

(1004,'olivia.davis','Pass@1004'),

(1005,'ava.evans','Pass@1005'),

(1006,'mason.foster','Pass@1006'),

(1007,'sophia.garcia','Pass@1007'),

(1008,'james.hernandez','Pass@1008'),

(1009,'isabella.ivanov','Pass@1009'),

(1010,'ethan.johnson','Pass@1010'),

(1011,'mia.kim','Pass@1011'),

(1012,'alex.lopez','Pass@1012');

INSERT INTO
Login_information(Login_id,Customer_id,Login_time,Device_information,Login_geo_location,sucess_status)VALUES

(4001,1001,'2025-11-01 08:30:00','iPhone 14/ iOS','Tehran','Success'),

(4002,1002,'2025-11-02 09:45:00','Galaxy S22 / Android','Karachi','Failed'),

(4003,1003,'2025-11-03 12:10:00','Windows 11 PC','Istanbul','Success'),

(4004,1004,'2025-11-04 18:20:00','MacBook Air / macOS','Bandar-e-khamir','Success'),

(4005,1005,'2025-11-05 07:55:00','iPad Pro / iPadOS','Toronto','Success'),

(4006,1006,'2025-11-06 10:15:00','Pixel 8 / Android','Karachi','Failed'),

(4007,1007,'2025-11-07 21:40:00','Windows 10 Laptop','Tehran','Success'),

(4008,1008,'2025-11-08 13:05:00','MacBook Pro / macOS','London','Success'),

(4009,1009,'2025-11-09 16:30:00','iPhone 13 / iOS','Toronto','Success'),

(4010,1010,'2025-11-10 08:10:00','Galaxy S21 / Android','Karachi','Failed'),

(4011,1011,'2025-11-11 11:11:00','Windows 11 Desktop','London','Success'),

(4012,1012,'2025-11-12 19:25:00','Surface Pro / Windows','Istnabul','Success');

INSERT INTO
Loan_payment(Payment_id,Loan_id,Payment_date,Payment_amount,Remaining_amount,Interest_amount)VALUES

(5001,3001,'2025-01-01',220.00000,3580.00000,45.00000),

(5002,3002,'2025-01-10',650.00000,15500.00000,80.00000),

(5003,3003,'2025-02-01',250.00000,9650.00000,60.00000),

(5004,3004,'2025-03-01',500.00000,5800.00000,75.00000),

(5005,3005,'2024-06-01',260.00000,1100.00000,40.00000),

(5006,3006,'2025-05-01',1200.00000,46800.00000,95.00000),

(5007,3007,'2025-06-01',500.00000,14200.00000,70.00000),

(5008,3008,'2025-07-01',700.00000,19800.00000,85.00000),

(5009,3009,'2025-08-01',260.00000,5600.00000,50.00000),

(5010,3010,'2024-11-01',1100.00000,39000.00000,90.00000),

(5011,3011,'2025-09-01',1300.00000,67700.00000,110.00000),

(5012,3012,'2025-10-01',400.00000,8600.00000,55.00000);

INSERT INTO
Customer_feedback(Feedback_id,Customer_id,Rating,feedback_type,submission_date,Follow_up_required)VALUES

(6001,1001,5,'Service','2025-01-02',1),

(6002,1002,4,'Mobile App','2025-02-12',0),

(6003,1003,3,'Fees','2025-03-22',1),

(6004,1004,5,'Loan Support','2025-04-11',0),

(6005,1005,2,'ATM','2025-05-09',1),

(6006,1006,4,'Business Desk','2025-06-30',0),

(6007,1007,5,'FD Services','2025-07-15',0),

(6008,1008,1,'Closure','2025-08-01',1),

(6009,1009,4,'Transfers','2025-09-03',0),

(6010,1010,3,'Support','2025-10-18',1),

(6011,1011,5,'Premium Desk','2025-11-05',0),

(6012,1012,2,'Business Desk','2025-11-18',1);

Procedure 1

1. **Why we used it and what it does:**

This procedure automates the complex logic of transferring money. It ensures that when money moves, it's deducted from sender, added to receiver, and recorded in transaction log, reducing the risk of manual errors.

2. **Query Question:**

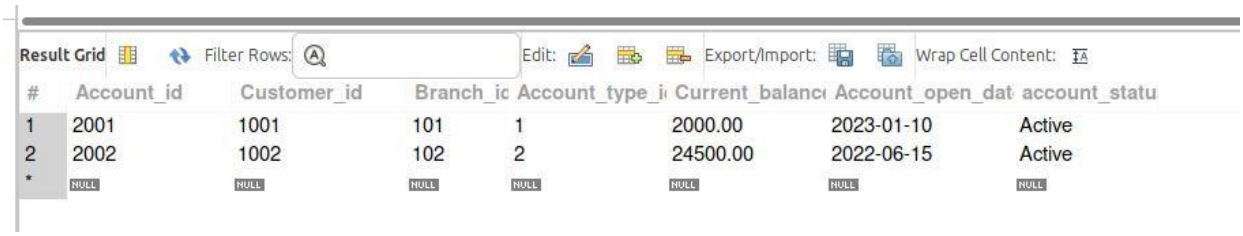
Provide a code that helps transfer the amount of $500 from an account 2002 to another account 2001, using the store procedure.

3. **Code:**

```
DELIMITER //
create PROCEDURE
Process_Fund_Transfer( in sender_acc int,
   in receiver_acc int,
   in amount decimal(12,2)
)
Begin
update Account
set current_balance = current_balance - amount
   where Account_id = sender_acc;
   update Account
set current_balance = current_balance + amount
   where Account_id = receiver_acc;
   insert into Transactions(Transaction_id, transaction_type, current_balance,
```

```
    Account_id, transaction_reason, receiver_Account, sender_name, transaction_date,
transaction_amount)
    values (floor(rand() * 100000), 'Transfer', (select current_balance from Account where
Account_id = sender_acc),
    sender_acc, 'Fund Transfer procedure', receiver_acc, 'System', now(), amount);
END //
DELIMITER ;

call Process_Fund_Transfer(2002,2001,500.00);
select * from Account where Account_id in (2002,2001);
```

4. **Out Come:**



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

#	Account_id	Customer_id	Branch_ic	Account_type_i	Current_balanc	Account_open_dat	account_statu
1	2001	1001	101	1	2000.00	2023-01-10	Active
2	2002	1002	102	2	24500.00	2022-06-15	Active
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Procedure 2

1. **Why we used it and What it does:**

   This streamlines the expansion of the bank. it creates a new branch and immediately assigns a manager to it in the Branch_employee table, ensuring the tule "A Branch must have a valid manager" is easier to follow

2. **Query Question:**

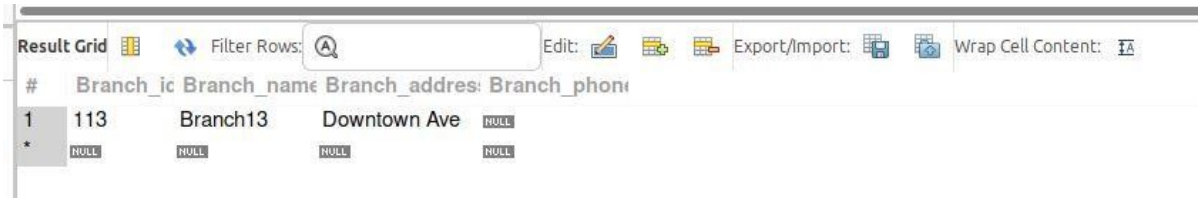
   Add information about new branch “Branch13” and new branch manager.

3. **Code:**

```
DELIMITER //

create PROCEDURE

        Add_New_Branch_With_Manager( in b_id int,
    in b_name varchar(50),
    in b_addr varchar(255),
    in emp_id int,
    in emp_first varchar(50),
    in emp_last varchar(50)

)

Begin

insert into Branches(Branch_id, Branch_name, Branch_address)
    values(b_id, b_name, b_addr);
    insert into Branch_employee(Employee_id, Branch_id, Employee_first_name,
Employee_last_name, employee_status, current_position)
    values(emp_id, b_id, emp_first, emp_last, 'Active', 'Branch Manager');

END //

DELIMITER ;
```

call Add_New_Branch_With_Manager(113, 'Branch13', 'Downtown Ave', 10013, 'John', 'Doe');
select * from Branches Where Branch_id = 113;

4. **Out Come:**



Function 1

1. **Why we used it and What it does:**

This function aggregates data to calculate the total wealth of a specific customer across all their accounts. It helps in identifying VIP customers.

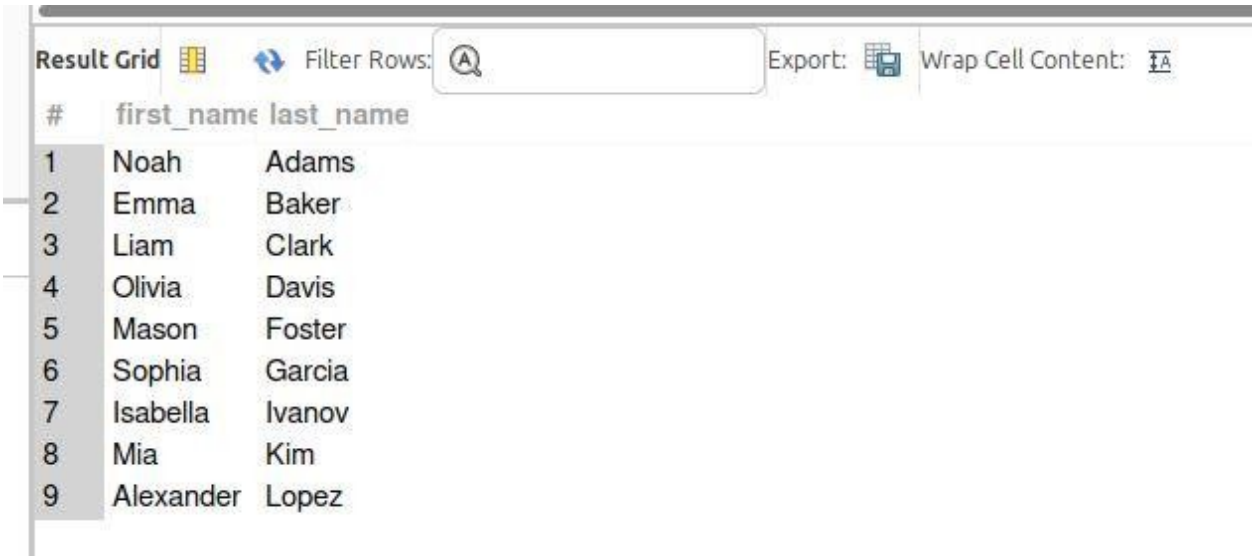
2. **Query Question:**

Use the function to find the total balance of the customer with id 1002.

3. **Code:**

```
DELIMITER //
create FUNCTION calculate_customer_total_balance(cust_id int)
returns decimal(12,2)
deterministic
Begin
Declare total decimal(12,2);
   select sum(Current_balance) into total
   From account
   where Customer_id = cust_id;
   return ifnull(total,0);
END //
DELIMITER ;

SELECT Customer_id, first_name, calculate_customer_total_balance(Customer_id) as
total_wealth
from Customers
where Customer_id = 1002;
```

4. **Out Come**



#	first_name	last_name
1	Noah	Adams
2	Emma	Baker
3	Liam	Clark
4	Olivia	Davis
5	Mason	Foster
6	Sophia	Garcia
7	Isabella	Ivanov
8	Mia	Kim
9	Alexander	Lopez

Function 2

1. **Why we used it and What it does:**

   This function counts how many 'Active' loans a specific customer currently has It is useful for credit checks before approving new services.

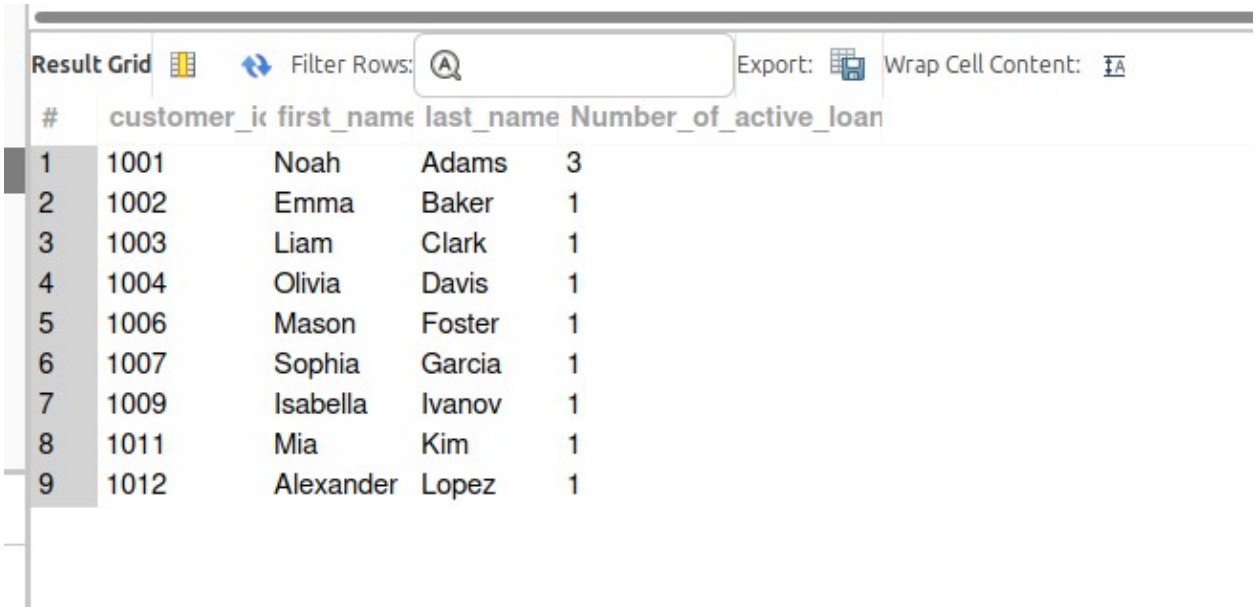
2. **Query Question:**

   List all customers who have active loans using the stored function.

3. **Code:**

```
DELIMITER //
create FUNCTION Count_Customer_Active_loans(cust_id int)
returns int
deterministic
Begin
Declare loan_count int;
   select count(*) into loan_count
   from loans
   Where customer_id = cust_id and loan_status = 'Active';
   return loan_count;
END //
DELIMITER ;

SELECT Customer_id, first_name, last_name, Count_Customer_Active_loans(customer_id) AS Number_of_active_loans
FROM Customers
WHERE Count_Customer_Active_loans(customer_id) > 0;
```

4. **Out Come:**



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	customer_ic	first_name	last_name	Number_of_active_loan
1	1001	Noah	Adams	3
2	1002	Emma	Baker	1
3	1003	Liam	Clark	1
4	1004	Olivia	Davis	1
5	1006	Mason	Foster	1
6	1007	Sophia	Garcia	1
7	1009	Isabella	Ivanov	1
8	1011	Mia	Kim	1
9	1012	Alexander	Lopez	1

View 1

1. **Why we used it and What it does:**

This view simplifies complex reporting by joining the Branches, Account and Branch_employee tables. It provides a snapshot of how many accounts and how much money is held at each branch.

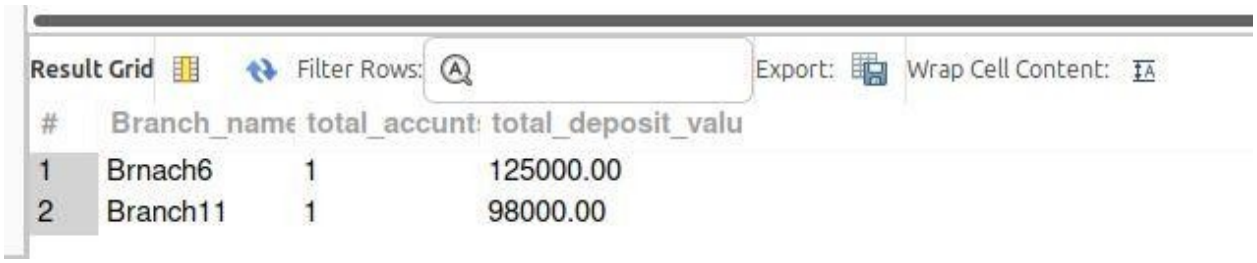
2. **Query Question:**

Use view to find Branches that hold more than $50,000 in total deposits.

3. **Code:**

```
create view Branch_Performance_summary AS
select
b.branch_name,
   count(a.account_id) as total_accunts,
   sum(a.current_balance) as total_deposit_value

from Branches b
join account a on b.branch_id = a.branch_id
group by b.branch_name;

select * from Branch_Performance_summary
where total_deposit_value > 50000;
```

4. **Out Come:**



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	Branch_name	total_accunt	total_deposit_valu
1	Brnach6	1	125000.00
2	Branch11	1	98000.00

View 2

1. **Why we used it and What it does:**

This view joins loans and customers to provide a readable report of who owes money, filtering out closed loans. it hides underlying IDs and shows human-readable names and balances.

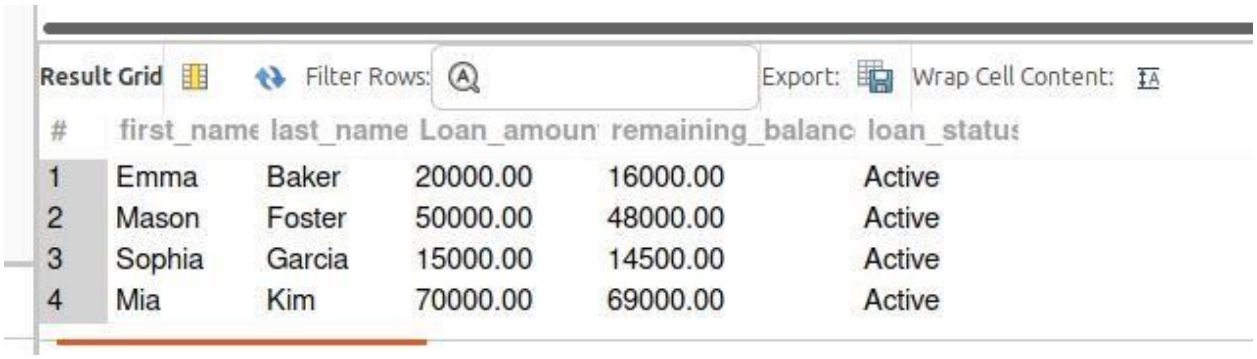
2. **Query Question:**

Select from the view to find all active loans where the remaining balance is greater than $10,000.

3. **Code:**

```
create view loan_status_report as
select
c.first_name,
   c.last_name,
   l.loan_amount,
   l.remaining_balance,
   l.loan_status
from loans l
join customers c on l.customer_id = c.customer_id
where l.loan_status = 'Active';

select * from loan_status_report
where remaining_balance > 10000;
```

4. **Out Come:**



#	first_name	last_name	Loan_amoun	remaining_balanc	loan_status
1	Emma	Baker	20000.00	16000.00	Active
2	Mason	Foster	50000.00	48000.00	Active
3	Sophia	Garcia	15000.00	14500.00	Active
4	Mia	Kim	70000.00	69000.00	Active

Trigger 1

1. **Why we used it and What it does:**

   This trigger ensures data integrity. Whenever a payment is recorded in the loan_payment table, this trigger automatically reduces the remaining_balance in the main loans table, so they never go out of sync.

2. **Query Question:**

   Insert a new payment of $200 for loan 3001 and show that the loans table was updated automatically.

3. **Code:**

```
DELIMITER //
create trigger auto_update_loan_balance
after insert on loan_payment
for each row
Begin
update loans
   set remaining_balance = remaining_balance - new.payment_amount
   where loan_id = new.loan_id;
END //
DELIMITER ;

select remaining_balance from loans where loan_id = 3001;
insert into loan_payment (payment_id , loan_id, payment_date, payment_amount,
remaining_amount, interest_amount)
values(5020, 3001, now(), 200.00, 0, 0);
select remaining_balance from loans where loan_id = 3001;
```

4. **Out Come:**


Result Grid
Filter Rows:
Export:
Wrap Cell Content:
Remaining_balanc
1 3600.00

Trigger 2

1. **Why we used it and What it does:**

This enforces the constraint that account balances cannot become negative. it checks a transaction before it happens; if the withdrawal amount exceeds the balance, it stops the action.

2. **Query Question:**

Attempt to withdrawal $1,000,000 from an account to test that the trigger blocks the invalid transaction.

3. **Code:**

```
DELIMITER //
create trigger Prevent_Low_Balance_Withdrawal
before insert on Transactions
for each row
Begin
declare Current_bal decimal(12,2);
   select Current_balance into Current_bal
   from Account
   where Account_id = new.Account_id;
   if new.Transaction_type = 'Withdrawal' and (Current_bal - new.Transaction_amount) < 0 then
   signal sqlstate '45000'
   set message_text = 'Error: Insufficient funds for this withdrawal.';
   END if;
END //
DELIMITER ;
```

insert into Transactions(Transaction_id, Transaction_type, Account_id, Transaction_amount, Transaction_date)

values(9999,'Withdrawal', 2001, 1000000.00, now());

4. **Out Come**

Transaction 1

1. **Why we used it and What it does:**

Used to ensure atomicity. When a loan is approved, we must create the loan record and deposit the money into the customer's account. If one fails, both must fail to prevent financial discrepancies.

2. **Query Question:**

Execute a transaction to approve a $5000 loan for customer 1001 and deposit it into account 2001.

3. **Code:**

```
Start Transaction;

insert into loans(loan_id, customer_id, account_id, loan_amount, interest_rate,

loan_term, monthly_minimum_payment, remaining_balance, loan_issue_date, loan_status)

values(3020, 1001, 2001, 5000.00, 5, 12, 450.00, 5000.00, now(), 'Active');

update account

set current_balance = current_balance + 5000.00

where account_id = 2001;


insert into transactions(transaction_id, transaction_type, account_id, transaction_amount,
transaction_reason, transaction_date)

values(8001, 'Deposit', 2001, 5000.00, 'Loan disbarsment', now());

Commit;
```

4. **Out Come:**

current_balan...
7000.00

Transaction 2

1. **Why we used it and What it does:**

   Ensures an account is not closed if it has remaining money or debt. it sets the balance to zero and then changes the status to 'Closed' in a signal atomic step.

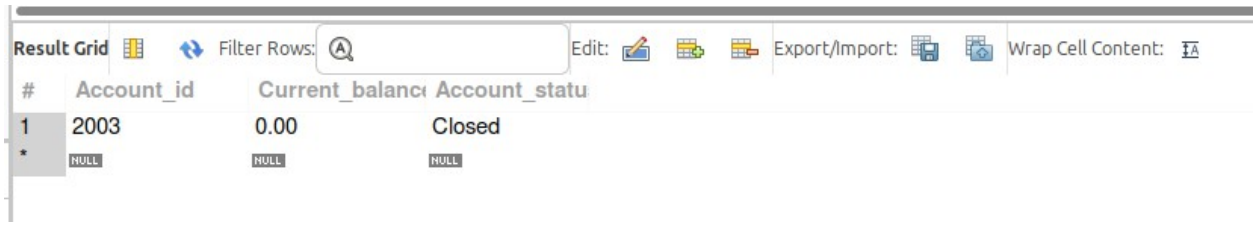
2. **Query Question:**
   Close account 2003 by withdrawing remaining funds and updating status.

3. **Code:**

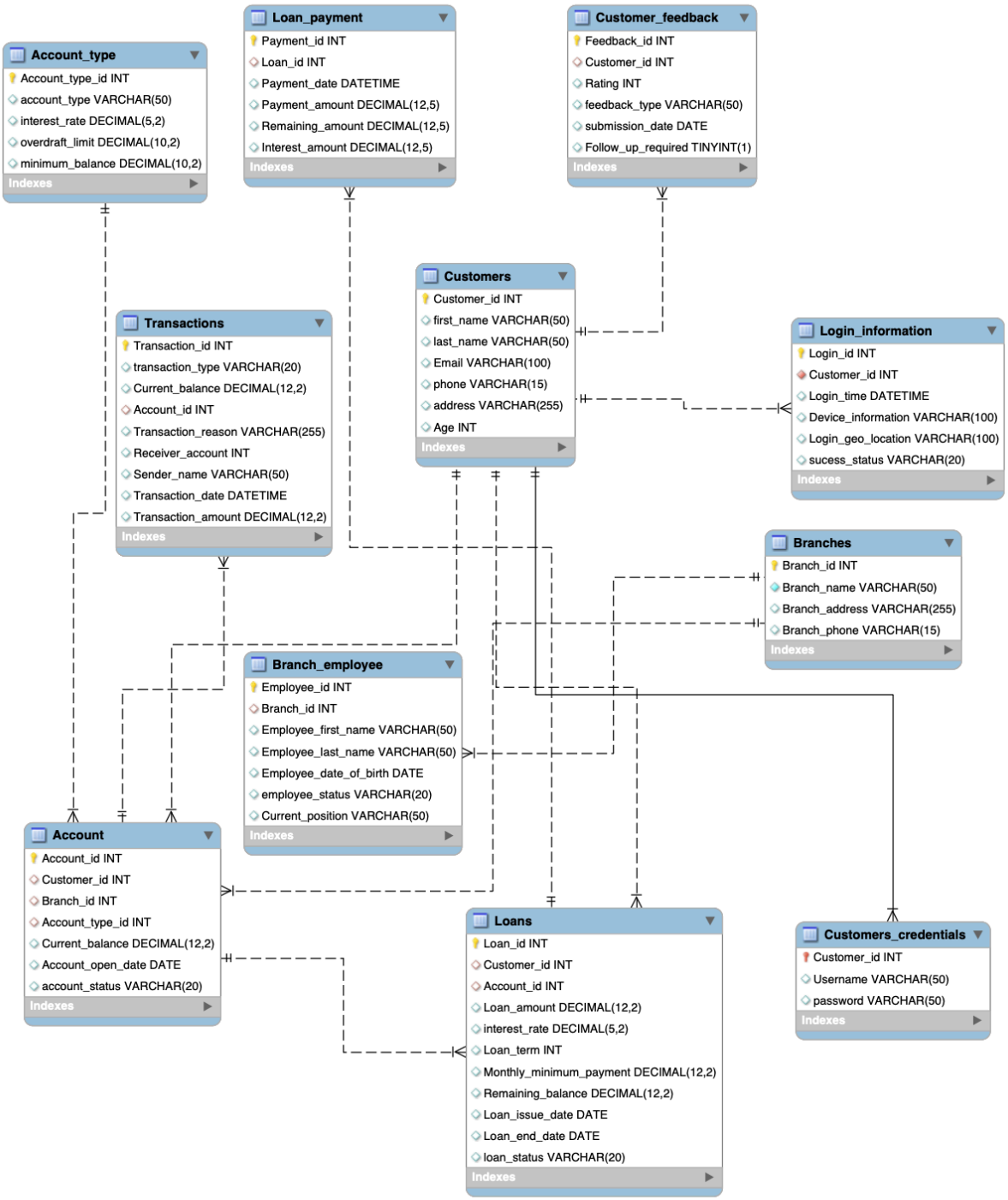
```
START TRANSACTION;

INSERT INTO Transactions(Transaction_id, Transaction_type, Account_id, Transaction_amount, Transaction_reason, Transaction_date)

SELECT 8002, 'Withdrawal', 2003, Current_balance, 'Account closure cashout', now()

FROM Account WHERE Account_id = 2003;


UPDATE Account

SET Current_balance = 0.00

WHERE Account_id = 2003;


UPDATE Account

SET Account_status = 'Closed'

WHERE Account_id = 2003;


COMMIT;
```

```
SELECT Account_id, Current_balance, Account_status
FROM Account
WHERE Account_id = 2003;
```

4. **Out Come:**



ER Diagram before normalization:



Normalization

Normalization is the procedure of organizing data in a database amidst reducing data redundancy, avoidance of anomalies, and maintenance of data integrity.

This system has used normalization up to Third Normal Form, 3NF.

A table is in 1NF if:

Every field has atomic, or indivisible, values
There are no repeating groups or multi-valued attributes
Each record is uniquely identified by a primary key.

Application into the system:
Each table contains a primary key that uniquely identifies each entry, such as customer_id, account_id, and loan_id.
Details like phone numbers, emails, balances, dates are stored as single values.
Data which repeats itself, such as multiple accounts, loans, and transactions go into different tables.

Example:
Accounts and customers are in different tables so that a customer can have more than one account without having to repeat customer details.

A table is in 2NF if:
It is already in 1NF.
Each non-key attribute is fully dependent on the whole primary key.
No partial dependency is allowed.

Application in the system:
Tables use single column primary keys.
All non-key attributes completely depend on their table's primary key.

Example:
The attributes like current_balance, account_status, and account_open_date in the account table depend only on account_id.
Customer information does not come under the Account table, hence there is no partial dependency.

A table is in 3NF if:
It is in 2NF.
There are no transitive dependencies
Non-key attributes depend upon a primary key only and not upon any other non-key attribute.

Application in the system:
Descriptive data are divided into relevant tables.
Cascaded lookup and dependent data are stored separately.

Examples:
The account types store interest rate and minimum balance instead of repeating these in Account.
Branch_employee maintains a list of employee data separate from Branches.
Customer_credentials is separated from Customers because of security and clarity.
Loans and Loan_payment are separated in order to avoid repeating the loan details.

Benefits of Normalization in This System

Data redundancy is eliminated.
It prevents update, insert, and delete anomalies.
Ensures data is always consistent and up to date.
Makes the database scalable, maintainable Clearly defines relationships with foreign keys.
Conclusion - The Bank Management System database is normalized to 3NF. Each entity is representing one concept, the relationships are enforced by foreign keys, and all functional dependencies are correctly maintained. This ensures data integrity, efficiency, and reliability that are important for banking systems.

ER Diagram after normalization