COP 4710 Team Project

Final Report

Sheng Rao, U9568-1295

Quan Nguyen, U9326-4250

Joshua Rivas-Ferreira, U2535-3287

Nathaniel Navarro, U1215-7787

Spencer Durell, U6238-7873

### 

### Organization Overview:

We will be building our database for a banking organization. This organization will serve the banking needs of customers, who will be conducting their transactions at physical bank locations staffed by tellers and bank managers. The main purpose of the database is to track the transactions and current balance of customers’ accounts, with a secondary purpose of tracking the staffing and transaction metrics of individual bank locations. The customer will be able to view their account balance, while the tellers will process transactions to/from/between accounts (deposit/withdraw/transfer). The database will also keep track of management and employee responsibilities.

### Business Rules:

Sources used for Ideas:

<https://www.geeksforgeeks.org/er-diagram-of-bank-management-system/>

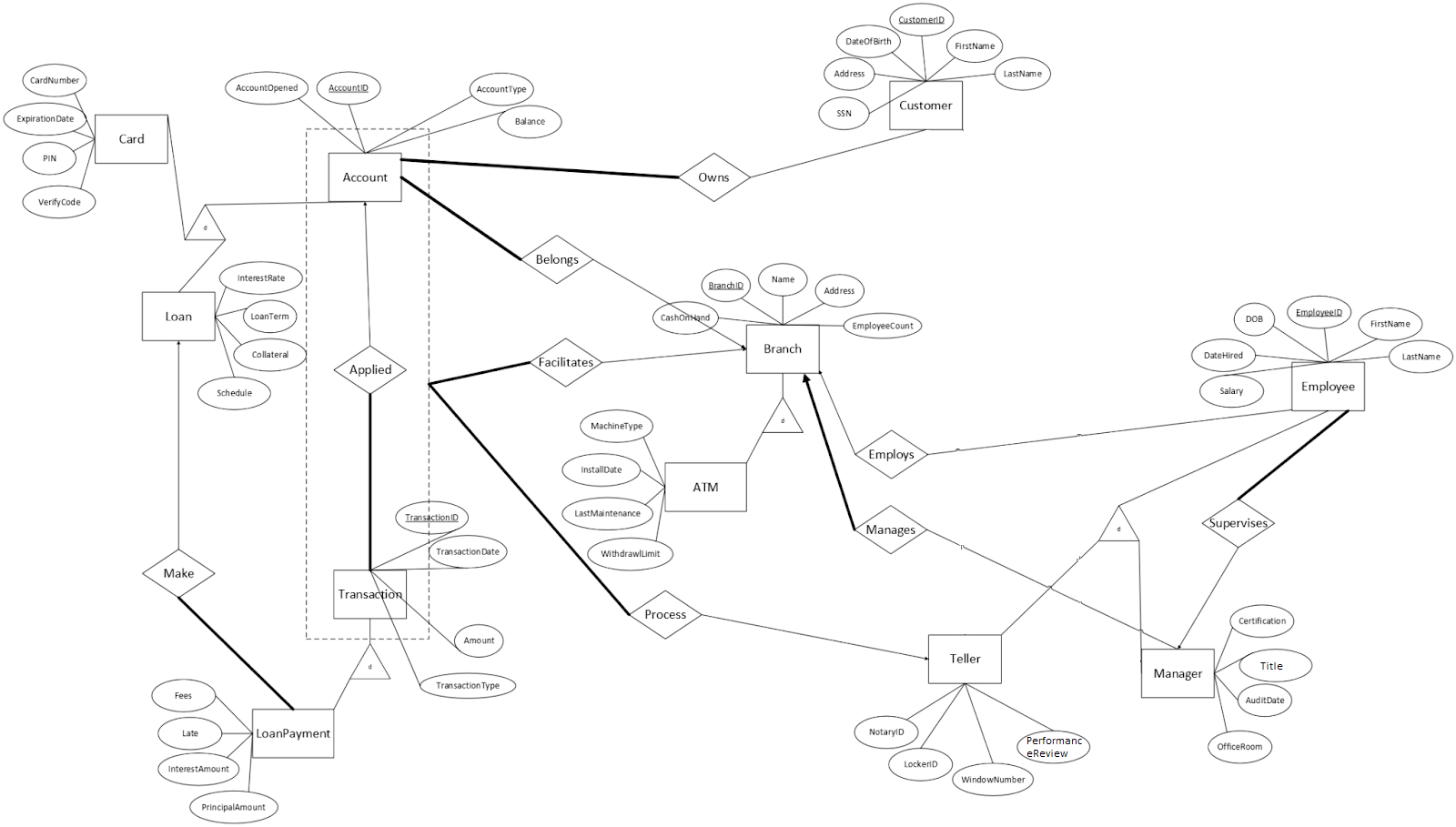
<http://ijeais.org/wp-content/uploads/2024/4/IJAER240404.pdf>

<https://www.scaler.com/topics/er-diagram-for-bank-database/>

***Note: Business rules defining relationships will have entities bolded and underlined and the relationship italicized***

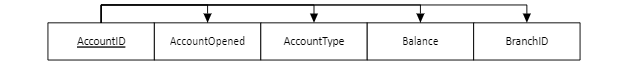
1. An **Account** is an entity with attributes of AccountID (Int), AccountType (String), Balance (Float), AccountOpened (Date)
2. An **Account** can be *owned* by many **Customers** (ex: joint accounts) but must have at least one owner, and a **Customer** can *own* many **Accounts**.
3. An **Account** must *belong* to a **Branch** (typically the place it’s opened at).
4. A **Customer** is an entity with attributes of CustomerID (Int, Key), FirstName (String), LastName (String), DateOfBirth (Date), Address (String), SSN (Int)
5. A **Customer** cannotopen an **Account** without a SSN and valid Address with proof of residence, in other words these cannot be NULL.
6. A **Loan** is a subtype of **Account** and has additional attributes of LoanID (key), InterestRate (Float), LoanTerm (Date) which is the date of the end of the loan, Collateral (String), and Schedule (String) which is the rate of repayment (either monthly, quarterly, biannually, or annually).
7. A **Loan**’s InterestRate must be a positive float value, and its balance must be a negative value (as a calculation of debt)
8. A **Card** is a subtype of **Account** and has additional attributes of CardNumber (Int), ExpirationDate (Date), PIN (Int) which is the 5-digit code you enter for swipe transactions, and VerifyCode (Int) which is the three additional numbers found on the back of the card that you are constantly asked for in online purchases.
9. A **Card**’s ExpirationDate must be a future date. Cards with past/present ExpirationDates are invalid.
10. A **Transaction** is an entity with attributes of TransactionID (Int, Key), TransactionDate (Date), Amount (Float), and TransactionType (String) which can be used to define categories for further processing.
11. Every **Transaction** is *applied* to exactly one **Account**, but an **Account** can have multiple **Transactions**.
12. A **LoanPayment** is a subtype of **Transaction**, with additional attributes of Late (Boolean), Fees (Int) for late payment, InterestAmount (Float), and PrincipalAmount (Float).
13. A **LoanPayment** is *made* towards a **Loan**. A **Loan** can have many **LoanPayments** and each LoanPayment is associated with exactly one Loan.
14. A **Branch** is an entity with attributes of BranchID (Int, Key), Name (String), Address (String), EmployeeCount (Int, Nullable), and CashOnHand (Int) for the amount of physical money in the Branch’s possession which should be kept above a certain limit and is particularly important for the subtype **ATM**.
15. An **ATM** is a subtype of **Branch**, with additional attributes of MachineType (Int), InstallDate (Date), LastMaintenance (Date), and WithdrawalLimit (Int).
16. A Branch can *facilitate* many **Transactions**, and each **Transaction** must be tied to exactly one **Branch**.
17. An **Employee** is an entity with attributes of EmployeeID (Int, Key), FirstName (String), LastName (String), DOB (Date), and Salary (Float).
18. A **Branch** can have many **Employees** but each Employee *works* at one Branch
19. A **Teller** is a subtype of **Employee**, with additional attributes of NotaryID (Int) for the ID of their Notary stamp, LockerID (Int) for the locker that they store their personal effects during work, WindowNumber (Int), and PerformanceReview (Boolean) for whether they are under performance review.
20. A **Teller** can *process* many **Transactions**, and each **Transaction** must be *processed* by exactly one **Teller**.
21. A **Manager** is a subtype of **Employee**, with additional attributes of Title(String), AuditDate (Date) for the date of the last performed Audit at the **Branch** they are managing, OfficeRoom (String) for the room designation of their office, and Certification (String) for the certification that makes them qualified for wealth management (ex: degree or CPA etc cert)..
22. A **Manager** *manages* one **Branch**, and all **Branches** have at least one **Manager**, with some (large) branches having multiple managers.
23. A **Manager** *supervises* many **Employees** but each Employee must be *supervised* by one Manager.

### ER Diagram:

****

### Normalization and DDL statements

**Accounts:**



The table Accounts has columns that all contain singular values and does not have repeating rows or columns. The above diagram does not show any partial or transitive dependencies, therefore the table is in 3NF.

SQL:

CREATE TABLE Accounts(

AccountID INT PRIMARY KEY,

AccountOpened DATE,

AccountType VARCHAR(100),

Balance DECIMAL(9,2) DEFAULT 0.00,

BranchID INT NOT NULL,

CONSTRAINT Belongs FOREIGN KEY (BranchID) REFERENCES Branches(BranchID)

);

**Cards:**

****

The table Cards has columns that all contain singular values and does not have repeating rows or columns. The above diagram does not show any partial or transitive dependencies, therefore the table is in 3NF.

SQL:

CREATE TABLE Cards(

AccountID INT PRIMARY KEY,

CardNumber BIGINT,

ExpirationDate DATE CHECK (ExpirationDate > CURRENT\_DATE),

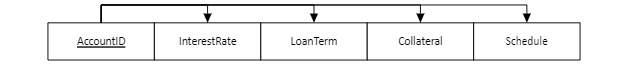
PIN INT,

VerifyCode INT,

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID) ON DELETE CASCADE

);

**Loans:**

****

The table Loans has columns that all contain singular values and does not have repeating rows or columns. The above diagram does not show any partial or transitive dependencies, therefore the table is in 3NF.

SQL:

CREATE TABLE Loans(

AccountID INT PRIMARY KEY,

InterestRate FLOAT CHECK (InterestRate > 0),

LoanTerm DATE,

Collateral VARCHAR(100),

Schedule VARCHAR(100),

Balance DECIMAL(15, 2) CHECK (Balance < 0),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID) ON DELETE CASCADE

);

**Employees:**

| EmployeeID | FirstName | LastName | DOB | Salary | BranchID | ManagerID |
| --- | --- | --- | --- | --- | --- | --- |
| **…** | **…** | **…** | **…** | **…** | **…** | **…** |

****

The above diagram is in 3NF because it has an identified primary key with atomic values, has no repeating groups, and does not have any partial or transitive dependencies.

SQL:

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(100) NOT NULL,

LastName VARCHAR(100) NOT NULL,

DOB DATE NOT NULL,

Salary DECIMAL(9,2) DEFAULT 0.00 NOT NULL,

BranchID INT,

ManagerID INT NOT NULL,

FOREIGN KEY (BranchID) REFERENCES Branches,

FOREIGN KEY (ManagerID) REFERENCES Employees(EmployeeID)

);

**Tellers:**

| EmployeeID | NotaryID | LockerID | WindowNumber | PerformanceReview |
| --- | --- | --- | --- | --- |
| **…** | **…** | **…** | **…** | **…** |

****

The above diagram is in 3NF because it has an identified primary key with atomic values, has no repeating groups, and does not have any partial or transitive dependencies.

SQL:

CREATE TABLE Tellers (

EmployeeID INT PRIMARY KEY,

NotaryID INT,

LockerID SMALLINT,

WindowNumber SMALLINT,

PerformanceReview BOOLEAN,

FOREIGN KEY (EmployeeID) REFERENCES Employees ON DELETE CASCADE

);

**Managers:**

| EmployeeID | Title | AuditDate | OfficeRoom | Certification | BranchID |
| --- | --- | --- | --- | --- | --- |
| **…** | **…** | **…** | **…** | **…** | **…** |

****

The above diagram is in 3NF because it has an identified primary key with atomic values, has no repeating groups, and does not have any partial or transitive dependencies.

SQL:

CREATE TABLE Managers (

EmployeeID INT PRIMARY KEY,

Title VARCHAR(100) NOT NULL,

AuditDate DATE,

OfficeRoom VARCHAR(100),

Certification VARCHAR(100),

BranchID INT NOT NULL,

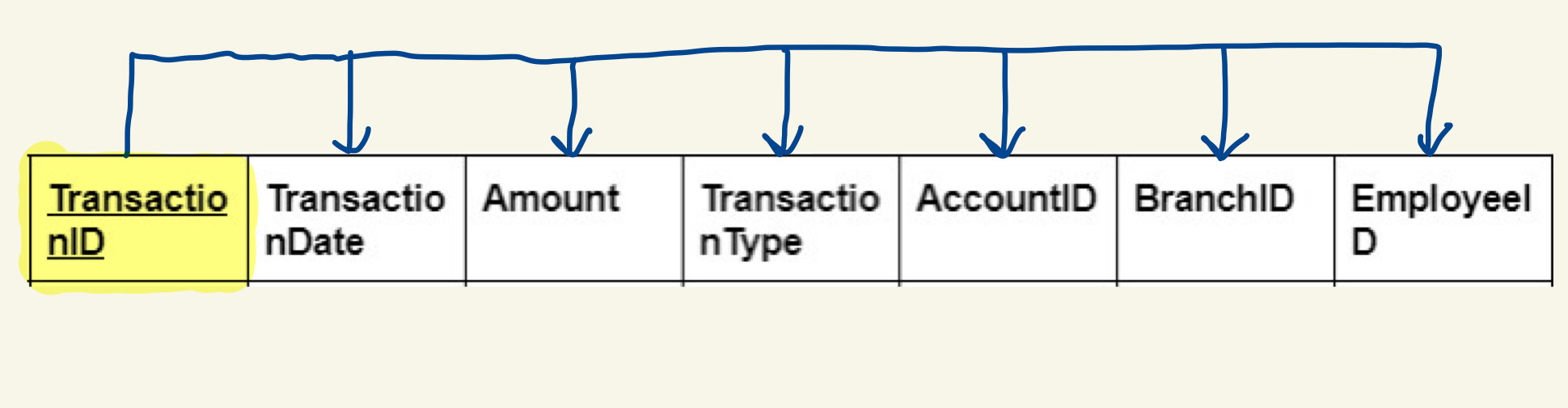
FOREIGN KEY (EmployeeID) REFERENCES Employees ON DELETE CASCADE,

FOREIGN KEY (BranchID) REFERENCES Branches

);

**Transactions:**

| **TransactionID** | **TransactionDate** | **Amount** | **TransactionType** | **AccountID** | **BranchID** | **EmployeeID** |
| --- | --- | --- | --- | --- | --- | --- |
| **…** | **…** | **…** | **…** | **…** | **…** | **…** |

****

In the Transactions table above, a primary key is defined so all rows are unique, values are atomic, and there are no repeating groups so it passes 1NF. According to the dependency diagram, there aren’t any partial or transitive dependencies, so it also passes 2NF and 3NF. As a result, the table is in 3NF.

PostgreSQL:

CREATE TABLE Transactions (

TransactionID SERIAL PRIMARY KEY,

TransactionDate DATE NOT NULL,

Amount DECIMAL(8, 2) NOT NULL CHECK (amount > 0.00 AND amount <= 250000.00),

TransactionType VARCHAR(100) NOT NULL,

AccountID INT NOT NULL,

BranchID INT NOT NULL,

EmployeeID INT NOT NULL,

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID),

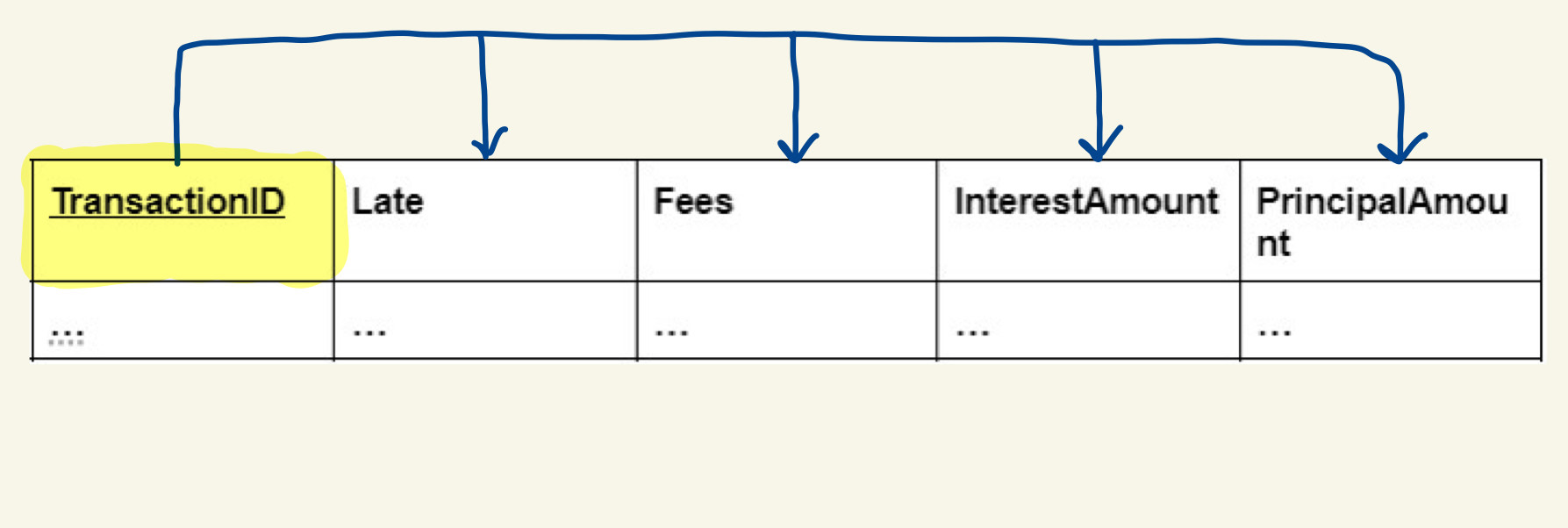
FOREIGN KEY (BranchID) REFERENCES Branches(BranchID),

FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)

);

**Loan Payments:**

| **TransactionID** | **Late** | **Fees** | **InterestAmount** | **PrincipalAmount** |
| --- | --- | --- | --- | --- |
| **…** | **…** | **…** | **…** | **…** |



In the Loan Payments table above, a primary key is defined so all rows are unique, values are atomic, and there are no repeating groups so it passes 1NF. According to the dependency diagram, there aren’t any partial or transitive dependencies, so it also passes 2NF and 3NF. As a result, the table is in 3NF.

PostgreSQL:

CREATE TABLE LoanPayments (

TransactionID INT PRIMARY KEY,

LoanID

Late BOOLEAN NOT NULL,

Fees DECIMAL(7, 2) DEFAULT 0.00,

InterestAmount DECIMAL(7, 2) NOT NULL,

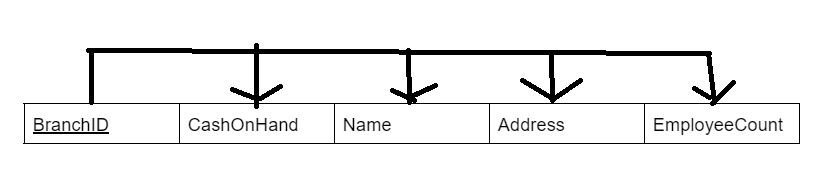
PrincipalAmount DECIMAL(9, 2) NOT NULL,

FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID) ON DELETE CASCADE

);

**Branches:**

| BranchID | CashOnHand | Name | Address | EmployeeCount |
| --- | --- | --- | --- | --- |

****

The Branches table is in 3NF. The primary key is defined meaning all rows are unique, its values are atomic, and there are no repeating groups; therefore it is in 1NF. The dependency diagram also shows that there are no partial or transitive dependencies, meaning it is in 2NF and 3NF. Therefore, the table is at least 3NF.

SQL:

CREATE TABLE Branches(

BranchID INT PRIMARY KEY,

CashOnHand DECIMAL(12,2) DEFAULT 0.00,

BranchName VARCHAR(100),

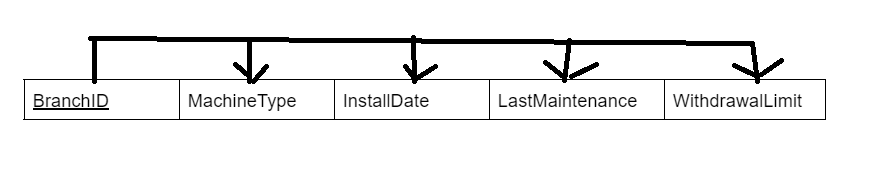
Address VARCHAR(100),

EmployeeCount INT

);

**ATMs:**

| BranchID | MachineType | InstallDate | LastMaintenance | WithdrawalLimit |
| --- | --- | --- | --- | --- |

****

The ATMs table is in 3NF. The primary key is defined meaning all rows are unique, its values are atomic, and there are no repeating groups; therefore it is in 1NF. The dependency diagram also shows that there are no partial or transitive dependencies, meaning it is in 2NF and 3NF. Therefore, the table is at least 3NF.

SQL:

CREATE TABLE ATM(

BranchID INT PRIMARY KEY,

MachineType VARCHAR(100),

InstallDate DATE,

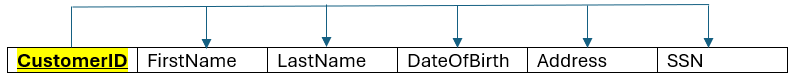
LastMaintenance DATE,

WithdrawalLimit DECIMAL,

FOREIGN KEY (BranchID) REFERENCES Branches(BranchID) ON DELETE CASCADE

);

**Customers:**



In the Customer table above, a primary key is defined so all rows are unique, values are atomic, and there are no repeating groups so it passes 1NF. According to the dependency diagram, there aren’t any partial or transitive dependencies, so it also passes 2NF and 3NF. As a result, the table is at least in 3NF

SQL:

CREATE TABLE customers (

CustomerID SERIAL PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

DateOfBirth DATE NOT NULL,

Address TEXT NOT NULL,

SSN CHAR(11) NOT NULL UNIQUE

);

**Customer Accounts / Owns:**

****

In the Customer Accounts / Owns table above, a primary key is defined so all rows are unique, values are atomic, and there are no repeating groups so it passes 1NF. According to the dependency diagram, there aren’t any partial or transitive dependencies, so it also passes 2NF and 3NF. As a result, the table is at least in 3NF

SQL:

CREATE TABLE Owns (

CustomerID INT,

AccountID INT,

PRIMARY KEY (CustomerID, AccountID),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE,

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID) ON DELETE CASCADE

);

### Data Dictionary:

| **Table Name** | **Attribute Name** | **Contents** | **Type** | **Format** | **Range** | **Required** | **PK OR FK** | **FK REFERENCED TABLE** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Accounts | AccountID | Account ID | INT |  |  | Y | PK |  |
|  | AccountOpened | Opened date | DATE | YYYY-MM-DD |  |  |  |  |
|  | AccountType | Type of account | VARCHAR(100) |  |  |  |  |  |
|  | Balance | Amount in account | DECIMAL(9,2) |  |  |  |  |  |
|  | BranchID | Branch ID | INT |  |  | Y | FK | Branches |
| Cards | AccountID | Account ID | INT |  |  | Y | PK, FK | Accounts |
|  | CardNumber | Number on card | INT |  |  |  |  |  |
|  | ExpirationDate | Expiration on card | DATE | YYYY-MM-DD | > CURRENT\_DATE |  |  |  |
|  | PIN | PIN for card | INT |  |  |  |  |  |
|  | VerifyCode | Security code on card | INT | XXX |  |  |  |  |
| Loans | AccountID | Account ID | INT |  |  | Y | PK, FK | Accounts |
|  | InterestRate | Interest rate on loan | DECIMAL(9,2) | XX.XX | > 0 |  |  |  |
|  | LoanTerm | Date loan ends | DATE | YYYY-MM-DD |  |  |  |  |
|  | Collateral | Item used as collateral | VARCHAR(100) |  |  |  |  |  |
|  | Schedule | How often payments made | VARCHAR(100) |  |  |  |  |  |
|  | Balance | Amount left on loan | DECIMAL(9,2) |  | < 0 |  |  |  |
| Employees | EmployeeID | Identifier of Employee | INT | ######## |  | Y | PK |  |
|  | FirstName | First name | VARCHAR(100) | XXXXXXX.. |  | Y |  |  |
|  | LastName | Last name | VARCHAR(100) | XXXXXXX.. |  | Y |  |  |
|  | DOB | Date of Birth | DATE | YYYY-MM-DD |  | Y |  |  |
|  | Salary | Yearly Salary | DECIMAL(9, 2) | #######.## |  | Y |  |  |
|  | BranchID | Identifier of Branch they work at | INT | ######## |  | N | FK | Branches |
|  | ManagerID | Identifier of the employee’s supervisor | INT | ######## |  | Y | FK | Managers (EmployeeID) |
| Tellers | EmployeeID | Identifier of Teller | INT | ######## |  | Y | PK, FK | Employees |
|  | NotaryID | Identifier of their unique Notary Stamp | INT | ######## |  | N |  |  |
|  | LockerID | Identifier of their assigned locker for personal effects | SMALLINT | ## |  | N |  |  |
|  | WindowNumber | Identifier of the Window they work at | SMALLINT | ## |  | N |  |  |
|  | PerformanceReview | True/False whether they are under performance review | BOOLEAN | True/False |  | N |  |  |
| Managers | EmployeeID | Identifier of Manager (Employee) | INT | ###### |  | Y | PK, FK | Employees |
|  | Title | Specific Job Title | VARCHAR(100) | XXXXXXX.. |  | Y |  |  |
|  | AuditDate | Date of last performed internal audit | DATE | YYYY-MM-DD |  | N |  |  |
|  | OfficeRoom | Designation of room of their office | VARCHAR(100) | XXXXXXX.. |  | N |  |  |
|  | Certification | Qualification used for their wealth management authority (degree, CPA cert, etc) | VARCHAR(100) | XXXXXXX.. |  | N |  |  |
|  | BranchID | Identifyer of the Branch that they manage | INT | ###### |  | Y | FK | Branches |
| Transactions | TransactionID | Transaction ID | INT | ###### | NA | Y | PK |  |
|  | TransactionDate | Date of transaction | DATE | YYYY-MM-DD | NA | Y |  |  |
|  | Amount | Amount of money transferred | DECIMAL(8, 2) | ######.## | 0.01-250000.00 | Y |  |  |
|  | TransactionType | Type of transaction | VARCHAR(100) | XXXXXXX… | NA | Y |  |  |
|  | AccountID | Account ID | INT | ######## | NA | Y | FK | Accounts |
|  | BranchID | Branch ID | INT | ### | NA | Y | FK | Branches |
|  | EmployeeID | Employee ID | INT | ###### | NA | Y | FK | Employees |
| Loan Payments | TransactionID | Transaction ID | INT | ### | NA | Y | PK, FK | Transactions |
|  | Late | Indicates whether loan is overdue | BOOLEAN | TRUE/FALSE | NA | Y |  |  |
|  | Fees | Fees associated with payment | DECIMAL(7, 2) | #####.## | NA | N |  |  |
|  | InterestAmount | Amount of interest charged | DECIMAL(7, 2) | #####.## | NA | Y |  |  |
|  | PrincipalAmount | Amount of payment that reduces the loan’s principal | DECIMAL(9, 2) | #######.## | NA | Y |  |  |
| Branches | BranchID | Branch ID | INT |  |  | Y | PK |  |
|  | CashOnHand | Amount of Cash stored at the bank currently | DECIMAL(11,2) | ###.## |  | Y |  |  |
|  | Name | Name of Branch | VARCHAR(100) |  |  | Y |  |  |
|  | Address | Address of Branch | VARCHAR(100) |  |  | Y |  |  |
|  | EmployeeCount | Numbers of Employees Employed at a Branch | INT |  |  |  |  |  |
| ATMs | BranchID | Branch ID | INT |  |  | Y | PK, FK |  |
|  | MachineType | Type of Machine | VARCHAR(100) |  |  | Y |  |  |
|  | InstallDate | When ATM was Installed | DATE | YYYY-MM-DD |  | Y |  |  |
|  | LastMaintenance | When ATM last had Maintenance | DATE | YYYY-MM-DD |  | Y |  |  |
|  | WithdrawalLimit | Max Cash able to be withdrawn | DECIMAL(9,2) | ####.## |  | Y |  |  |
| Customers | CustomerID | Identifier of Customer | INT |  |  | Y | PK |  |
|  | FirstName | Customer’s first name | VARCHAR(100) |  |  | Y |  |  |
|  | LastName | Customer’s last name | VARCHAR(100) |  |  | Y |  |  |
|  | DateOfBirth | Customer’s legal date of birth | DATE | YYYY-MM-DD |  | Y |  |  |
|  | Address | Customer’s address | TEXT |  |  | Y |  |  |
|  | SSN | Social Security Number | DECIMAL(9,0) | XXX-XX-XXXX |  | Y |  |  |
| Customer Accounts | CustomerID | Identifier of the Customer owning the Account | INT |  |  | Y | PK, FK | Customers (CustomerID) |
|  | AccountID | Identifier of the Account being owned by the Customer | INT |  |  | Y | PK, FK | Accounts (AccountID) |

### Table Entries

| Name | Number of Entries |
| --- | --- |
| Branches | 20 |
| ATM | 10 |
| Employees | 20 |
| Managers | 10 |
| Tellers | 10 |
| Accounts | 20 |
| Cards | 10 |
| Loans | 10 |
| Transactions | 21 |
| LoanPayments | 10 |
| Customers | 10 |
| Owns | 20 |

### Interesting Updates & Results

UPDATE LoanPayments

SET Fees = Fees \* 1.20

WHERE Late = TRUE AND PrincipalAmount > 1000.00;

This update is interesting because it compounds late fees by 20%, a simple and efficient way to implement automated late fee compounding on all accounts with a single instruction. More specific qualifiers can be added to the WHERE conditional if necessary.

UPDATE Transactions

SET Amount = Amount + 100.00

WHERE AccountID = 1 AND TransactionType = 'Deposit';

This update is interesting because it adds 100 to all deposit transactions for accountID 1, which seems like a fix for some bug or mistake that caused all records from this account to underreport by 100.

UPDATE Transactions

SET Amount = Amount \* 2.00

WHERE BranchID = 1 AND AccountID < 11;

This update is interesting because it seems like evidence of embezzlement. A bad actor is perhaps using the intersection of branchID and low accountID (which typically is given to management) to double their money without using specific names or accountIDs.

UPDATE Transactions

SET Amount = 5000.00

WHERE TransactionType = 'Withdrawal' AND Amount > 5000.00;

This update is interesting because it’s a blanket operation that works on all transactions that are withdrawals and of an amount larger than 5000. The result is that all withdrawal transactions over 5000 are capped to 5000 instead.

### Constraints Check Results

-- Integrity

INSERT INTO Branches (BranchID, CashOnHand, BranchName, Address, EmployeeCount) VALUES

(1, 30000.00, 'Dupe', 'Dupe', 10);

This query results in an error: ERROR: Key (branchid)=(1) already exists.duplicate key value violates unique constraint "branches\_pkey", since you can’t have two entries with the same primary key.

-- Check

UPDATE Cards

SET expirationdate = '2023-06-27'

WHERE accountid = 1;

This query results in an error: ERROR: Failing row contains (1, 5579747144710460, 2023-06-27, 1992, 505).new row for relation "cards" violates check constraint "cards\_expirationdate\_check", since the expiration date is in the past instead of the future.

UPDATE Loans

SET balance = 100

WHERE accountid = 11;

This query results in an error: ERROR: Failing row contains (11, 34.01, 2026-03-26, Vehicles, Biweekly, 100.00).new row for relation "loans" violates check constraint "loans\_balance\_check", because loan balances have to be negative, not positive.

UPDATE Loans

SET interestrate = -10.00

WHERE accountid = 11;

This query results in an error: ERROR: Failing row contains (11, -10, 2026-03-26, Vehicles, Biweekly, -1500.00).new row for relation "loans" violates check constraint "loans\_interestrate\_check", because interest rates have to be positive, not negative.

UPDATE Transactions

SET Amount = 500000.00

WHERE TransactionID = 15;

This query results in an error: ERROR: Failing row contains (15, 2025-10-30, 500000.00, Deposit, 4, 10, 9010).new row for relation "transactions" violates check constraint "transactions\_amount\_check", since amounts have to be below 250,000 which 500,000 isn’t.

-- Referential

INSERT INTO ATM (BranchID, MachineType, InstallDate, LastMaintenance, WithdrawalLimit) VALUES

(99, 'BadFK', '1990-01-01', '1990-01-01', 5000.00);

This query results in an error: ERROR: Key (branchid)=(99) is not present in table "branches".insert or update on table "atm" violates foreign key constraint "atm\_branchid\_fkey", because the FK being referenced doesn’t exist in the Branches table.

INSERT INTO Owns (CustomerID, AccountID)

VALUES (999, 1);

This query results in an error: ERROR: Key (customerid)=(999) is not present in table "customers".insert or update on table "owns" violates foreign key constraint "owns\_customerid\_fkey", because the FK being referenced (999) doesn’t exist in the Customers table.

INSERT INTO Owns (CustomerID, AccountID)

VALUES (101, 999);

This query results in an error: ERROR: Key (accountid)=(999) is not present in table "accounts".insert or update on table "owns" violates foreign key constraint "owns\_accountid\_fkey", because the FK being referenced (999) doesn’t exist in the Accounts table.

-- Not Null

INSERT INTO Branches (BranchID, CashOnHand, BranchName, Address, EmployeeCount) VALUES

(NULL, 25000.00, 'Error', 'Error', 10);

This query results in an error: ERROR: Failing row contains (null, 25000.00, Error, Error, 10).null value in column "branchid" of relation "branches" violates not-null constraint, because the branchID cannot be NULL by constraint as well as regular dbms format.

INSERT INTO Customers (CustomerID, FirstName, LastName, DateOfBirth, SSN, Address)

VALUES (111, NULL, 'NullName', '1990-01-01', '111-22-3333', '789 Willow St, Lincoln, IL');

This query results in an error: ERROR: Failing row contains (111, null, NullName, 1990-01-01, 789 Willow St, Lincoln, IL, 111-22-3333).null value in column "firstname" of relation "customers" violates not-null constraint, because that constraint requires that column to be not null.

INSERT INTO LoanPayments (TransactionID, Late, Fees, InterestAmount, PrincipalAmount)

VALUES

(1, FALSE, 100.00, 29.32, NULL);

This query results in an error: ERROR: Failing row contains (1, f, 100.00, 29.32, null).null value in column "principalamount" of relation "loanpayments" violates not-null constraint, because the principalamount being inserted is null, which violates the constraint.

### Queries & Results

-- (Aggregation) Find the amount of accounts associated with each branch. List them in descending order and display total balance across all accounts

SELECT branches.branchname,

COUNT(\*) AS accounts,

SUM(accounts.balance) AS balance\_sum

FROM Accounts

JOIN branches on accounts.branchid = branches.branchid

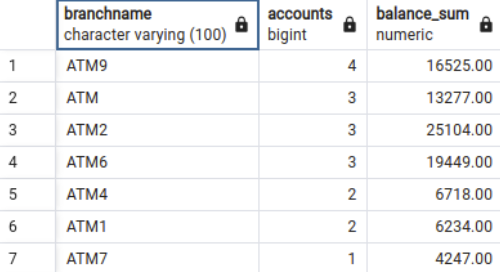
GROUP BY

branches.branchname

ORDER BY

accounts DESC;

This query results in the table:



Plus more rows that are too much for the window to display.

-- (Aggregation, Subquery) Find branches that have average balances higher than the average branch of overall accounts

SELECT branches.branchname,

AVG(balance) AS average\_balance

FROM Accounts

JOIN branches on accounts.branchid = branches.branchid

WHERE

accounts.balance > (SELECT AVG(accounts.balance) FROM accounts)

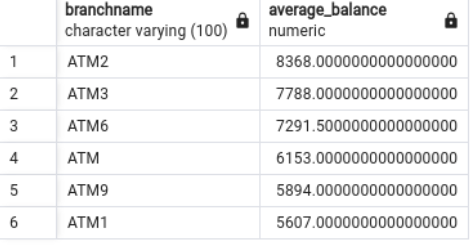
GROUP BY

branches.branchname

ORDER BY

average\_balance DESC;

This query results in the table:



-- (Aggregation, Subquery) We want to find the average salary of tellers at each branch (including branches with no tellers), ordered from highest to lowest.

SELECT

b.BranchName,

ROUND(AVG(ts.Salary),2) AS av

FROM Branches AS b LEFT JOIN (

SELECT

e.EmployeeID,

e.BranchID,

e.Salary

FROM Employees AS e JOIN Tellers AS t

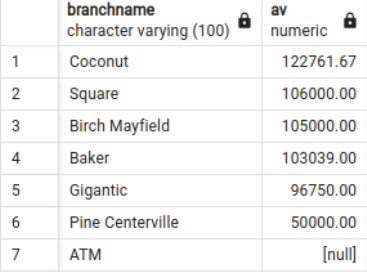
ON e.EmployeeID = t.EmployeeID

) AS ts ON b.BranchID = ts.BranchID

GROUP BY b.BranchName

ORDER BY av DESC NULLS LAST;

This query results in the table:



-- We want to find the names of Managers born before the year 1964, ordered by their DOB in descending order.

SELECT

FirstName,

LastName,

DOB

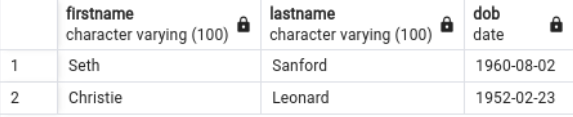
FROM Employees AS e JOIN Managers AS m

ON e.EmployeeID = m.EmployeeID

WHERE DOB < '1964-01-01'

ORDER BY DOB DESC;

This query results in the table:



-- (Subquery) Get total withdrawals from ATMs that have been used this year

SELECT Branches.BranchName, Branches.Address, (

SELECT SUM(Transactions.Amount)

FROM Transactions

WHERE Transactions.BranchID = Branches.BranchID

AND Transactions.TransactionType = 'Withdrawal'

AND Transactions.TransactionDate >= '2024-01-01'

) AS TotalWithdrawals

FROM Branches

JOIN ATM ON Branches.BranchID = ATM.BranchID

WHERE ATM.BranchID IN (

SELECT DISTINCT Transactions.BranchID

FROM Transactions

WHERE Transactions.TransactionType = 'Withdrawal'

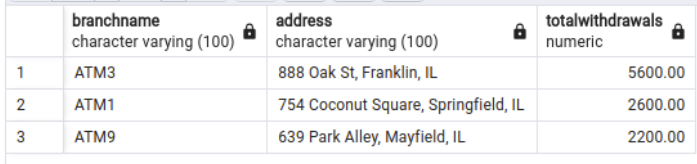
AND Transactions.TransactionDate >= '2024-01-01'

AND Transactions.Amount > 500

)

ORDER BY TotalWithdrawals DESC;

This query results in the table:



-- (Aggregation) Get Addresses with no ATM installed, just a Bank

SELECT Branches.Address

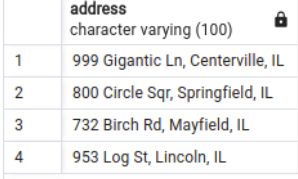
FROM Branches

LEFT JOIN ATM ON ATM.BranchID = Branches.BranchID

GROUP BY Branches.Address

HAVING COUNT(ATM.BranchID) = 0;

This query results in the table:



-- (Aggregation) Get the monthly transaction count and total amount for each branch

SELECT

B.BranchID,

B.BranchName,

date\_trunc('month', T.TransactionDate)::DATE AS Month,

COUNT(T.TransactionID) AS TransactionCount,

SUM(T.Amount) AS TotalTransactionAmount

FROM

Transactions T

JOIN

Branches B ON T.BranchID = B.BranchID

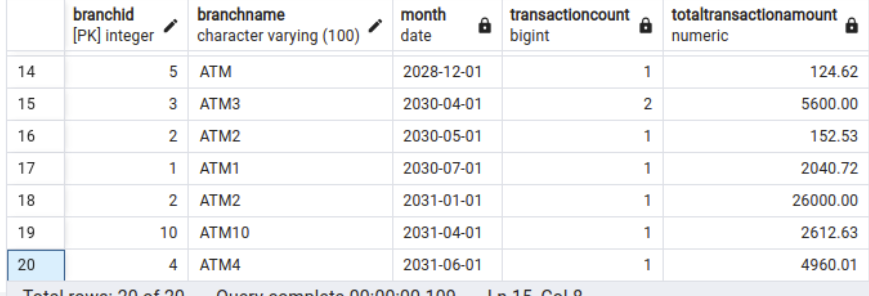
GROUP BY

B.BranchID, Month

ORDER BY

Month, B.BranchID;

This query results in the table:



It’s pretty basic since the transactions table is pretty sparse spread over the large time frame, but you can see it’s aggregating correctly on 04-2030

-- (3+, Aggregation) Get the remaining balance left on each loan, along with associated customer info, and list them in descending order

SELECT

C.FirstName,

C.LastName,

C.SSN,

L.AccountID,

L.Balance,

SUM(LP.PrincipalAmount + LP.InterestAmount) AS TotalLoanPayment,

(SUM(LP.PrincipalAmount + LP.InterestAmount) + L.Balance) AS RemainingBalance

FROM

LoanPayments LP

JOIN

Loans L ON LP.TransactionID = L.AccountID

JOIN

Owns O ON L.AccountID = O.AccountID

JOIN

Customers C ON O.CustomerID = C.CustomerID

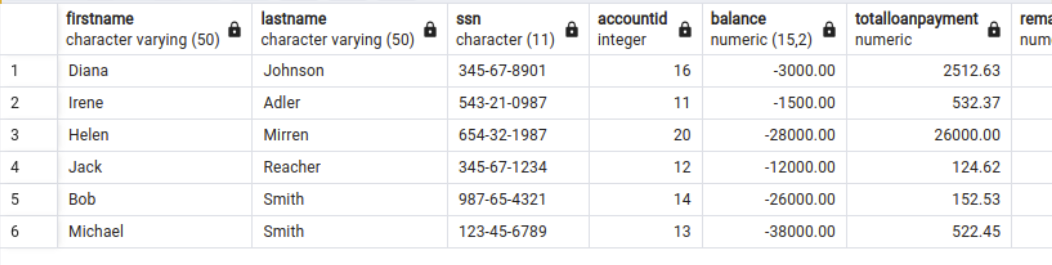
GROUP BY

L.AccountID, L.Balance, C.FirstName, C.LastName, C.SSN

ORDER BY

RemainingBalance DESC;

This query results in the table:



-- (Subquery) Find customers with more than one account

SELECT

C.FirstName,

C.LastName,

COUNT(o.AccountID) AS NumberOfAccounts

FROM Customers C

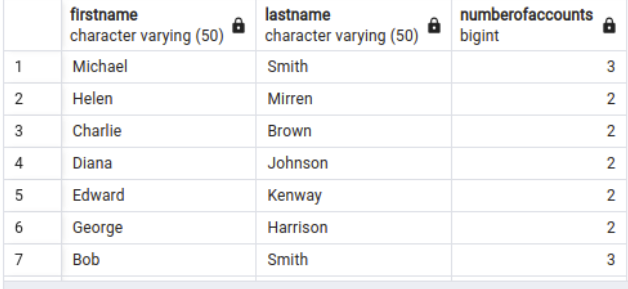
JOIN

Owns O ON C.CustomerID = O.CustomerID

GROUP BY C.CustomerID

HAVING COUNT(O.AccountID) > 1;

This query results in the table:



-- (3+) Get customers with account balances and account types

SELECT

C.FirstName,

C.LastName,

A.AccountType,

A.Balance

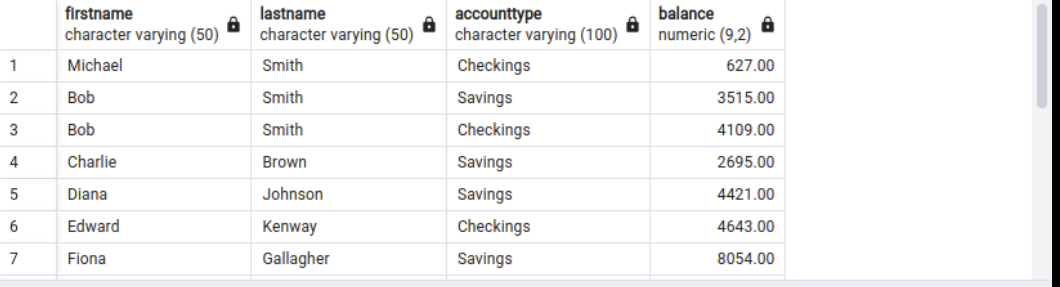
FROM Customers C

JOIN

Owns O ON C.CustomerID = O.CustomerID

JOIN

Accounts A ON O.AccountID = A.AccountID;

This query results in the table:

With more rows not displayed in the window but existing downward.