Online Consumer Banking DB

Team 6: Jiashu Chen, Wenjing Li, Qianqian Liu, Irene Yang, Chesie Yu

Table of contents

01

DB Design DOC

Database Purpose, Business Problems,
Business Rules

Database Purpose

The purpose of this database is to...

- Support efficient **storage**, **update**, and **retrieval** of consumer banking accounts, transaction data, and branch information
- Enable banks to **manage** and **track** the users' or clients' account activities by providing real-time access to accurate and secure information

And the most important thing: aiming to facilitate daily banking activities for consumer banking

Business Problems

The business problems we want to address are...

- Facilitate daily banking activities including account opening and transaction recording (i.e., transfer in, transfer out, withdrawal, or saving)
- Streamline account management and tracking process/transaction for customers

The benefits of database

Access
Access account information such as account number and balances

Retrieve
Retrieve transaction history and detailed transaction information

Generate
Generate detailed financial statements on transaction history, spending & budgeting, etc.

Track
Track branches' transaction data for different locations

Alert
Set up alerts with regard to low balance, overdraft, or transaction limits, etc.

Business Rules

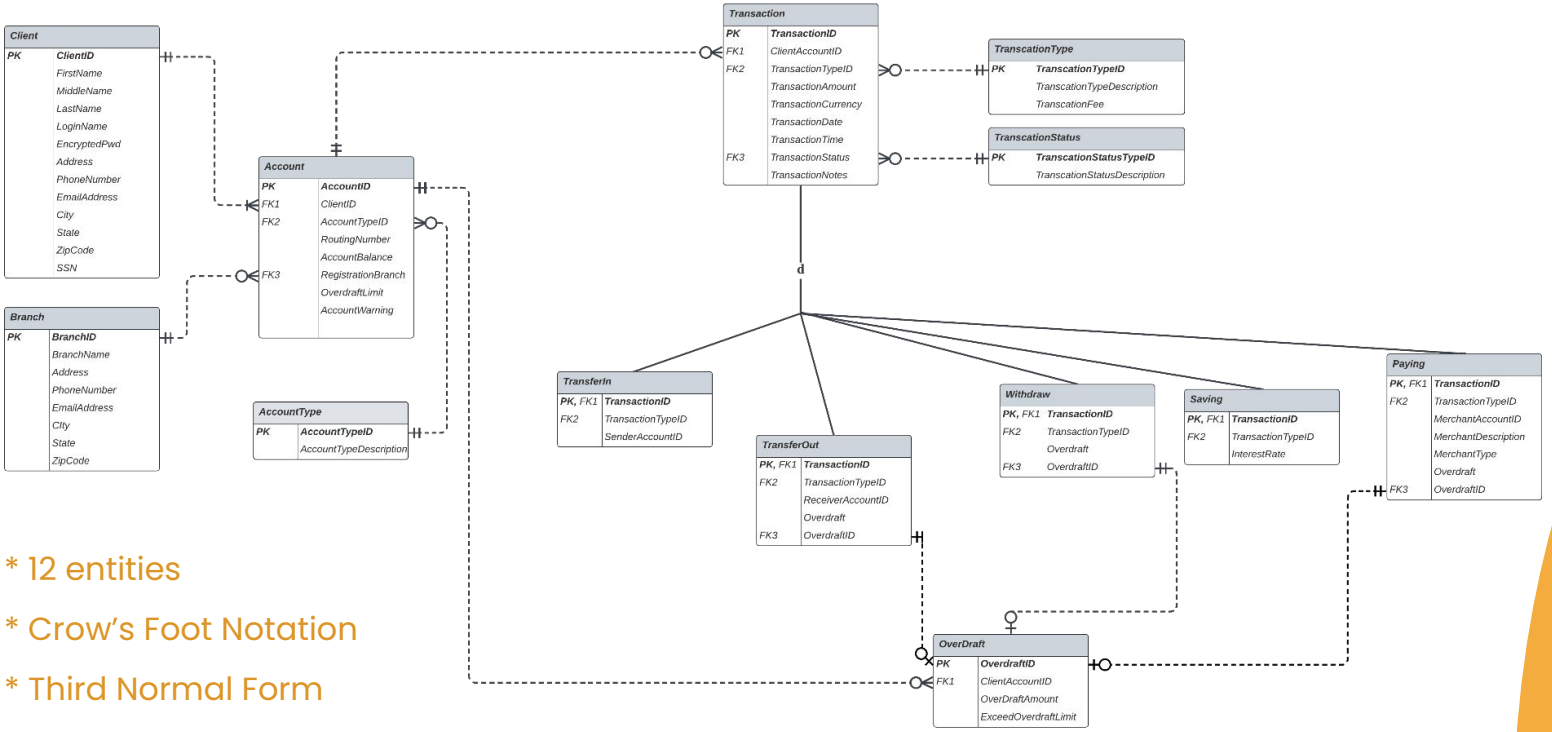
The business rules are...

- Each client may have one or two accounts, including saving and checking accounts
- Each client may have zero or more transactions
- Each account can only belong to one client
- Each account may have one and only one account type
- Each account may have zero or more transactions
- Each account may belong to one and only one branch

02

ER Diagram

ERD Overview



* 12 entities
* Crow's Foot Notation
* Third Normal Form

Design Decisions

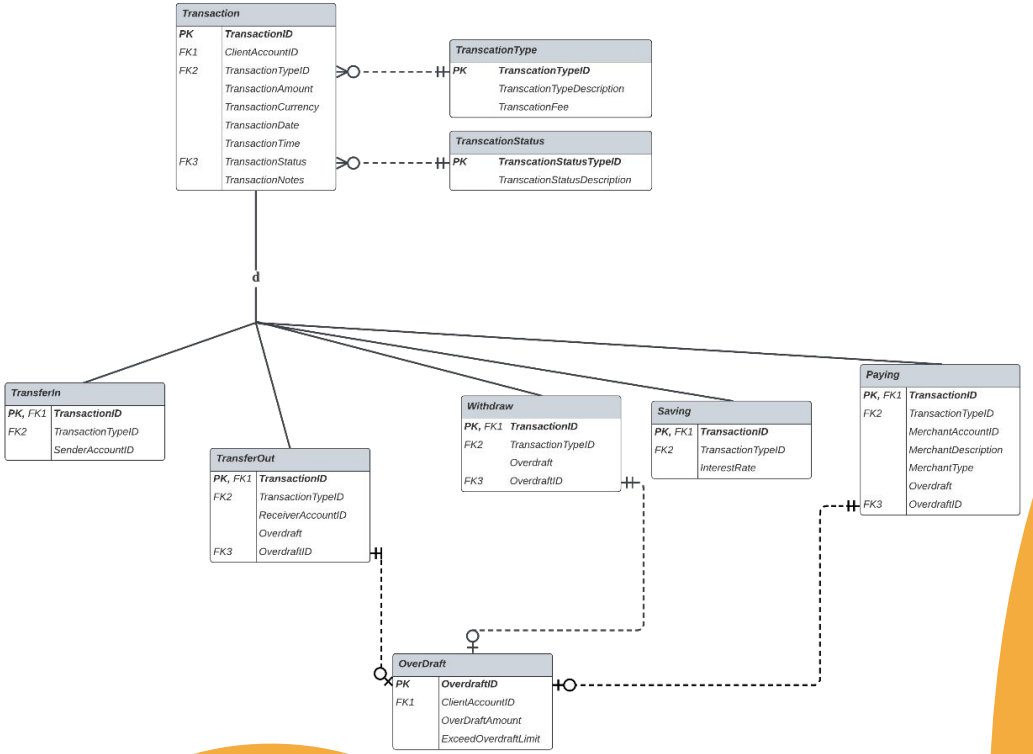
Entity Name	Purpose/Significance	Relationship to Other Entities
Client	Stores **client personal information** such as name, address, and contact information, each identified by a unique ClientID.	**Client ← Account (One to Many)**: One client can have multiple accounts – as the parent entity to Account, its primary key will serve as the foreign key to the Account entity.
Account	Stores **account information** including account type, number, balance, overdraft limit, warning, etc., with AccountID as its unique identifier.	**Account ← Transaction (One to Many)**: **Account ← OverDraft (One to Many)**: Account is the parent entity for Transaction and OverDraft – each account can be associated with multiple transactions or overdraft records.
Transaction	Stores **transaction details** including transaction type, amount, date, status, and description, with a unique TransactionID assigned to each transaction record.	**Transaction → TransactionType (Many to One)** **Transaction → TransactionStatus (Many to One)**: Each transaction can only belong to one type and one status. Transaction will be further separated into five subtypes, using transaction type as the discriminator.

Enhanced ER Model

Purpose: To more precisely model the complex system

- **Flexibility:** Stores subclass-specific attributes and relationships separately
- **Efficiency:** Reduces NULL entries and avoid maintenance burden

Specialization: Top-down process of subdividing an existing superclass into more specialized subclasses



Enhanced ER Model

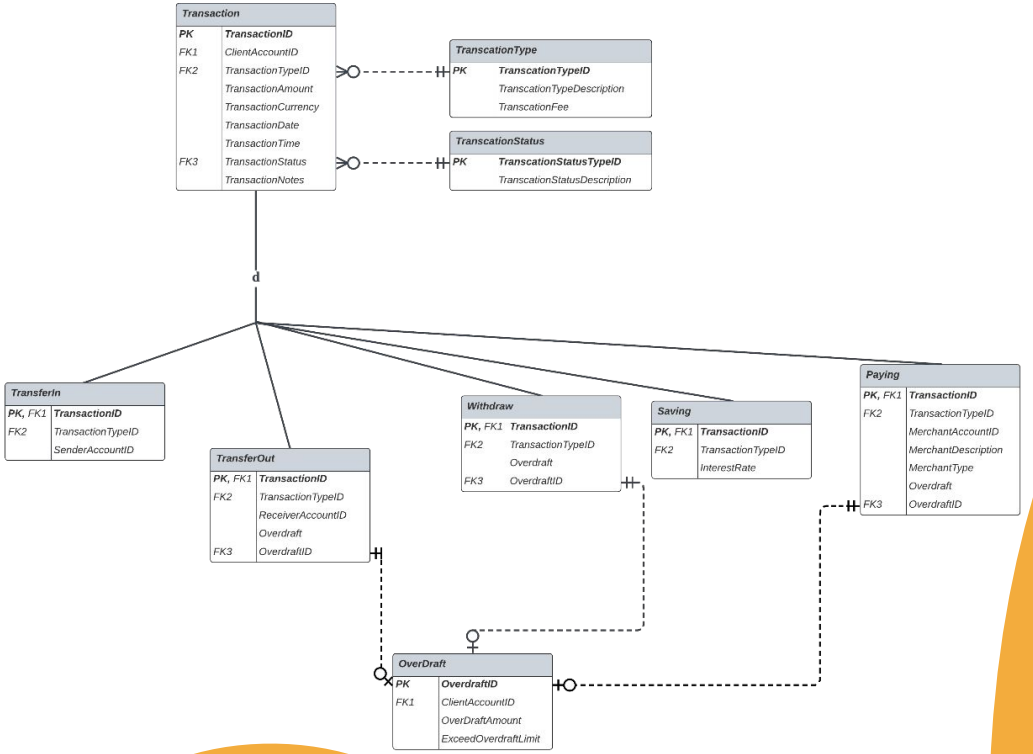
Subtype Discriminator: TransactionType

Superclass: Transaction

- Stores common attributes for all types of transactions

Subclasses: TransferIn, TransferOut, Withdraw, Saving, Paying

- Stores unique attributes apart from inherited ones

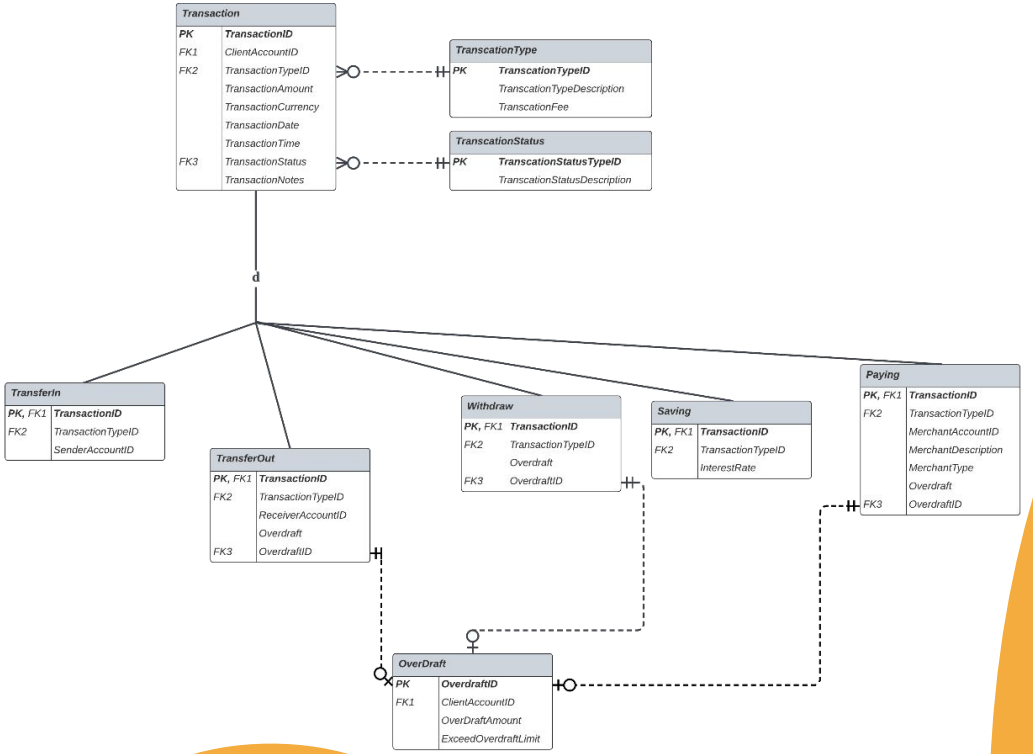


Enhanced ER Model

Business Rule: Each transaction can belong to zero or one type

Constraint: {Optional, Or}

- **Optional Participation**: A superclass member need not be a member of any subclass
- **Disjoint Rule**: Non-overlapping set of entities



03

SQL DDL Statements

Client DDL

IDENTITY
Automatic ID generation
Start from 10,000,000

```sql
-- Create Tables
CREATE TABLE Client
    (
    ClientID int IDENTITY(10000000, 1) NOT NULL PRIMARY KEY,
    FirstName varchar(40) NOT NULL,
    MiddleName varchar(40),
    LastName varchar(40) NOT NULL,
    LoginName varchar(40) NOT NULL,
    EncryptedPassword varbinary(250),
    PhoneNumber varchar(20) NOT NULL,
    EmailAddress varchar(100) NOT NULL,
    [Address] varchar(200) NOT NULL,
    City varchar(30) NOT NULL,
    [State] varchar(30) NOT NULL,
    Zipcode varchar(10) NOT NULL,
    SSN varchar(11)
    );
```

Encryption
Varbinary data type

```sql
--========== Column Encryption ==========--

-- Create DMK
CREATE MASTER KEY
ENCRYPTION BY PASSWORD = 'Test_P@sswOrd';

-- Create certificate to protect symmetric key
CREATE CERTIFICATE TestCertificate
WITH SUBJECT = 'Team6 Test Certificate',
EXPIRY_DATE = '2026-10-31';

-- Create symmetric key to encrypt data
CREATE SYMMETRIC KEY TestSymmetricKey
WITH ALGORITHM = AES_128
ENCRYPTION BY CERTIFICATE TestCertificate;

-- Open symmetric key
OPEN SYMMETRIC KEY TestSymmetricKey
DECRYPTION BY CERTIFICATE TestCertificate;
```

Varchar
Varbinary data type

Account, AccountType & Branch DDL

```sql
CREATE TABLE Account
    (
    AccountID int IDENTITY(10000000, 1) NOT NULL PRIMARY KEY,
    ClientID int NOT NULL
    REFERENCES Client(ClientID),
    AccountTypeID tinyint NOT NULL
    REFERENCES AccountType(AccountTypeID),
    RoutingNumber varchar(9) NOT NULL,
    AccountBalance money NOT NULL,
    RegistrationBranch int NOT NULL
    REFERENCES Branch(BranchID),
    InterestRate decimal(6, 2) NOT NULL CHECK (InterestRate >= 0),
    OverDraftLimit money NOT NULL CHECK (OverDraftLimit >= 0),
    AccountWarning bit NOT NULL
    );
```

AccountType
Tinyint
1: saving account
2: checking account

```sql
CREATE TABLE AccountType
    (
    AccountTypeID tinyint NOT NULL PRIMARY KEY,
    AccountTypeDescription varchar(200)
    );
```

```sql
CREATE TABLE Branch
    (
    BranchID int IDENTITY NOT NULL PRIMARY KEY,
    BranchName varchar(100) NOT NULL,
    PhoneNumber varchar(20) NOT NULL,
    EmailAddress varchar(100) NOT NULL,
    [Address] varchar(200) NOT NULL,
    City varchar(30) NOT NULL,
    [State] varchar(30) NOT NULL,
    Zipcode varchar(10) NOT NULL
    );
```

Data type
InterestRate: decimal
OverDraftLimit: money
AccountingWarning: bit

Check
- InterestRate >= 0
- OverDraftLimit >= 0

Transaction DDL

```
CREATE TABLE [Transaction]
    (
    TransactionID int IDENTITY(1000000000, 1) NOT NULL PRIMARY KEY,
    ClientAccountID int NOT NULL
    REFERENCES Account(AccountID),
    TransactionTypeID tinyint NOT NULL
    REFERENCES TransactionType(TransactionTypeID),
    TransactionAmount money NOT NULL,
    TransactionCurrency varchar(15) NOT NULL,
    TransactionDate date NOT NULL,
    TransactionTime time NOT NULL,
    TransactionStatus tinyint NOT NULL
    REFERENCES TransactionStatus(TransactionStatusTypeID),
    TransactionNotes varchar(200),
    CONSTRAINT Transaction_AltPK UNIQUE(TransactionID, TransactionTypeID)
    );
```

TransactionType
Subtype discriminator
1: TransferIn, 2: TransferOut,
3: Withdraw 4: Saving 5: Paying

```
CREATE TABLE TransactionType
    (
    TransactionTypeID tinyint NOT NULL PRIMARY KEY,
    TransactionTypeDescription varchar(200) NOT NULL,
    TransactionFee decimal(6,2) NOT NULL CHECK (TransactionFee >= 0)
    );
```

```
CREATE TABLE TransactionStatus
    (
    TransactionStatusTypeID tinyint NOT NULL PRIMARY KEY,
    TransactionStatusDescription varchar(200) NOT NULL,
    );
```

TransferIn, TransferOut & Withdraw DDL

```
CREATE TABLE TransferIn
    (
    TransactionID int PRIMARY KEY NOT NULL,
    TransactionTypeID tinyint NOT NULL CHECK (TransactionTypeID = 1),
    SenderAccountID varchar(20) NOT NULL,
    FOREIGN KEY (TransactionID, TransactionTypeID)
    REFERENCES [Transaction](TransactionID, TransactionTypeID)
    );
```

```
CREATE TABLE TransferOut
    (
    TransactionID int PRIMARY KEY NOT NULL,
    TransactionTypeID tinyint NOT NULL CHECK (TransactionTypeID = 2),
    ReceiverAccountID varchar(20) NOT NULL,
    Overdraft bit NOT NULL, -- 1: true; 0: false
    OverdraftID int
    REFERENCES Overdraft(OverdraftID),
    FOREIGN KEY (TransactionID, TransactionTypeID)
    REFERENCES [Transaction](TransactionID, TransactionTypeID)
    );
```

```
CREATE TABLE Withdraw
    (
    TransactionID int PRIMARY KEY NOT NULL,
    TransactionTypeID tinyint NOT NULL CHECK (TransactionTypeID = 3),
    Overdraft bit NOT NULL,
    OverdraftID int
    REFERENCES Overdraft(OverdraftID),
    FOREIGN KEY (TransactionID, TransactionTypeID)
    REFERENCES [Transaction](TransactionID, TransactionTypeID)
    );
```

Check
Subtype discriminator

Unique Fields

- **TransferIn:**
  SenderAccountID
- **TransferOut:**
  ReceiverAccountID
  Overdraft
  OverdraftID
- **Withdraw:**
  Overdraft
  OverdraftID

Saving & Paying DDL

```
CREATE TABLE Saving
    (
    TransactionID int PRIMARY KEY NOT NULL,
    TransactionTypeID tinyint NOT NULL CHECK (TransactionTypeID = 4),
    InterestRate decimal(6, 2) NOT NULL CHECK (InterestRate >= 0),
    FOREIGN KEY (TransactionID, TransactionTypeID)
    REFERENCES [Transaction](TransactionID, TransactionTypeID)
    );
```

```
CREATE TABLE Paying
    (
    TransactionID int PRIMARY KEY NOT NULL,
    TransactionTypeID tinyint NOT NULL CHECK (TransactionTypeID = 5),
    MerchantAccountID varchar(20) NOT NULL,
    MerchantDescription varchar(200),
    MerchantType varchar(100),
    Overdraft bit NOT NULL,
    OverdraftID int
    REFERENCES Overdraft(OverdraftID),
    FOREIGN KEY (TransactionID, TransactionTypeID)
    REFERENCES [Transaction](TransactionID, TransactionTypeID)
    );
```

Check
Subtype discriminator

Unique Fields

- **Saving:**
  InterestRate
- **Paying:**
  MerchantAccountID
  MerchantDescription
  MerchantType
  Overdraft
  OverdraftID

Overdraft DDL

```
CREATE TABLE OverDraft
    (
    OverDraftID int IDENTITY NOT NULL PRIMARY KEY,
    ClientAccountID int NOT NULL
    REFERENCES Account(AccountID),
    OverDraftAmount money NOT NULL,
    ExceedOverDraftLimit bit NOT NULL -- 1: exceed; 0: not exceed
    );
```

Account
Connect to the account that the overdraft belongs to

Overdraft Amount
-1 * (Original Account Balance - Transaction Amount)

ExceedOverDraftLimit
Whether exceed account overdraft limit

Trigger

```
CREATE TRIGGER TransactionTrigger ON
[Transaction]
AFTER INSERT
AS BEGIN
    DECLARE @tid int -- TransactionID
    DECLARE @ttype tinyint -- TransactionType
    DECLARE @accountid int -- ClientAccountID
    DECLARE @tamount money -- TransactionAmount
    DECLARE @originBalance money -- AccountBalance
    DECLARE @overdraftLimit money -- OverdraftLimit
    DECLARE @accountWarning bit -- AccountWarning
    DECLARE @overdraftAmount money -- OverDraftAmount
    SELECT @tid = i.TransactionID FROM inserted i
    SELECT @ttype = i.TransactionTypeID FROM inserted i
    SELECT @accountid = i.ClientAccountID FROM inserted i
    SELECT @tamount = i.TransactionAmount FROM inserted i
    SELECT @originBalance = AccountBalance FROM Account WHERE AccountID = @accountid
    SELECT @overdraftLimit = OverdraftLimit FROM Account WHERE AccountID = @accountid
    SELECT @accountWarning = AccountWarning FROM Account WHERE AccountID = @accountid
    SELECT @overdraftAmount = -1 * (@originBalance - @tamount)
    IF @ttype = 1 or @ttype = 4 -- "1: TransferIn; 4: Saving"
    BEGIN
        -- Update account balance
        UPDATE Account
        SET Account.AccountBalance = @originBalance + @tamount
        WHERE AccountID = @accountid
        -- If warning account client pays back
        IF @accountWarning = 1 AND @originBalance + @tamount >= 0
        BEGIN
            UPDATE Account
            SET Account.AccountWarning = 0
            WHERE AccountID = @accountid
        END
    END
    ELSE  -- "2: TransferOut; 3: Withdraw; 5: Paying"
    BEGIN
        -- Update account balance
        UPDATE Account
        SET Account.AccountBalance = @originBalance - @tamount
        WHERE AccountID = @accountid
        -- If there is overdraft, insert into Overdraft table
        IF @originBalance - @tamount < 0
        BEGIN
            -- If overdraft amount exceed the overdraft limit,
            -- set on AccountWarning bit and ExceedOverDraftLimit bit
            IF @overdraftAmount > @overdraftLimit
            BEGIN
                INSERT Overdraft
                VALUES (@accountid, @overdraftAmount, 1)

                UPDATE Account
                SET Account.AccountWarning = 1
                WHERE AccountID = @accountid
            END
            ELSE
            -- If overdraft amount does not exceed the overdraft limit,
            -- insert into Overdraft table with ExceedOverDraftLimit bit off.
            BEGIN
                INSERT Overdraft
                VALUES (@accountid, @overdraftAmount, 0)
            END
        END
    END
END
```

Motivation

- To keep Account table updated after each transaction
- To keep Overdraft table updated after overdraft transactions

Trigger

- Transaction table

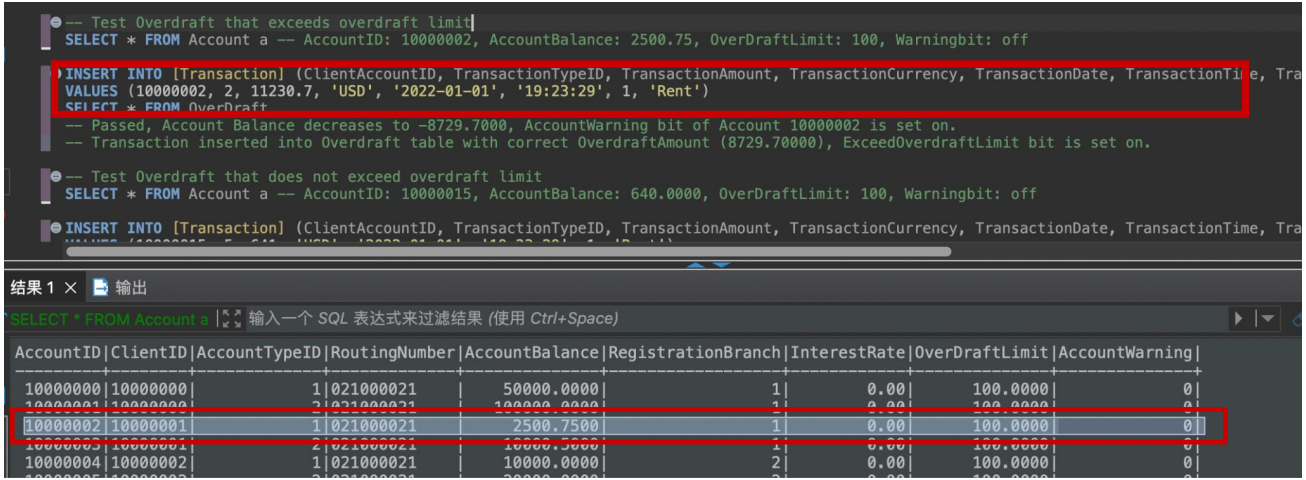
Functionality

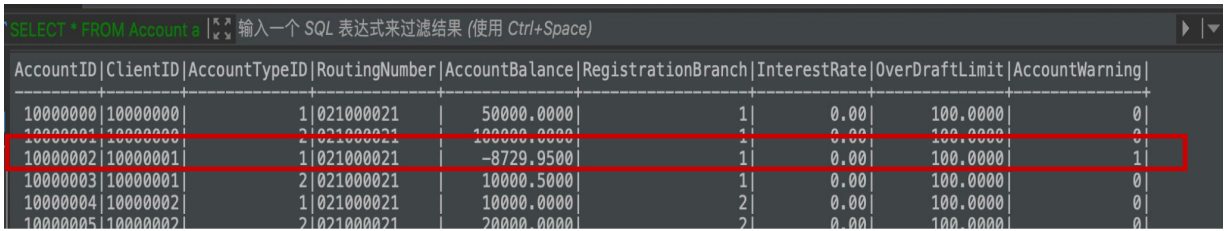
- Update the **account balance** after each transaction
- Set the **account warning bit** on when account overdraft exceed the overdraft limit
- Set the **account warning bit** off when the client pays back previous overdraft
- Insert into the **Overdraft table** when there is a overdraft transaction inserted into the **Transaction table**

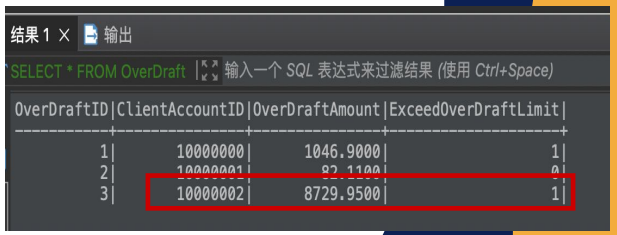
Trigger Test

Functionality

- Set the **account warning bit** on when account overdraft exceed the overdraft limit



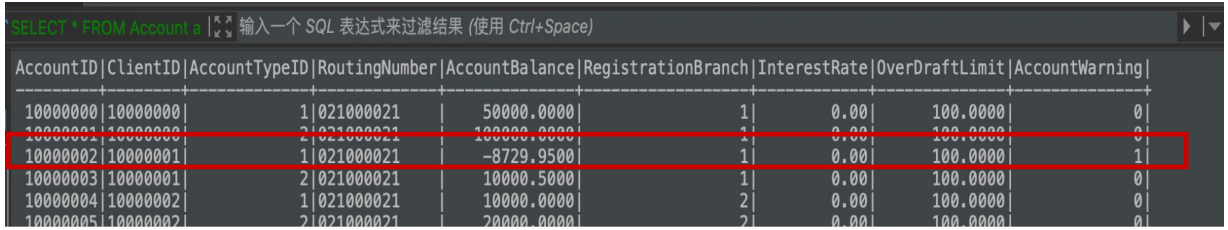




Trigger Test

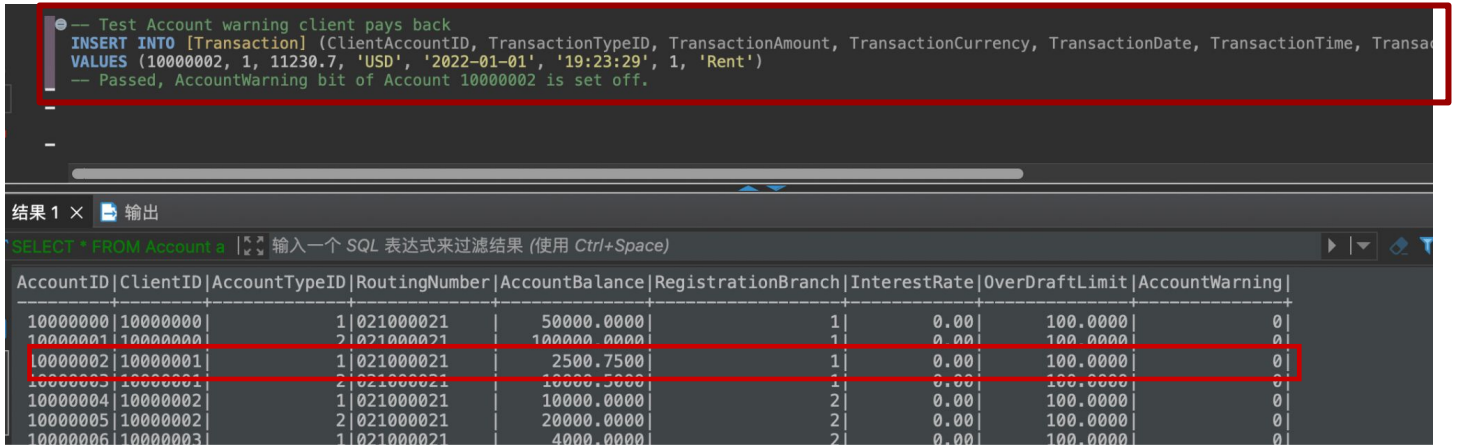
Functionality

- Set the **account warning bit** off when the client pays back previous overdraft



```
SELECT * FROM Account a  输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

AccountID|ClientID|AccountTypeID|RoutingNumber|AccountBalance|RegistrationBranch|InterestRate|OverDraftLimit|AccountWarning|
---------+--------+-------------+-------------+--------------+------------------+------------+--------------+--------------+
 10000000|10000000|            1|021000021    |    50000.0000|                 1|        0.00|      100.0000|             0|
 10000001|10000000|            2|021000021    |   100000.0000|                 1|        0.00|      100.0000|             0|
 10000002|10000001|            1|021000021    |    -8729.9500|                 1|        0.00|      100.0000|             1|
 10000003|10000001|            2|021000021    |    10000.5000|                 1|        0.00|      100.0000|             0|
 10000004|10000002|            1|021000021    |    10000.0000|                 2|        0.00|      100.0000|             0|
 10000005|10000002|            2|021000021    |    20000.0000|                 2|        0.00|      100.0000|             0|
```



```
-- Test Account warning client pays back
INSERT INTO [Transaction] (ClientAccountID, TransactionTypeID, TransactionAmount, TransactionCurrency, TransactionDate, TransactionTime, Transac
VALUES (10000002, 1, 11230.7, 'USD', '2022-01-01', '19:23:29', 1, 'Rent')
-- Passed, AccountWarning bit of Account 10000002 is set off.
```

结果 1 × 输出

```
SELECT * FROM Account a  输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

AccountID|ClientID|AccountTypeID|RoutingNumber|AccountBalance|RegistrationBranch|InterestRate|OverDraftLimit|AccountWarning|
---------+--------+-------------+-------------+--------------+------------------+------------+--------------+--------------+
 10000000|10000000|            1|021000021    |    50000.0000|                 1|        0.00|      100.0000|             0|
 10000001|10000000|            2|021000021    |   100000.0000|                 1|        0.00|      100.0000|             0|
 10000002|10000001|            1|021000021    |     2500.7500|                 1|        0.00|      100.0000|             0|
 10000003|10000001|            2|021000021    |    10000.5000|                 1|        0.00|      100.0000|             0|
 10000004|10000002|            1|021000021    |    10000.0000|                 2|        0.00|      100.0000|             0|
 10000005|10000002|            2|021000021    |    20000.0000|                 2|        0.00|      100.0000|             0|
 10000006|10000003|            1|021000021    |     4000.0000|                 2|        0.00|      100.0000|             0|
```

04

SQL Views

View 1:

Registration Branches & Their Performances

```
CREATE VIEW vwBranchesPerformances
    AS
    SELECT BranchID,
        BranchName,
        [Address] AS BranchAddress,
        City AS BranchCity,
        [State] AS BranchState,
        Zipcode AS BranchZipcode,
        SUM(T.TransactionAmount) AS TotalTransactionAmount
    FROM Branch b
    INNER JOIN Account a
        ON b.BranchID = a.RegistrationBranch
    INNER JOIN [Transaction] t
        ON a.AccountID = t.ClientAccountID
    GROUP BY BranchID, BranchName, [Address], City, [State], Zipcode;
```

View 1:

Registration Branches & Their Performances

```
SELECT * FROM vwBranchesPerformances;
```

	BranchID	BranchName	BranchAddress	BranchCity	BranchState	BranchZipcode	TotalTransactionAmount
1	1	CA_Los_Angeles_Branch1	123 Main St.	Los Angeles	CA	90001	77201.47
2	2	NY_New_York_Branch1	456 Elm St.	New York	NY	10001	26698.21
3	3	TX_Houston_Branch1	789 Oak St.	Houston	TX	77001	23712.07
4	4	AZ_Phoenix_Branch1	101 Maple Ave.	Phoenix	AZ	85001	16496.28
5	5	PA_Philadelphia_Branch1	543 Elm St.	Philadelphia	PA	19101	22313.69

View 2:

How much each client involve in different transaction activities

```
CREATE VIEW vwClientActivity
    WITH ENCRYPTION
    AS
    SELECT c.ClientID,
        c.FirstName,
        c.MiddleName,
        c.LastName,
        ta.TransactionTypeDescription AS TransactionType,
        ta.TransactionAmount
    FROM Client c
    INNER JOIN
    (SELECT ClientID, tt.TransactionTypeDescription, SUM(TransactionAmount) AS TransactionAmount
    FROM Account a
    INNER JOIN [Transaction] t
        ON a.AccountID = t.ClientAccountID
    INNER JOIN TransactionType tt
        ON t.TransactionTypeID = tt.TransactionTypeID
    GROUP BY ClientID, tt.TransactionTypeDescription) ta ON c.ClientID = ta.ClientID;
```

View 2:

How much each client involve in different transaction activities

```
SELECT *
FROM vwClientActivity
ORDER BY ClientID, TransactionType;
```

	ClientID	FirstName	MiddleName	LastName	TransactionType	TransactionAmount
1	10000000	Emily	G	Taylor	Paying	19866.99
2	10000000	Emily	G	Taylor	Saving	990.20
3	10000000	Emily	G	Taylor	Transfer In	10192.28
4	10000000	Emily	G	Taylor	Transfer Out	22779.96
5	10000000	Emily	G	Taylor	Withdraw	2233.19
6	10000001	Oliver	A	Wilson	Paying	13640.69
7	10000001	Oliver	A	Wilson	Saving	108.37
8	10000001	Oliver	A	Wilson	Withdraw	7389.79
9	10000002	Emily	B	Smith	Paying	10925.11
10	10000002	Emily	B	Smith	Saving	631.62
11	10000002	Emily	B	Smith	Transfer In	167.30
12	10000002	Emily	B	Smith	Transfer Out	2284.93
13	10000002	Emily	B	Smith	Withdraw	55.35
14	10000003	John	M	Davis	Paying	6382.17
15	10000003	John	M	Davis	Saving	1760.26
16	10000003	John	M	Davis	Transfer In	482.83
17	10000003	John	M	Davis	Transfer Out	2102.10

View 3:

Overdraft transactions and their corresponding clients & accounts

```
CREATE VIEW vwAccountOverdraft
    WITH ENCRYPTION
    AS
    SELECT c.ClientID,
        c.FirstName,
        c.MiddleName,
        c.LastName,
        a.AccountID,
        o.OverDraftAmount,
        o.ExceedOverDraftLimit
    FROM Account a
    INNER JOIN OverDraft o
        ON a.AccountID = o.ClientAccountID
    INNER JOIN Client c
        ON a.ClientID = c.ClientID;
```

View 3:

Overdraft transactions and their corresponding clients & accounts

```
SELECT * FROM vwAccountOverdraft;
```

	ClientID	FirstName	MiddleName	LastName	AccountID	OverDraftAmount	ExceedOverDraftLimit
1	10000000	Emily	G	Taylor	10000000	1046.90	1
2	10000000	Emily	G	Taylor	10000001	82.11	0

View 4:
Clients & Their Accounts

```
CREATE VIEW vwClientAccounts
    WITH ENCRYPTION
    AS
    SELECT c.ClientID,
        c.FirstName,
        c.MiddleName,
        c.LastName,
        a.AccountID,
        atype.AccountTypeDescription AS AccountType,
        a.AccountBalance
    FROM Client c
    INNER JOIN Account a
        ON c.ClientID = a.ClientID
    INNER JOIN AccountType atype
        ON a.AccountTypeID = atype.AccountTypeID;
```

View 4:
Clients & Their Accounts

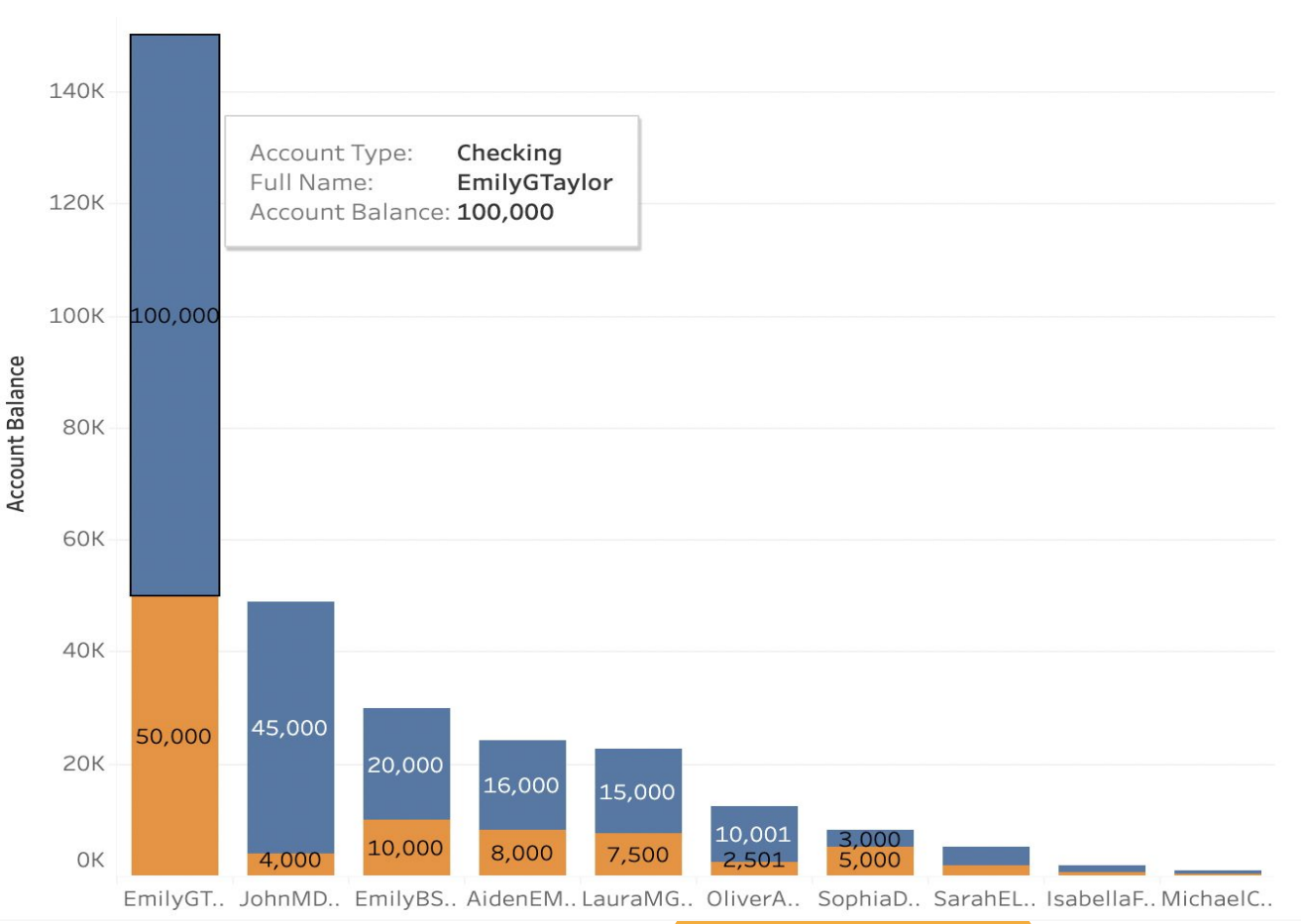
```
SELECT *
FROM vwClientAccounts
ORDER BY ClientID, AccountType
```

	ClientID	FirstName	MiddleName	LastName	AccountID	AccountType	AccountBalance
1	10000000	Emily	G	Taylor	10000001	Checking	100000.00
2	10000000	Emily	G	Taylor	10000000	Saving	50000.00
3	10000001	Oliver	A	Wilson	10000003	Checking	10000.50
4	10000001	Oliver	A	Wilson	10000002	Saving	2500.75
5	10000002	Emily	B	Smith	10000005	Checking	20000.00
6	10000002	Emily	B	Smith	10000004	Saving	10000.00
7	10000003	John	M	Davis	10000007	Checking	45000.00
8	10000003	John	M	Davis	10000006	Saving	4000.00
9	10000004	Sophia	D	Johnson	10000009	Checking	3000.00
10	10000004	Sophia	D	Johnson	10000008	Saving	5000.00
11	10000005	Sarah	E	Lee	10000011	Checking	3500.50
12	10000005	Sarah	E	Lee	10000010	Saving	1750.25
13	10000006	Aiden	E	Martinez	10000013	Checking	16000.00
14	10000006	Aiden	E	Martinez	10000012	Saving	8000.00
15	10000007	Michael	C	Johnson	10000015	Checking	640.00
16	10000007	Michael	C	Johnson	10000014	Saving	320.00
17	10000008	Laura	M	Garcia	10000017	Checking	15000.00
18	10000008	Laura	M	Garcia	10000016	Saving	7500.00
19	10000009	Isabella	F	Brown	10000019	Checking	1200.25
20	10000009	Isabella	F	Brown	10000018	Saving	600.50

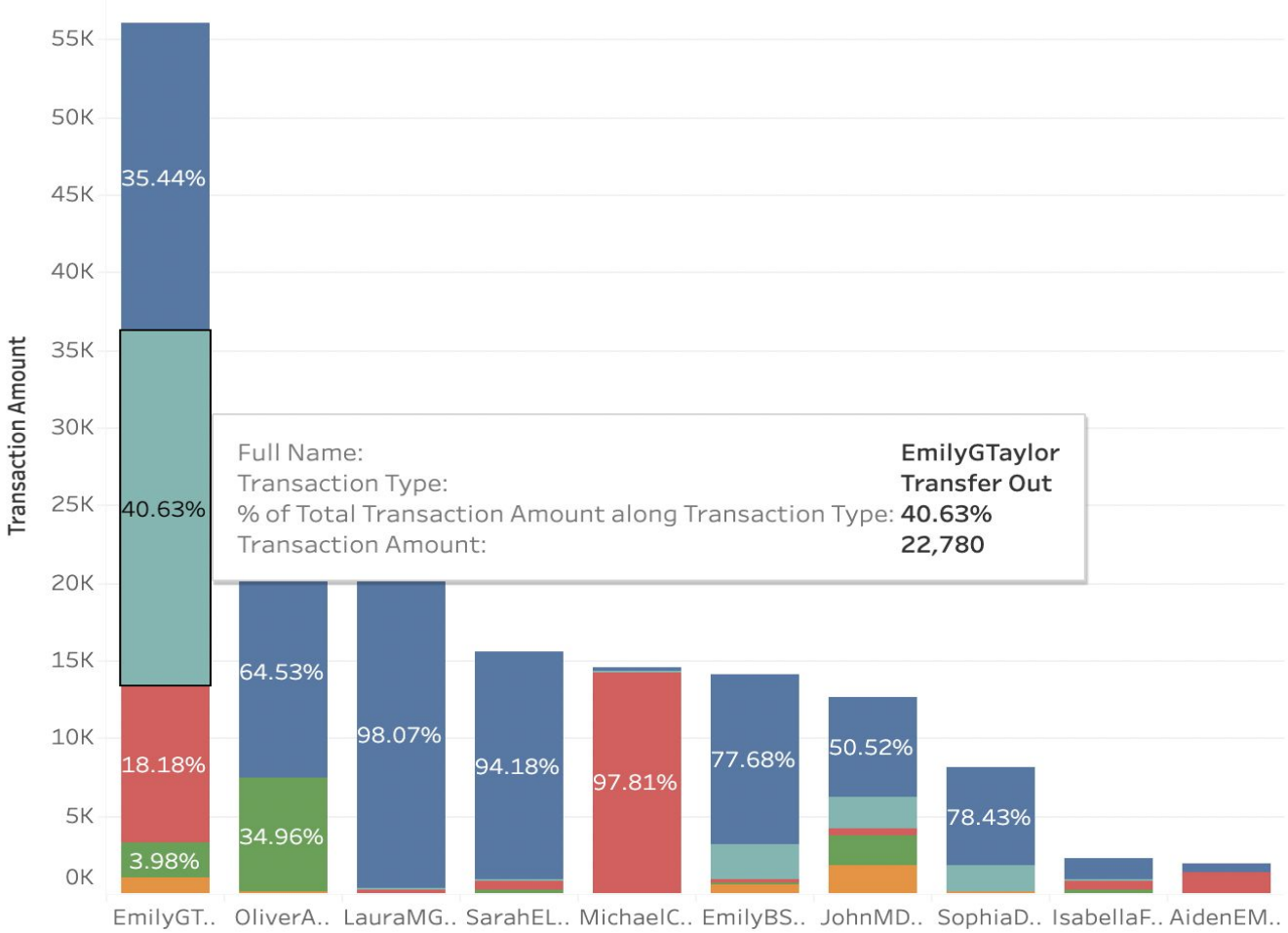
05

Tableau Visual Reports

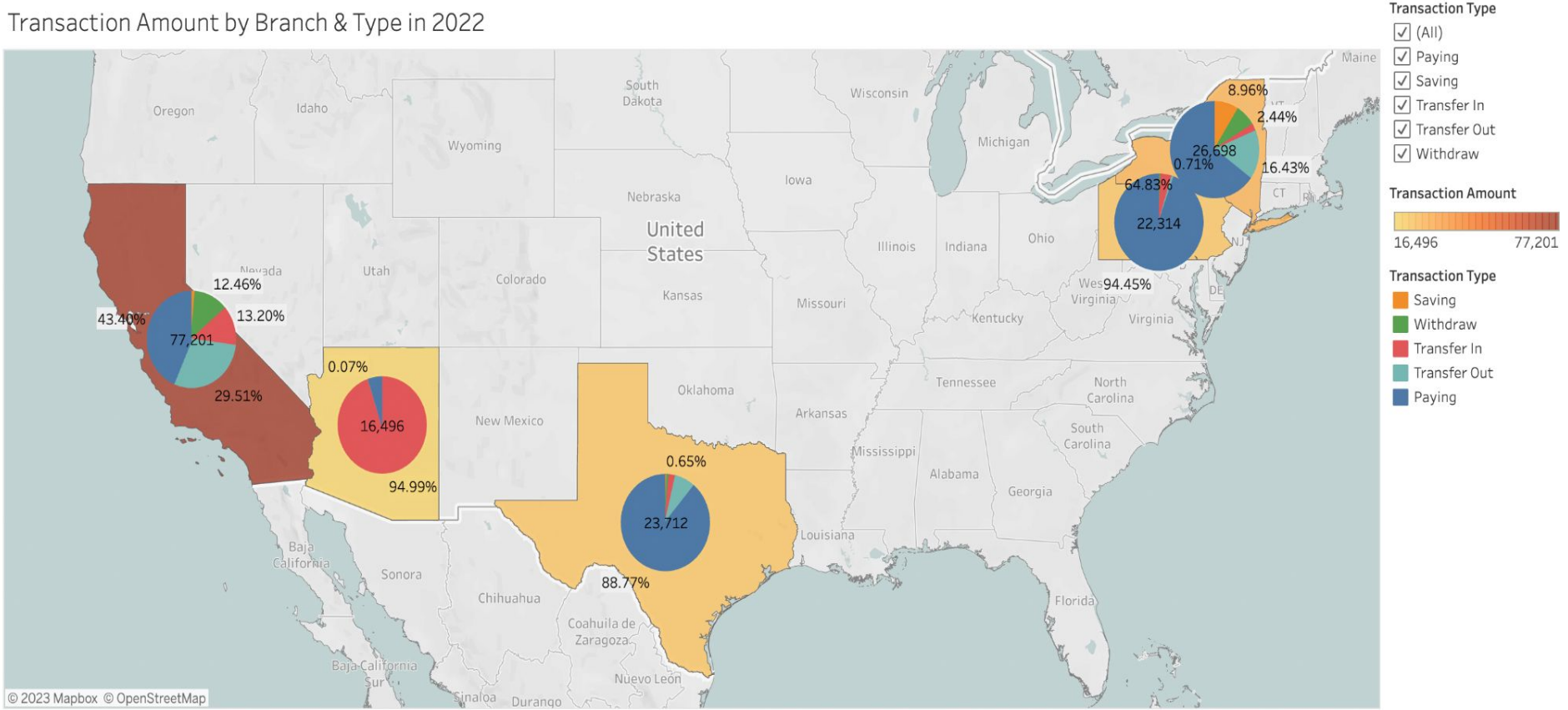
Client & Their Account Information



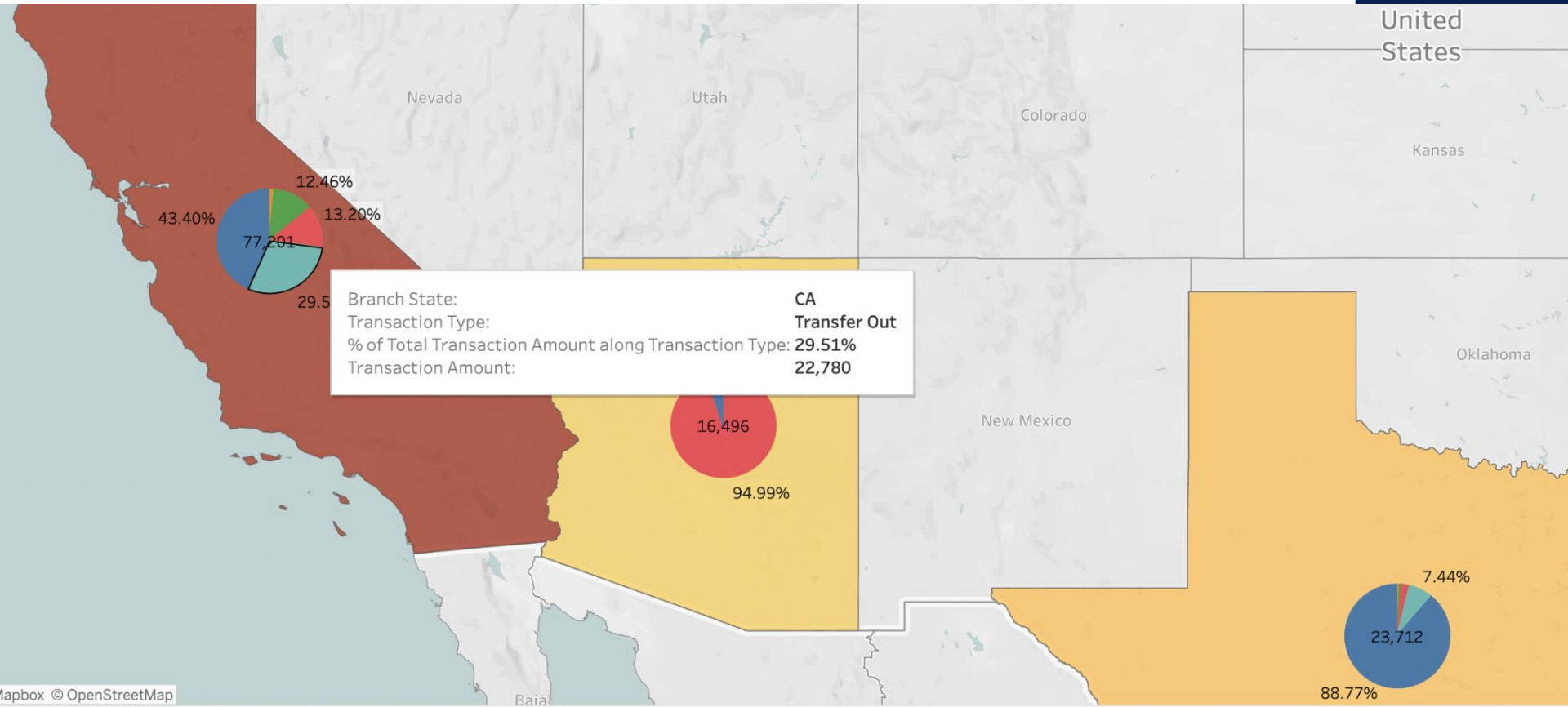
Individual Transaction Amount by Type in 2022



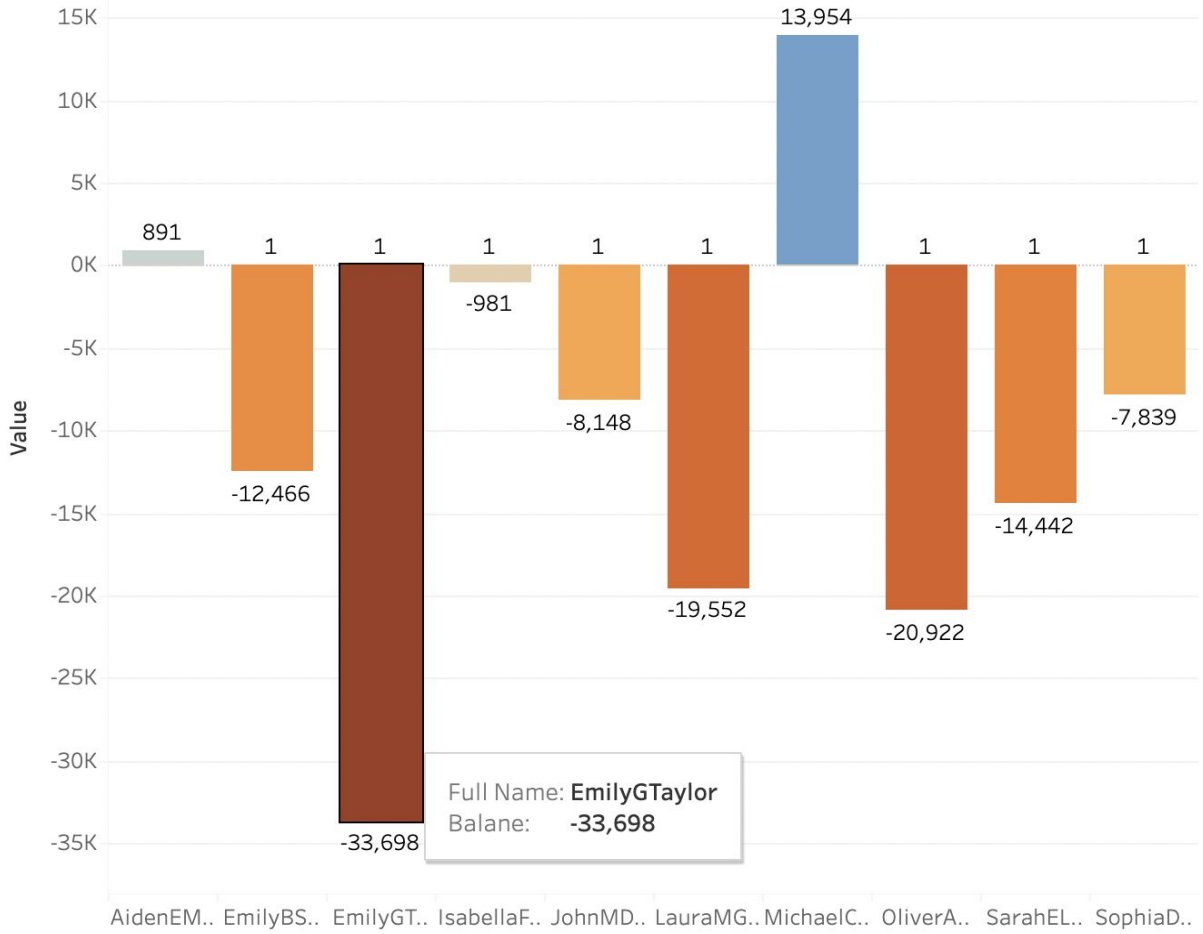
Transaction Amount by Branch & Type in 2022

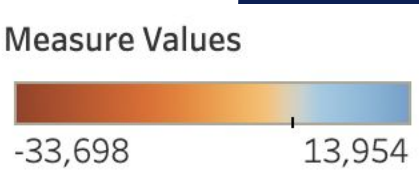


Transaction Amount by Branch & Type in 2022

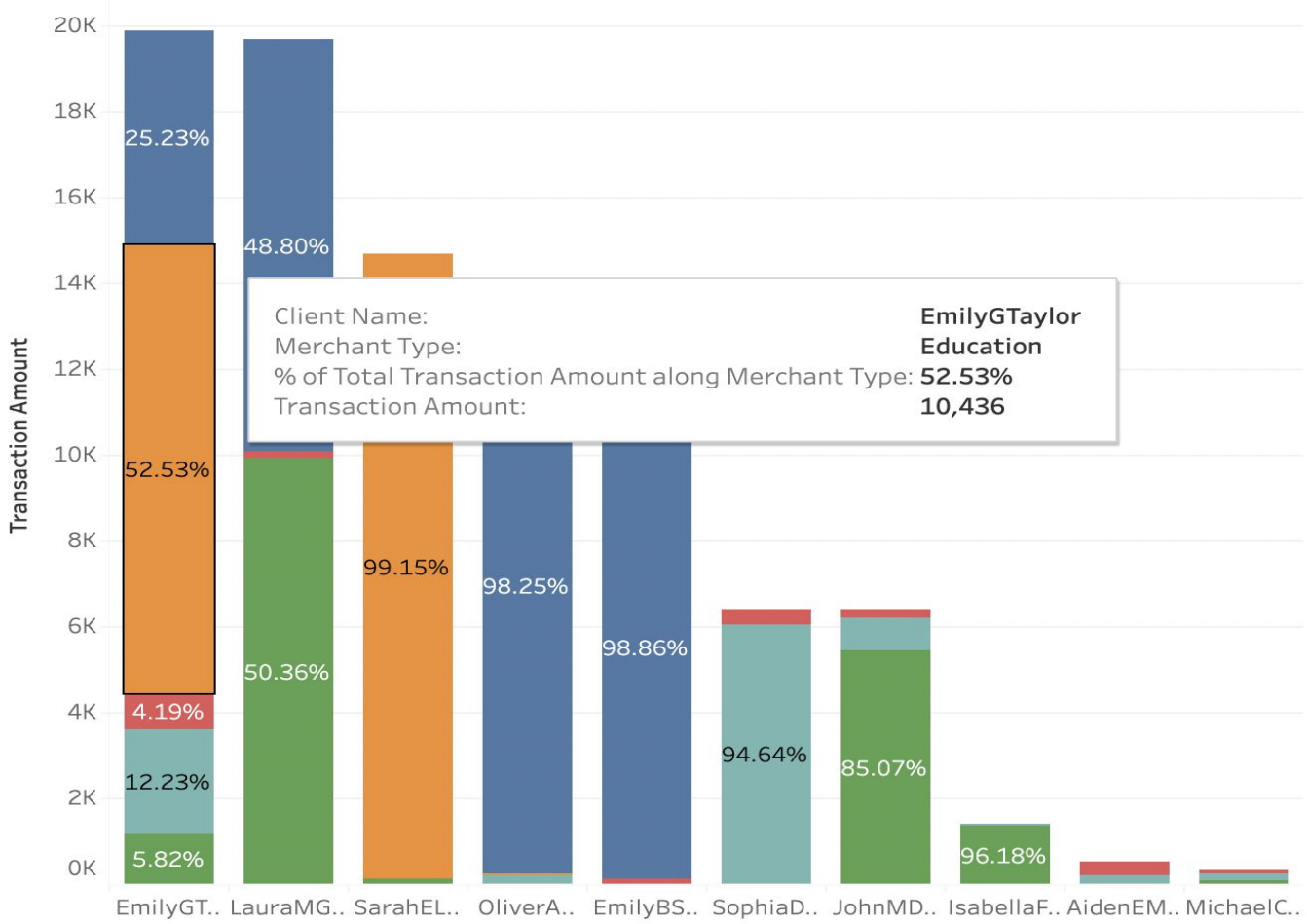


Individual Spending Behavior in 2022





Individual Spending Behavior in 2022 by Merchant Type



Thanks!

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik, and illustrations by Storyset