FINANCIAL BUDGETING APPLICATION

GROUP 11

Basak Bakirci

Ketan Keshava

Nagalekha Ramesh

Ruchika Shashidhara

BUSINESS PROBLEM

Many young adults unfortunately live paycheck to paycheck, they don't have a good view of their finances, they suffer from lifestyle creep wheere their spending increases with their income and their debts grow with a snowball effect without them realizing. They are often overwhelmed with the process of debt repayment, and they don't know where to start or where to do next without guidance from a financial advisor, which can come at its own additional cost.

MISSION STATEMENT

"Our goal is to empower professionals, between the ages of 25 and 60 by equipping them with the tools and insights to take charge of their finances, enhance their spending habits and build a financial future. With our NewSQL database we strive to offer a platform that not only tracks budgets, categorizes expenses and analyzes spending patterns but also facilitates effective debt repayment, efficient asset management and promotes financial transparency. By fostering a culture of improved saving habits, debt reduction strategies and financial literacy our aim is to guide individuals from living paycheck to paycheck towards a path of well being and long term prosperity."

BUSINESS OBJECTIVES

Improved Financial Understanding and Transparency

Enhance transparency and financial literacy by providing a holistic view of consumer transactions, debts and assets

Debt and Asset Management

Empower users to create structured plans for managing debts while enabling efficient management of assets for long term growth.

User Engagement and Motivation

Utilize existing user spending data to offer insights into spending trends, suggest avenues for financial advancement inspiring users to reach their objectives.

Scalability and Performance

Develop a scalable database infrastructure that can easily handle business expansion while emphasizing data analysis and performance to guarantee a good user experience

SOLUTION

Facilitate Budgeting

Facilitate budgeting, expense tracking, debt repayment, financial goal monitoring, and subscription management functionalities within the budgeting application.

Holistic View of Transactions

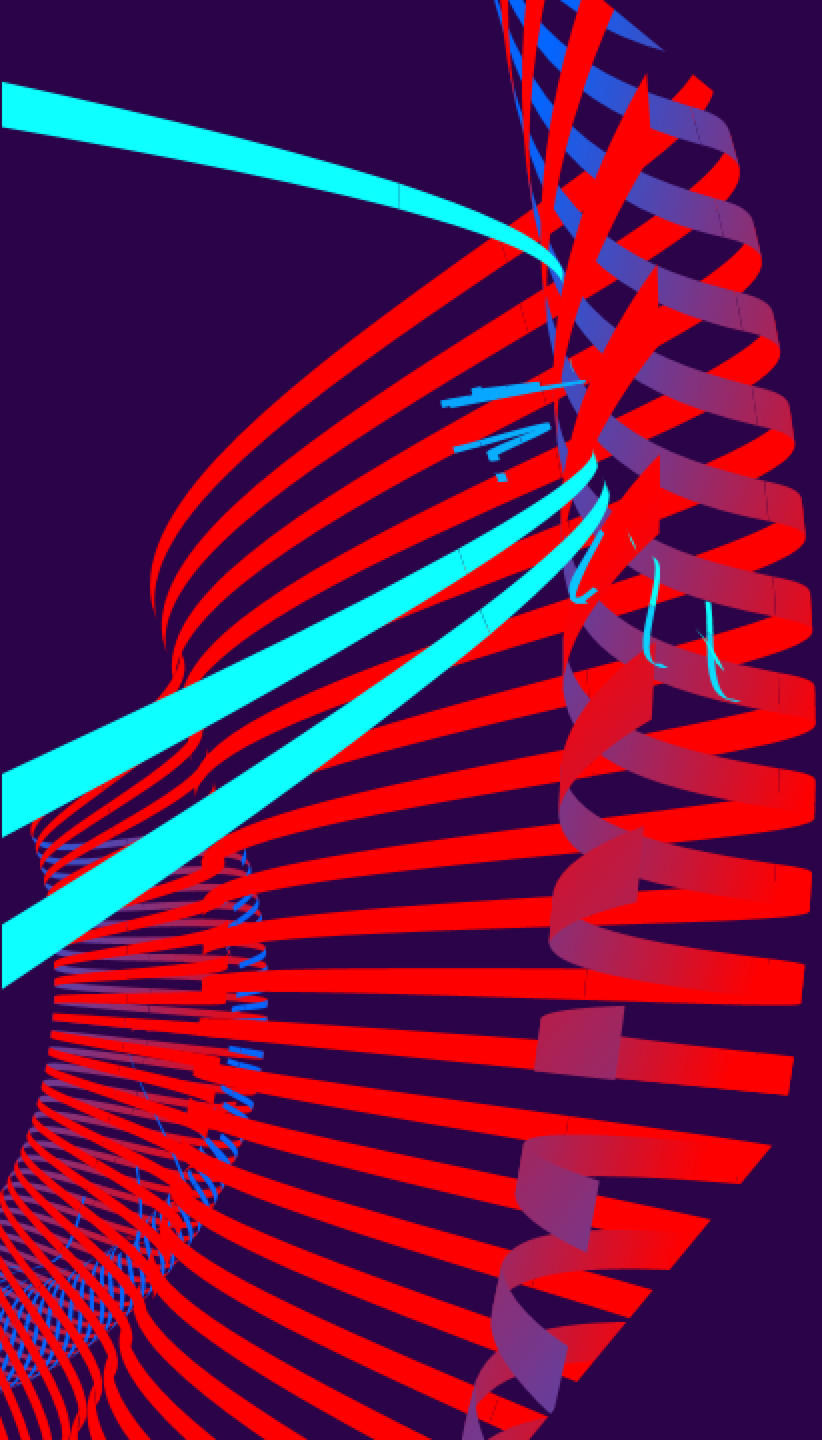
Provide transparency into transaction and spending habits holistically from different data sources.

Efficient Debt and Asset Management

Facilitate users to manage assets, debts, and subscriptions efficiently.

Tool for Accountability

Provide a tool for accountability in setting and monitoring financial goals while
introducing budgeting tools, expense tracking, and subscription management.



DATABASE OVERVIEW

DATABASE OBJECTIVES

To Maintain

Sensitive user data, transaction data, financial categorization data, asset and debt data

To Track The Status Of

Debt management, asset management, budgeting habits, spending habits, changes in net worth, overall finacial progress

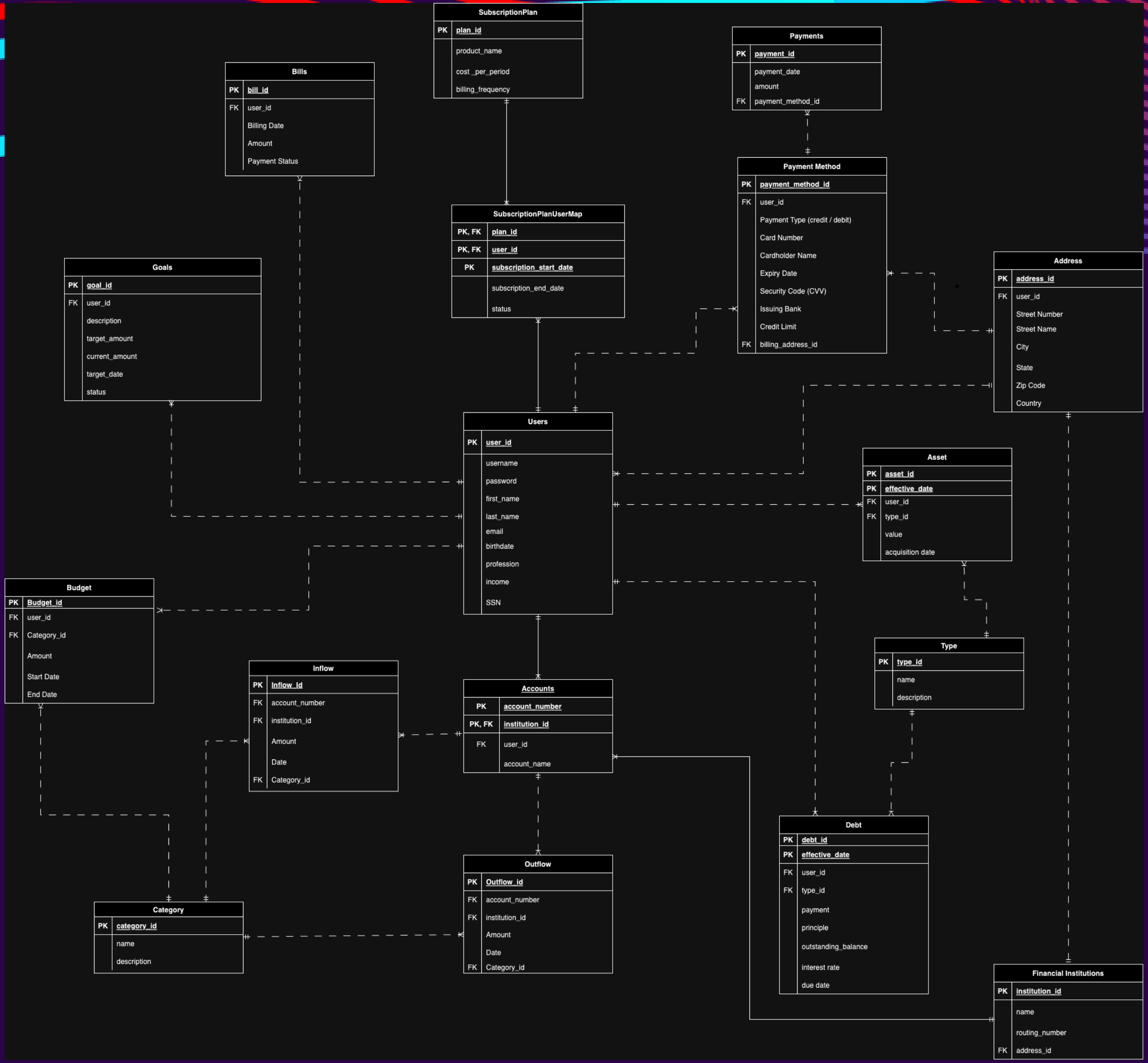
To Report On

Spending habits, debt reduction, asset growth, budget adherence, budget breakdowns by category, overall financial progress, spending trends

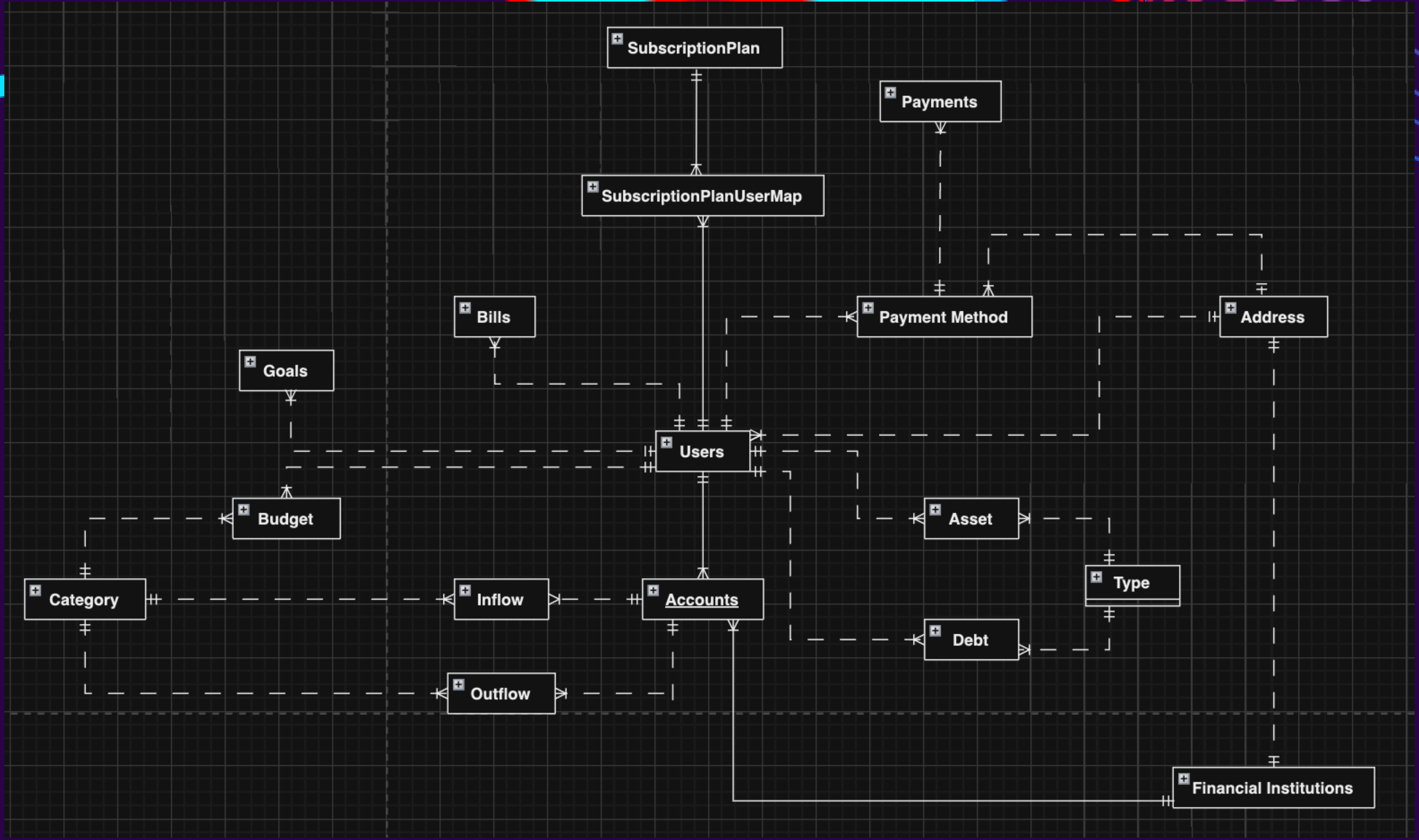
To Perform Search On

Transaction data, user profiles, financial history, assets and debts

FINAL ERD

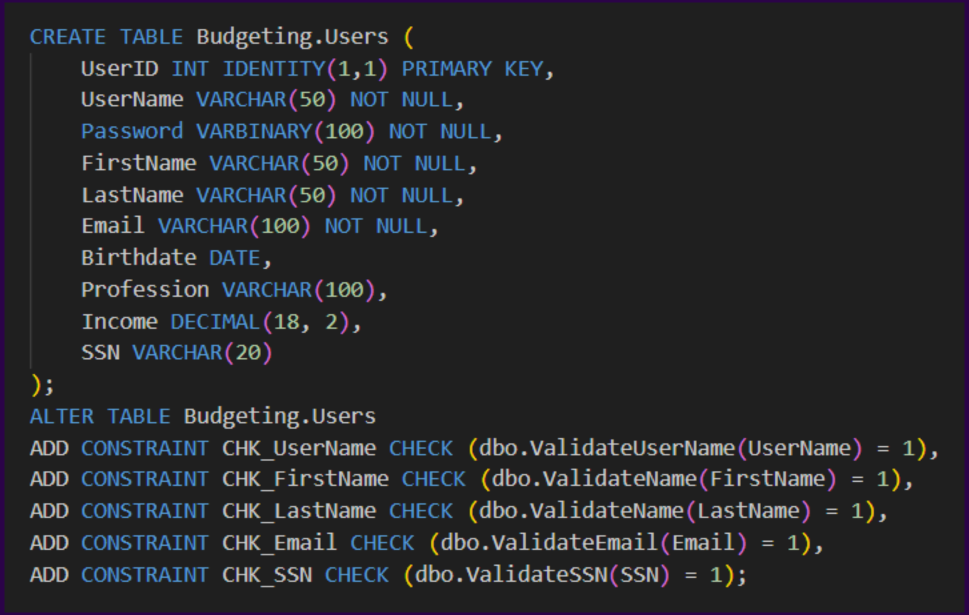

SubscriptionPlan
PK plan_id
product_name
cost _per_period
billing_frequency
Payments
PK payment_id
payment_date
amount
FK payment_method_id
Bills
PK bill_id
FK user_id
Billing Date
Amount
Payment Status
Payment Method
PK payment_method_id
FK user_id
Payment Type (credit / debit)
Card Number
Cardholder Name
Expiry Date
Security Code (CVV)
Issuing Bank
Credit Limit
FK billing_address_id
SubscriptionPlanUserMap
PK, FK plan_id
PK, FK user_id
PK subscription_start_date
subscription_end_date
status
Goals
PK goal_id
FK user_id
description
target_amount
current_amount
target_date
status
Address
PK address_id
FK user_id
Street Number
Street Name
City
State
Zip Code
Country
Users
PK user_id
username
password
first_name
last_name
email
birthdate
profession
income
SSN
Asset
PK asset_id
PK effective_date
FK user_id
FK type_id
value
acquisition date
Budget
PK Budget_id
FK user_id
FK Category_id
Amount
Start Date
End Date
Type
PK type_id
name
description
Inflow
PK inflow_id
FK account_number
FK institution_id
Amount
Date
FK Category_id
Accounts
PK account_number
PK, FK institution_id
FK user_id
account_name
Debt
PK debt_id
PK effective_date
FK user_id
FK type_id
payment
principle
outstanding_balance
interest rate
due date
Outflow
PK Outflow_id
FK account_number
FK institution_id
Amount
Date
FK Category_id
Category
PK category_id
name
description
Financial Institutions
PK institution_id
name
routing_number
FK address_id

FINAL ERD

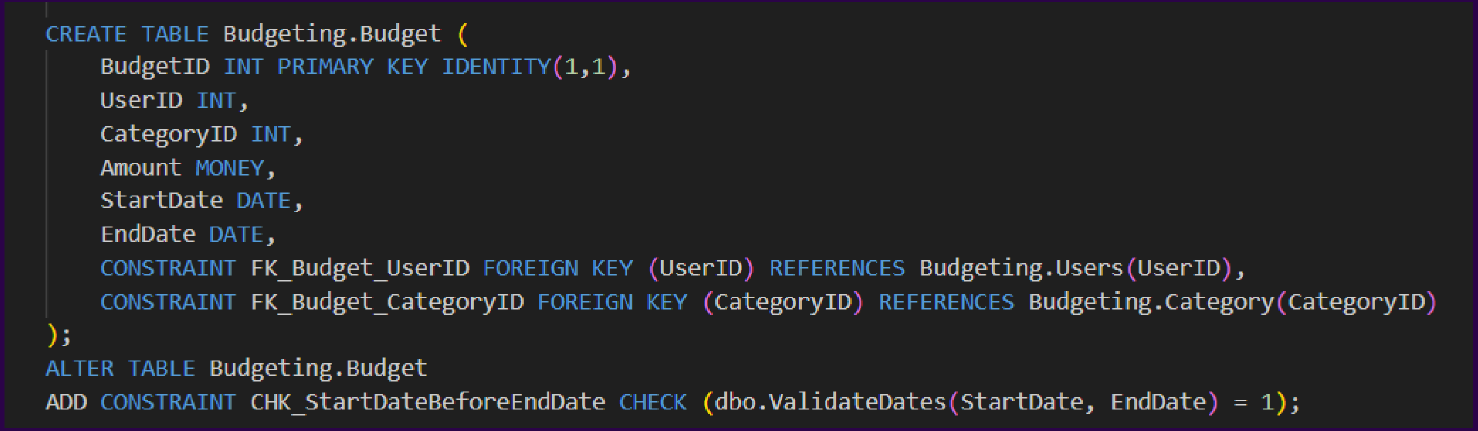


Link to ERD: https://bit.ly/final-erd-group-11

TABLE CREATION

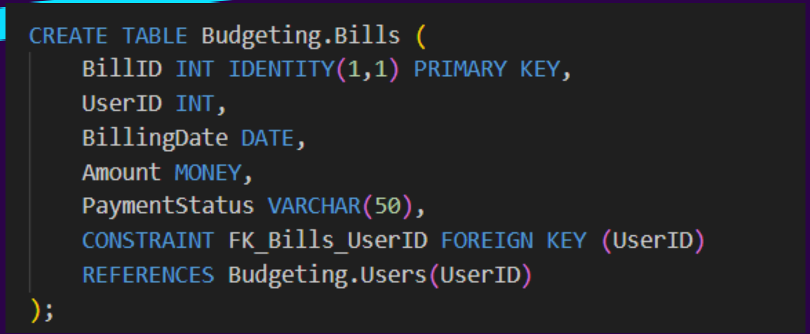


```sql
CREATE TABLE Budgeting.Users (
    UserID INT IDENTITY(1,1) PRIMARY KEY,
    UserName VARCHAR(50) NOT NULL,
    Password VARBINARY(100) NOT NULL,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    Birthdate DATE,
    Profession VARCHAR(100),
    Income DECIMAL(18, 2),
    SSN VARCHAR(20)
);
ALTER TABLE Budgeting.Users
ADD CONSTRAINT CHK_UserName CHECK (dbo.ValidateUserName(UserName) = 1),
ADD CONSTRAINT CHK_FirstName CHECK (dbo.ValidateName(FirstName) = 1),
ADD CONSTRAINT CHK_LastName CHECK (dbo.ValidateName(LastName) = 1),
ADD CONSTRAINT CHK_Email CHECK (dbo.ValidateEmail(Email) = 1),
ADD CONSTRAINT CHK_SSN CHECK (dbo.ValidateSSN(SSN) = 1);
```

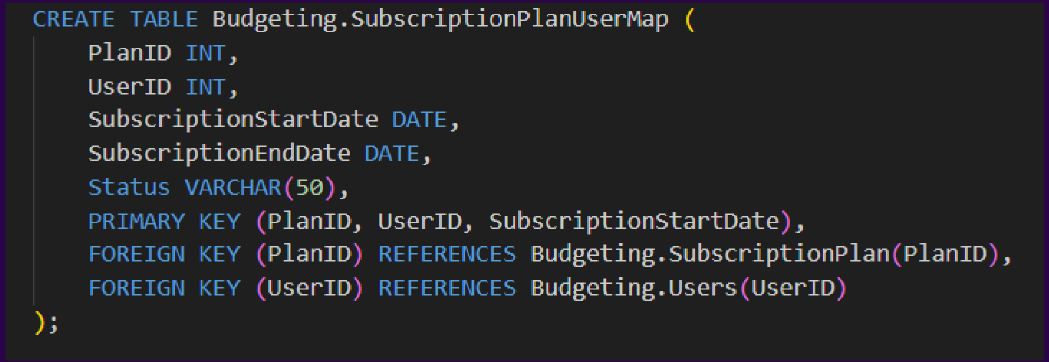


```sql
CREATE TABLE Budgeting.Budget (
    BudgetID INT PRIMARY KEY IDENTITY(1,1),
    UserID INT,
    CategoryID INT,
    Amount MONEY,
    StartDate DATE,
    EndDate DATE,
    CONSTRAINT FK_Budget_UserID FOREIGN KEY (UserID) REFERENCES Budgeting.Users(UserID),
    CONSTRAINT FK_Budget_CategoryID FOREIGN KEY (CategoryID) REFERENCES Budgeting.Category(CategoryID)
);
ALTER TABLE Budgeting.Budget
ADD CONSTRAINT CHK_StartDateBeforeEndDate CHECK (dbo.ValidateDates(StartDate, EndDate) = 1);
```

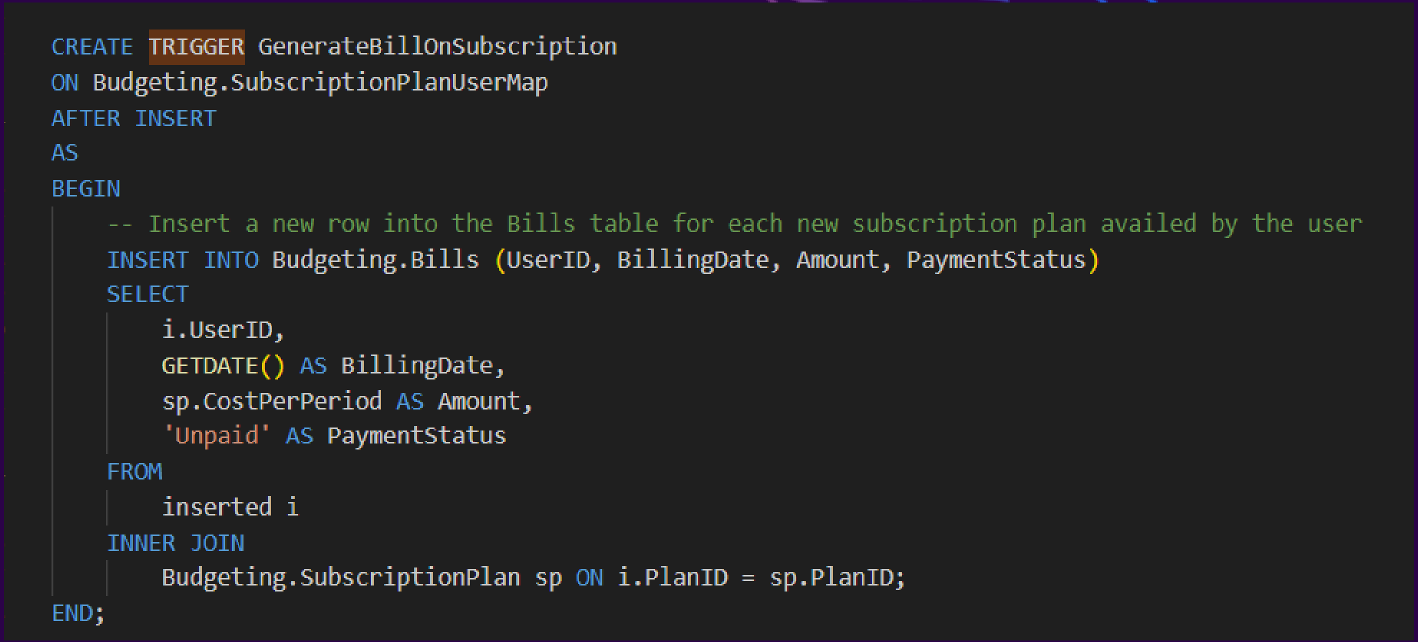
TABLE CREATION



```sql
CREATE TABLE Budgeting.Bills (
    BillID INT IDENTITY(1,1) PRIMARY KEY,
    UserID INT,
    BillingDate DATE,
    Amount MONEY,
    PaymentStatus VARCHAR(50),
    CONSTRAINT FK_Bills_UserID FOREIGN KEY (UserID)
    REFERENCES Budgeting.Users(UserID)
);
```

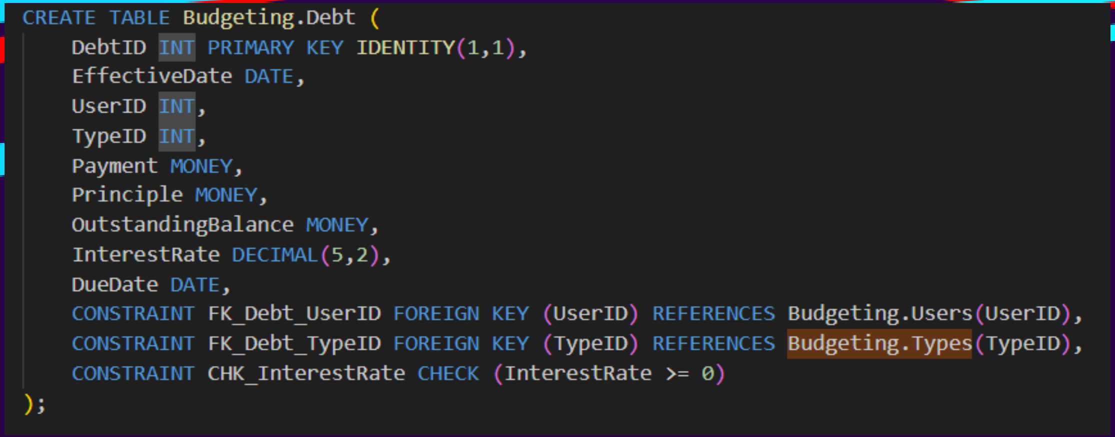


```sql
CREATE TABLE Budgeting.SubscriptionPlanUserMap (
    PlanID INT,
    UserID INT,
    SubscriptionStartDate DATE,
    SubscriptionEndDate DATE,
    Status VARCHAR(50),
    PRIMARY KEY (PlanID, UserID, SubscriptionStartDate),
    FOREIGN KEY (PlanID) REFERENCES Budgeting.SubscriptionPlan(PlanID),
    FOREIGN KEY (UserID) REFERENCES Budgeting.Users(UserID)
);
```

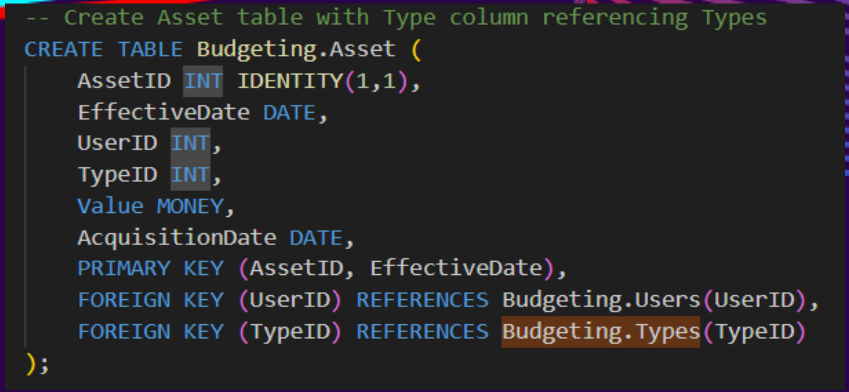


```sql
CREATE TRIGGER GenerateBillOnSubscription
ON Budgeting.SubscriptionPlanUserMap
AFTER INSERT
AS
BEGIN
    -- Insert a new row into the Bills table for each new subscription plan availed by the user
    INSERT INTO Budgeting.Bills (UserID, BillingDate, Amount, PaymentStatus)
    SELECT
        i.UserID,
        GETDATE() AS BillingDate,
        sp.CostPerPeriod AS Amount,
        'Unpaid' AS PaymentStatus
    FROM
        inserted i
    INNER JOIN
        Budgeting.SubscriptionPlan sp ON i.PlanID = sp.PlanID;
END;
```

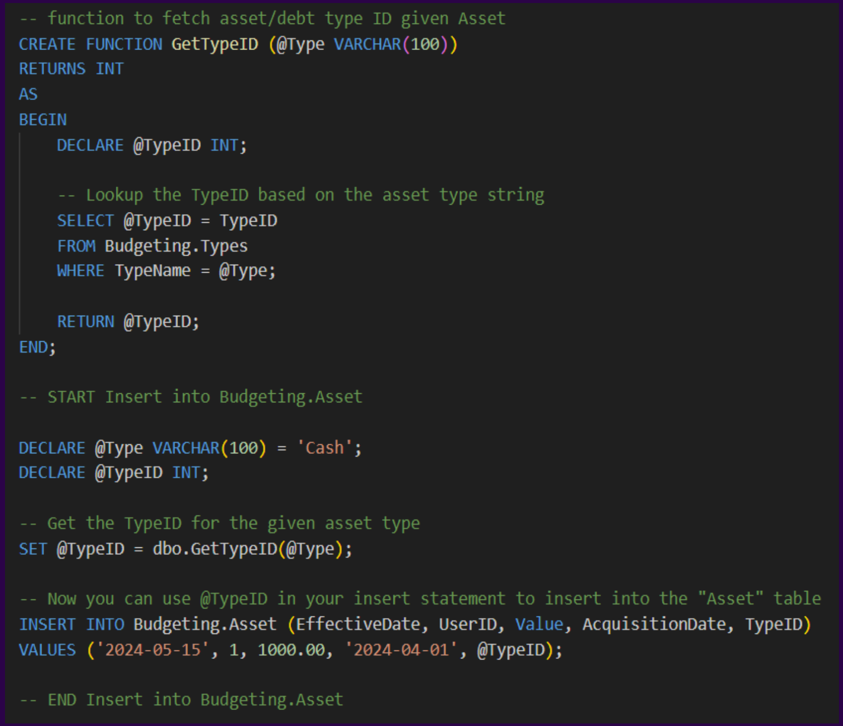
TABLE CREATION



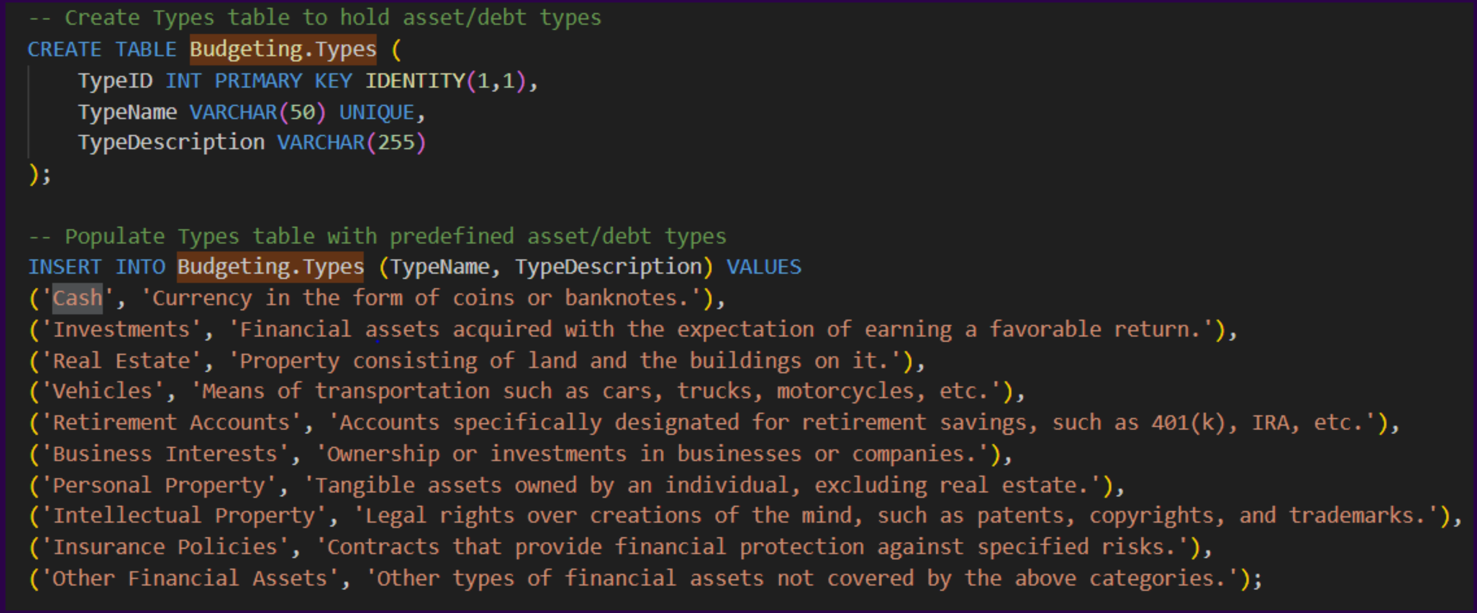
```sql
CREATE TABLE Budgeting.Debt (
    DebtID INT PRIMARY KEY IDENTITY(1,1),
    EffectiveDate DATE,
    UserID INT,
    TypeID INT,
    Payment MONEY,
    Principle MONEY,
    OutstandingBalance MONEY,
    InterestRate DECIMAL(5,2),
    DueDate DATE,
    CONSTRAINT FK_Debt_UserID FOREIGN KEY (UserID) REFERENCES Budgeting.Users(UserID),
    CONSTRAINT FK_Debt_TypeID FOREIGN KEY (TypeID) REFERENCES Budgeting.Types(TypeID),
    CONSTRAINT CHK_InterestRate CHECK (InterestRate >= 0)
);
```



```sql
-- Create Asset table with Type column referencing Types
CREATE TABLE Budgeting.Asset (
    AssetID INT IDENTITY(1,1),
    EffectiveDate DATE,
    UserID INT,
    TypeID INT,
    Value MONEY,
    AcquisitionDate DATE,
    PRIMARY KEY (AssetID, EffectiveDate),
    FOREIGN KEY (UserID) REFERENCES Budgeting.Users(UserID),
    FOREIGN KEY (TypeID) REFERENCES Budgeting.Types(TypeID)
);
```



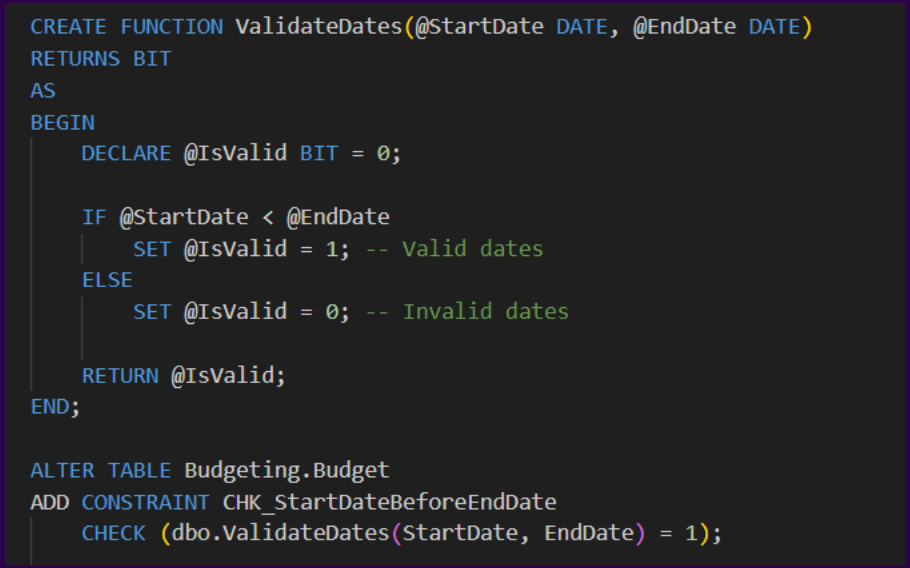
```sql
-- function to fetch asset/debt type ID given Asset
CREATE FUNCTION GetTypeID (@Type VARCHAR(100))
RETURNS INT
AS
BEGIN
    DECLARE @TypeID INT;

    -- Lookup the TypeID based on the asset type string
    SELECT @TypeID = TypeID
    FROM Budgeting.Types
    WHERE TypeName = @Type;

    RETURN @TypeID;
END;

-- START Insert into Budgeting.Asset

DECLARE @Type VARCHAR(100) = 'Cash';
DECLARE @TypeID INT;

-- Get the TypeID for the given asset type
SET @TypeID = dbo.GetTypeID(@Type);

-- Now you can use @TypeID in your insert statement to insert into the "Asset" table
INSERT INTO Budgeting.Asset (EffectiveDate, UserID, Value, AcquisitionDate, TypeID)
VALUES ('2024-05-15', 1, 1000.00, '2024-04-01', @TypeID);

-- END Insert into Budgeting.Asset
```



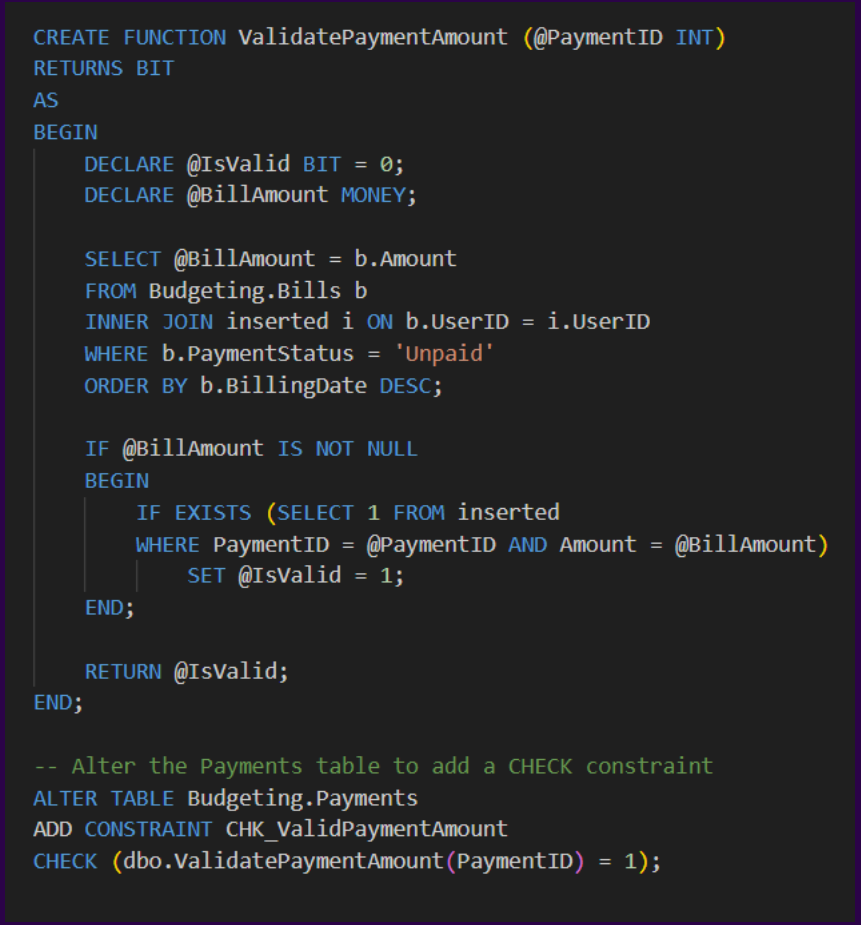
```sql
-- Create Types table to hold asset/debt types
CREATE TABLE Budgeting.Types (
    TypeID INT PRIMARY KEY IDENTITY(1,1),
    TypeName VARCHAR(50) UNIQUE,
    TypeDescription VARCHAR(255)
);

-- Populate Types table with predefined asset/debt types
INSERT INTO Budgeting.Types (TypeName, TypeDescription) VALUES
('Cash', 'Currency in the form of coins or banknotes.'),
('Investments', 'Financial assets acquired with the expectation of earning a favorable return.'),
('Real Estate', 'Property consisting of land and the buildings on it.'),
('Vehicles', 'Means of transportation such as cars, trucks, motorcycles, etc.'),
('Retirement Accounts', 'Accounts specifically designated for retirement savings, such as 401(k), IRA, etc.'),
('Business Interests', 'Ownership or investments in businesses or companies.'),
('Personal Property', 'Tangible assets owned by an individual, excluding real estate.'),
('Intellectual Property', 'Legal rights over creations of the mind, such as patents, copyrights, and trademarks.'),
('Insurance Policies', 'Contracts that provide financial protection against specified risks.'),
('Other Financial Assets', 'Other types of financial assets not covered by the above categories.');
```

TABLE LEVEL CHECK CONSTRAINTS

A table level check constraint that validates whether a Start Date is Before End Date in Budget Table



```sql
CREATE FUNCTION ValidateDates(@StartDate DATE, @EndDate DATE)
RETURNS BIT
AS
BEGIN
    DECLARE @IsValid BIT = 0;

    IF @StartDate < @EndDate
        SET @IsValid = 1; -- Valid dates
    ELSE
        SET @IsValid = 0; -- Invalid dates

    RETURN @IsValid;
END;

ALTER TABLE Budgeting.Budget
ADD CONSTRAINT CHK_StartDateBeforeEndDate
    CHECK (dbo.ValidateDates(StartDate, EndDate) = 1);
```

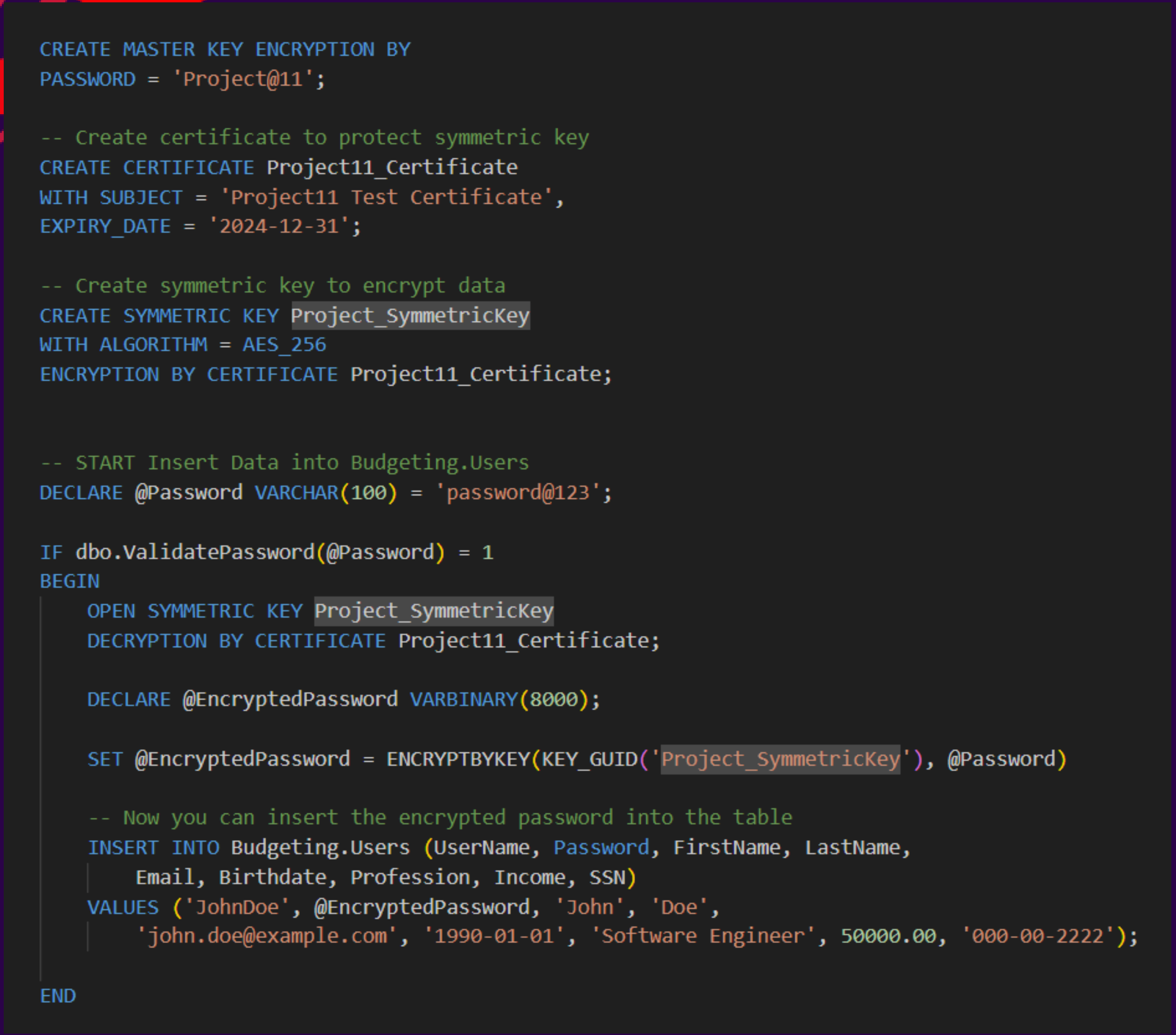


```sql
CREATE FUNCTION ValidatePaymentAmount (@PaymentID INT)
RETURNS BIT
AS
BEGIN
    DECLARE @IsValid BIT = 0;
    DECLARE @BillAmount MONEY;

    SELECT @BillAmount = b.Amount
    FROM Budgeting.Bills b
    INNER JOIN inserted i ON b.UserID = i.UserID
    WHERE b.PaymentStatus = 'Unpaid'
    ORDER BY b.BillingDate DESC;

    IF @BillAmount IS NOT NULL
    BEGIN
        IF EXISTS (SELECT 1 FROM inserted
        WHERE PaymentID = @PaymentID AND Amount = @BillAmount)
            SET @IsValid = 1;
    END;

    RETURN @IsValid;
END;

-- Alter the Payments table to add a CHECK constraint
ALTER TABLE Budgeting.Payments
ADD CONSTRAINT CHK_ValidPaymentAmount
CHECK (dbo.ValidatePaymentAmount(PaymentID) = 1);
```

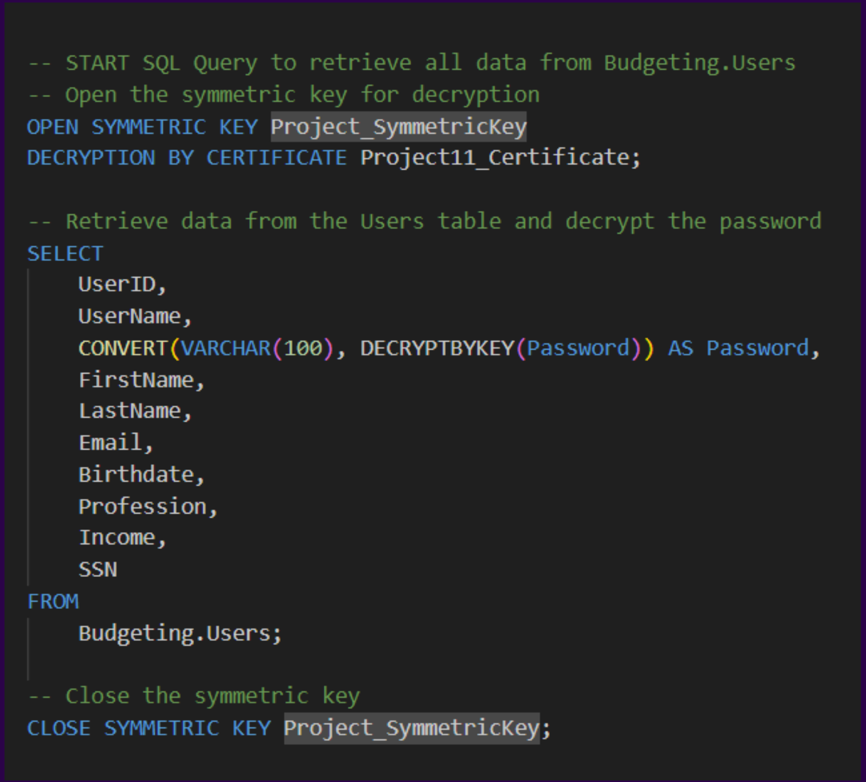
A table level check constraints that validates the amount that needs to be paid is exactly one Outstandings Bill Amount

Partial Amounts & Overhead amounts are not expected

COLUMN DATA ENCRYPTION

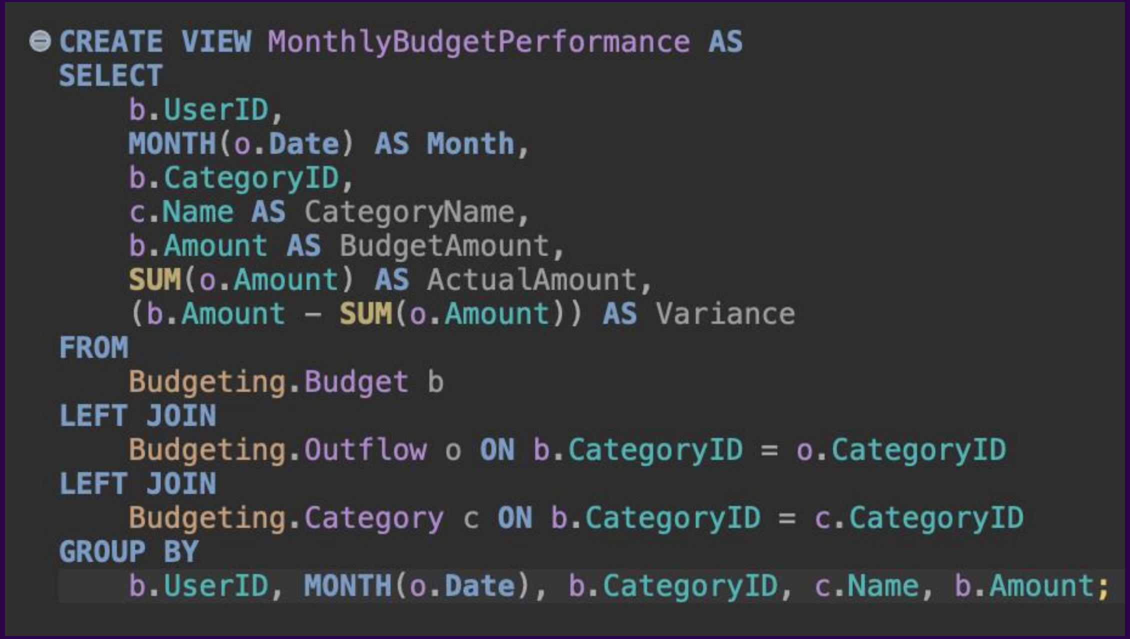


```sql
CREATE MASTER KEY ENCRYPTION BY
PASSWORD = 'Project@11';

-- Create certificate to protect symmetric key
CREATE CERTIFICATE Project11_Certificate
WITH SUBJECT = 'Project11 Test Certificate',
EXPIRY_DATE = '2024-12-31';

-- Create symmetric key to encrypt data
CREATE SYMMETRIC KEY Project_SymmetricKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE Project11_Certificate;


-- START Insert Data into Budgeting.Users
DECLARE @Password VARCHAR(100) = 'password@123';

IF dbo.ValidatePassword(@Password) = 1
BEGIN
    OPEN SYMMETRIC KEY Project_SymmetricKey
    DECRYPTION BY CERTIFICATE Project11_Certificate;

    DECLARE @EncryptedPassword VARBINARY(8000);

    SET @EncryptedPassword = ENCRYPTBYKEY(KEY_GUID('Project_SymmetricKey'), @Password)

    -- Now you can insert the encrypted password into the table
    INSERT INTO Budgeting.Users (UserName, Password, FirstName, LastName,
        Email, Birthdate, Profession, Income, SSN)
    VALUES ('JohnDoe', @EncryptedPassword, 'John', 'Doe',
        'john.doe@example.com', '1990-01-01', 'Software Engineer', 50000.00, '000-00-2222');

END
```



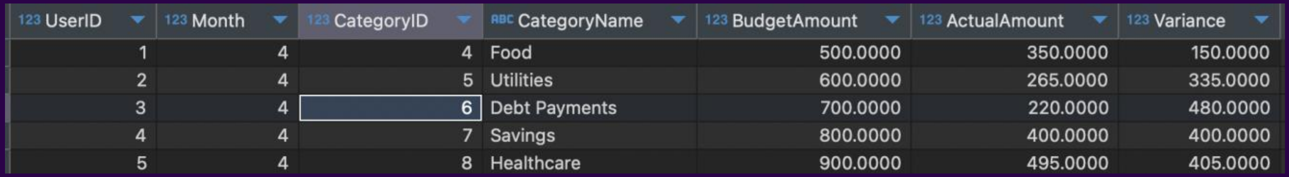
```sql
-- START SQL Query to retrieve all data from Budgeting.Users
-- Open the symmetric key for decryption
OPEN SYMMETRIC KEY Project_SymmetricKey
DECRYPTION BY CERTIFICATE Project11_Certificate;

-- Retrieve data from the Users table and decrypt the password
SELECT
    UserID,
    UserName,
    CONVERT(VARCHAR(100), DECRYPTBYKEY(Password)) AS Password,
    FirstName,
    LastName,
    Email,
    Birthdate,
    Profession,
    Income,
    SSN
FROM
    Budgeting.Users;

-- Close the symmetric key
CLOSE SYMMETRIC KEY Project_SymmetricKey;
```

MONTHLY BUDGET PERFORMANCE



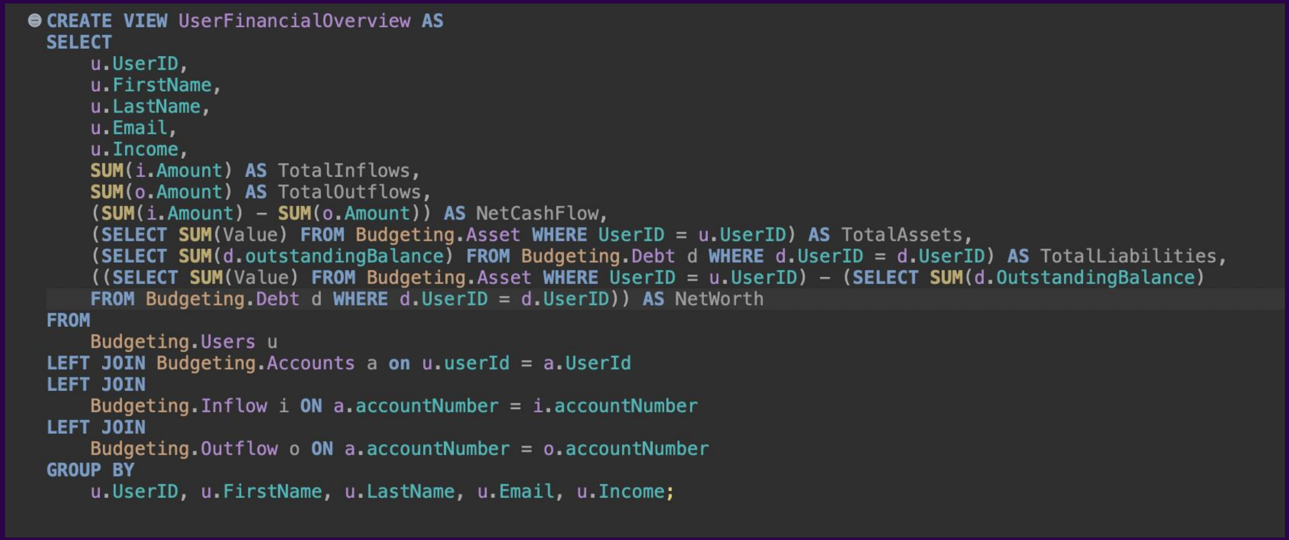
```
CREATE VIEW MonthlyBudgetPerformance AS
SELECT
    b.UserID,
    MONTH(o.Date) AS Month,
    b.CategoryID,
    c.Name AS CategoryName,
    b.Amount AS BudgetAmount,
    SUM(o.Amount) AS ActualAmount,
    (b.Amount - SUM(o.Amount)) AS Variance
FROM
    Budgeting.Budget b
LEFT JOIN
    Budgeting.Outflow o ON b.CategoryID = o.CategoryID
LEFT JOIN
    Budgeting.Category c ON b.CategoryID = c.CategoryID
GROUP BY
    b.UserID, MONTH(o.Date), b.CategoryID, c.Name, b.Amount;
```

Provides a view of each users spending for each category they have a budget set for and shows how much budget they have remaining within that category using the budget, category and outflow tables.

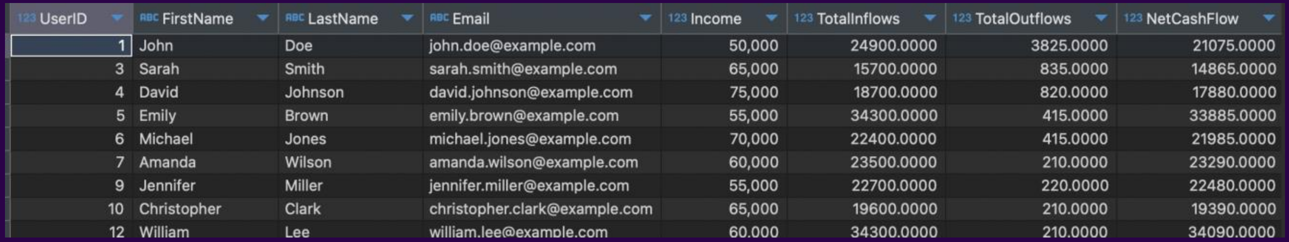


UserID	Month	CategoryID	CategoryName	BudgetAmount	ActualAmount	Variance
1	4	4	Food	500.0000	350.0000	150.0000
2	4	5	Utilities	600.0000	265.0000	335.0000
3	4	6	Debt Payments	700.0000	220.0000	480.0000
4	4	7	Savings	800.0000	400.0000	400.0000
5	4	8	Healthcare	900.0000	495.0000	405.0000

USER FINANCIAL OVERVIEW



```
CREATE VIEW UserFinancialOverview AS
SELECT
    u.UserID,
    u.FirstName,
    u.LastName,
    u.Email,
    u.Income,
    SUM(i.Amount) AS TotalInflows,
    SUM(o.Amount) AS TotalOutflows,
    (SUM(i.Amount) - SUM(o.Amount)) AS NetCashFlow,
    (SELECT SUM(Value) FROM Budgeting.Asset WHERE UserID = u.UserID) AS TotalAssets,
    (SELECT SUM(d.outstandingBalance) FROM Budgeting.Debt d WHERE d.UserID = d.UserID) AS TotalLiabilities,
    ((SELECT SUM(Value) FROM Budgeting.Asset WHERE UserID = u.UserID) - (SELECT SUM(d.OutstandingBalance)
    FROM Budgeting.Debt d WHERE d.UserID = d.UserID)) AS NetWorth
FROM
    Budgeting.Users u
LEFT JOIN Budgeting.Accounts a on u.userId = a.UserId
LEFT JOIN
    Budgeting.Inflow i ON a.accountNumber = i.accountNumber
LEFT JOIN
    Budgeting.Outflow o ON a.accountNumber = o.accountNumber
GROUP BY
    u.UserID, u.FirstName, u.LastName, u.Email, u.Income;
```

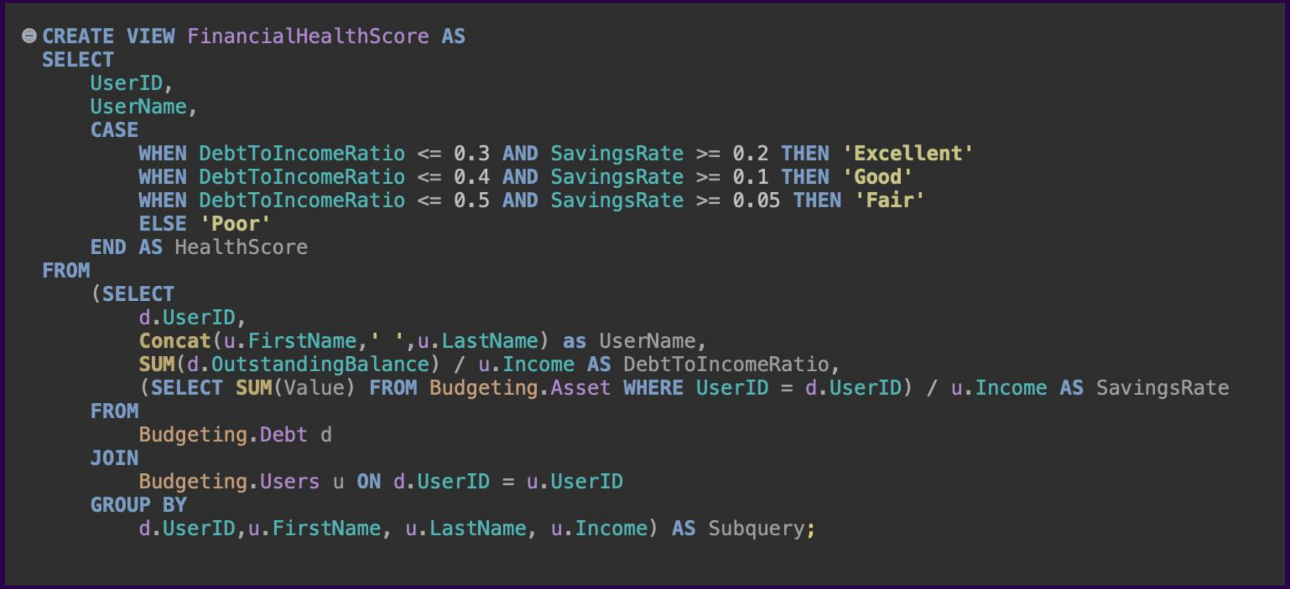


UserID	FirstName	LastName	Email	Income	TotalInflows	TotalOutflows	NetCashFlow
1	John	Doe	john.doe@example.com	50,000	24900.0000	3825.0000	21075.0000
3	Sarah	Smith	sarah.smith@example.com	65,000	15700.0000	835.0000	14865.0000
4	David	Johnson	david.johnson@example.com	75,000	18700.0000	820.0000	17880.0000
5	Emily	Brown	emily.brown@example.com	55,000	34300.0000	415.0000	33885.0000
6	Michael	Jones	michael.jones@example.com	70,000	22400.0000	415.0000	21985.0000
7	Amanda	Wilson	amanda.wilson@example.com	60,000	23500.0000	210.0000	23290.0000
9	Jennifer	Miller	jennifer.miller@example.com	55,000	22700.0000	220.0000	22480.0000
10	Christopher	Clark	christopher.clark@example.com	65,000	19600.0000	210.0000	19390.0000
12	William	Lee	william.lee@example.com	60,000	34300.0000	210.0000	34090.0000

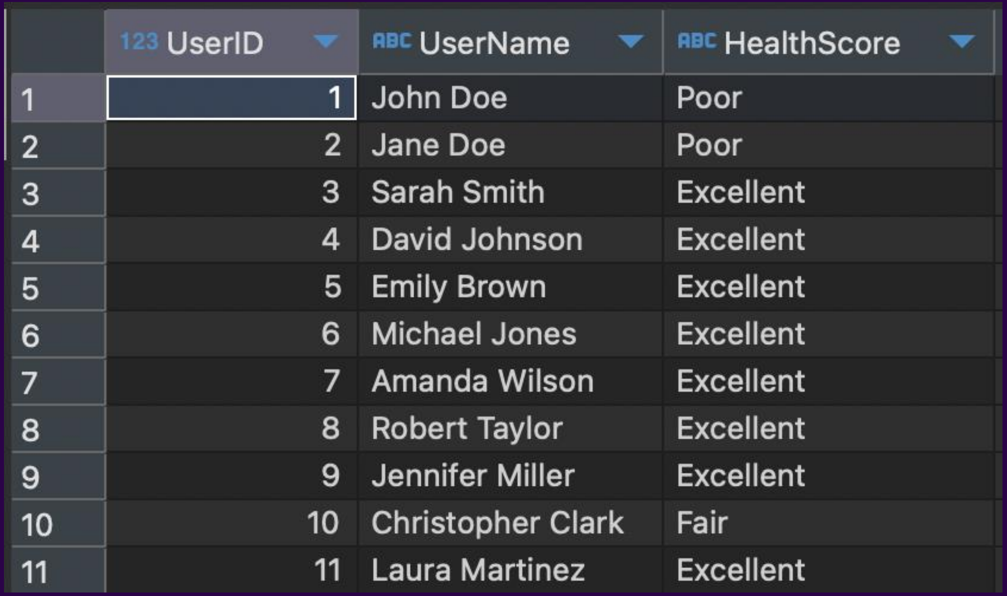
Provides a big picture view of a given user, calculates total inflow, outflow, net cash flow, total assets, liabilities and net worth all in one shot, using users, accounts, inflow, outflow, asset and debt tables.

FINANCIAL HEALTH SCORE

Utilizes a subquery to calculate the debt-to-income ratio for each user, as well as savings rate relative to income, leveraging debt, asset and user tables, and then assigns a financial health score based on the calculated financial metrics.

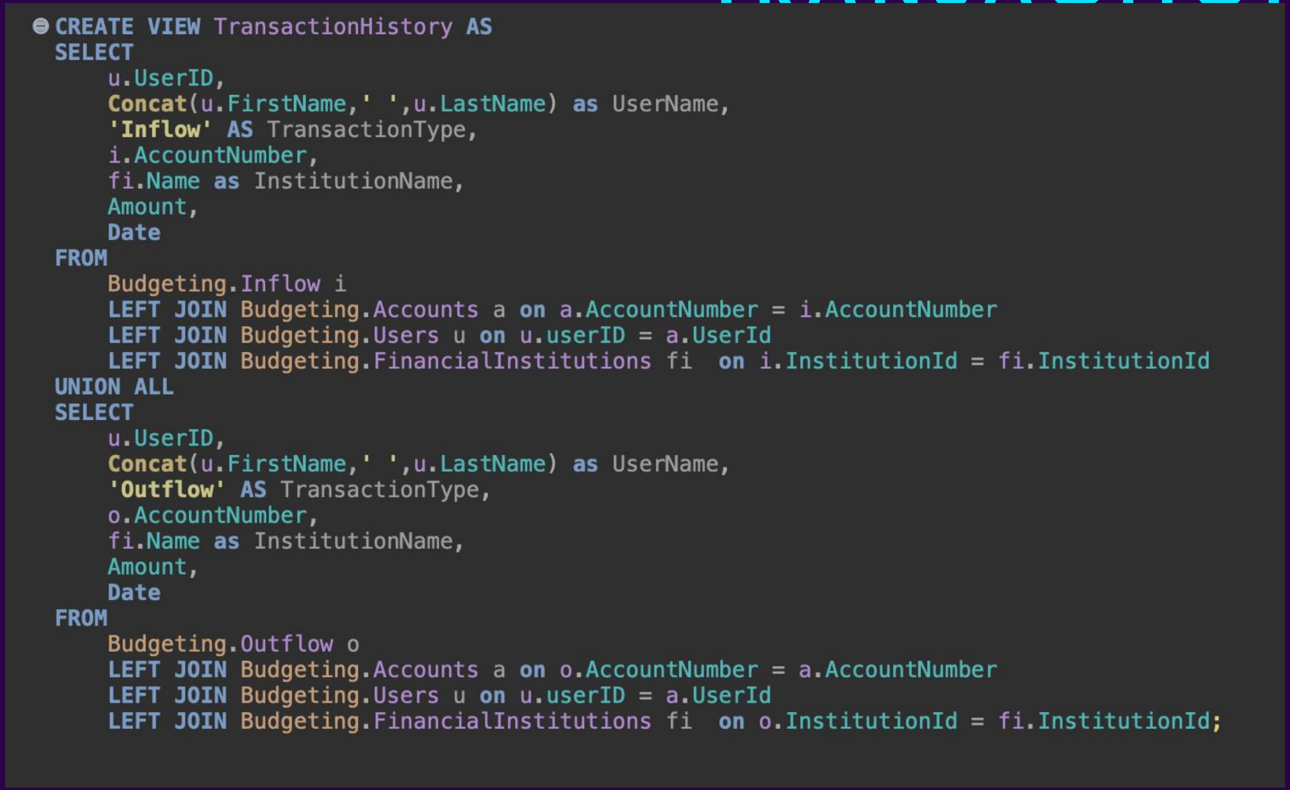


```sql
CREATE VIEW FinancialHealthScore AS
SELECT
    UserID,
    UserName,
    CASE
        WHEN DebtToIncomeRatio <= 0.3 AND SavingsRate >= 0.2 THEN 'Excellent'
        WHEN DebtToIncomeRatio <= 0.4 AND SavingsRate >= 0.1 THEN 'Good'
        WHEN DebtToIncomeRatio <= 0.5 AND SavingsRate >= 0.05 THEN 'Fair'
        ELSE 'Poor'
    END AS HealthScore
FROM
    (SELECT
        d.UserID,
        Concat(u.FirstName,' ',u.LastName) as UserName,
        SUM(d.OutstandingBalance) / u.Income AS DebtToIncomeRatio,
        (SELECT SUM(Value) FROM Budgeting.Asset WHERE UserID = d.UserID) / u.Income AS SavingsRate
    FROM
        Budgeting.Debt d
    JOIN
        Budgeting.Users u ON d.UserID = u.UserID
    GROUP BY
        d.UserID,u.FirstName, u.LastName, u.Income) AS Subquery;
```



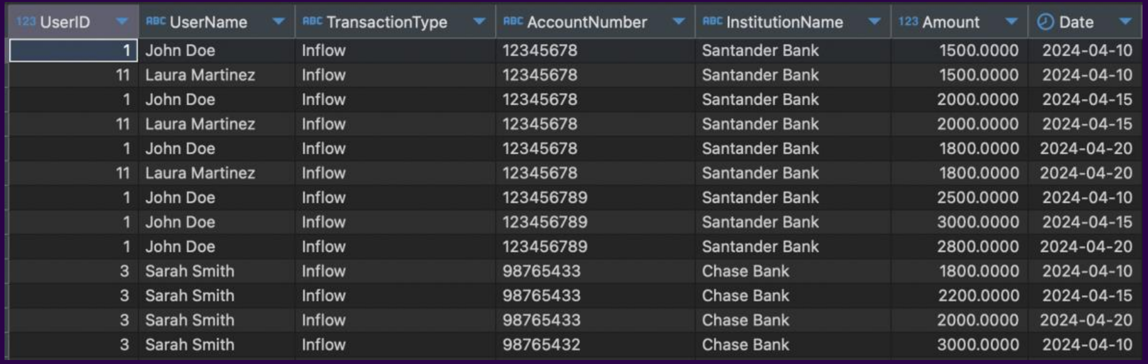
	UserID	UserName	HealthScore
1	1	John Doe	Poor
2	2	Jane Doe	Poor
3	3	Sarah Smith	Excellent
4	4	David Johnson	Excellent
5	5	Emily Brown	Excellent
6	6	Michael Jones	Excellent
7	7	Amanda Wilson	Excellent
8	8	Robert Taylor	Excellent
9	9	Jennifer Miller	Excellent
10	10	Christopher Clark	Fair
11	11	Laura Martinez	Excellent

TRANSACTION HISTORY

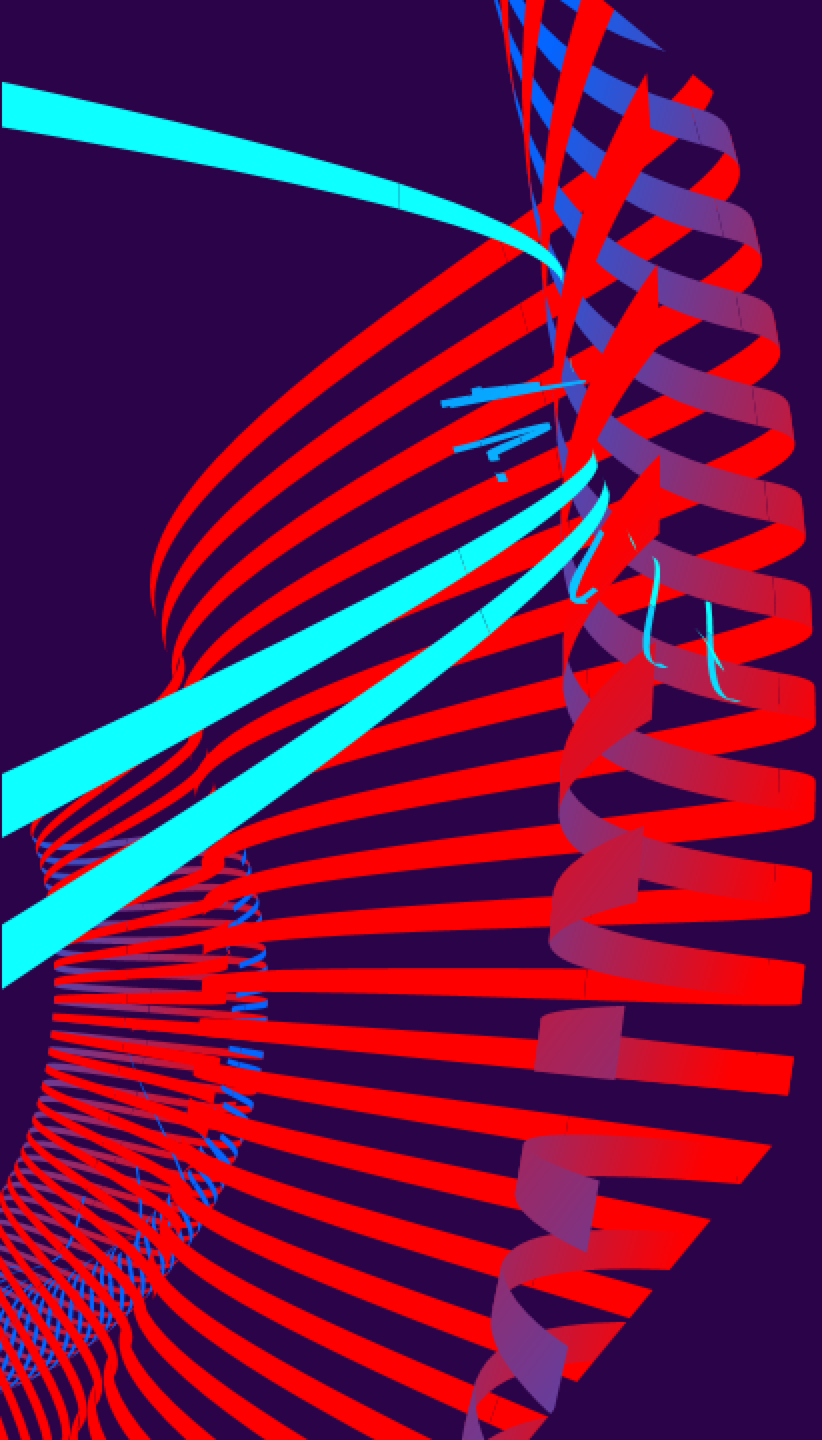


```sql
CREATE VIEW TransactionHistory AS
SELECT
    u.UserID,
    Concat(u.FirstName,' ',u.LastName) as UserName,
    'Inflow' AS TransactionType,
    i.AccountNumber,
    fi.Name as InstitutionName,
    Amount,
    Date
FROM
    Budgeting.Inflow i
    LEFT JOIN Budgeting.Accounts a on a.AccountNumber = i.AccountNumber
    LEFT JOIN Budgeting.Users u on u.userID = a.UserId
    LEFT JOIN Budgeting.FinancialInstitutions fi  on i.InstitutionId = fi.InstitutionId
UNION ALL
SELECT
    u.UserID,
    Concat(u.FirstName,' ',u.LastName) as UserName,
    'Outflow' AS TransactionType,
    o.AccountNumber,
    fi.Name as InstitutionName,
    Amount,
    Date
FROM
    Budgeting.Outflow o
    LEFT JOIN Budgeting.Accounts a on o.AccountNumber = a.AccountNumber
    LEFT JOIN Budgeting.Users u on u.userID = a.UserId
    LEFT JOIN Budgeting.FinancialInstitutions fi  on o.InstitutionId = fi.InstitutionId;
```

Provides a full picture view of user transactions, regardless of the type, similar to a cash flow statement. It leverages UNION ALL keyword to join the result set of two separate tables, with relevant data.

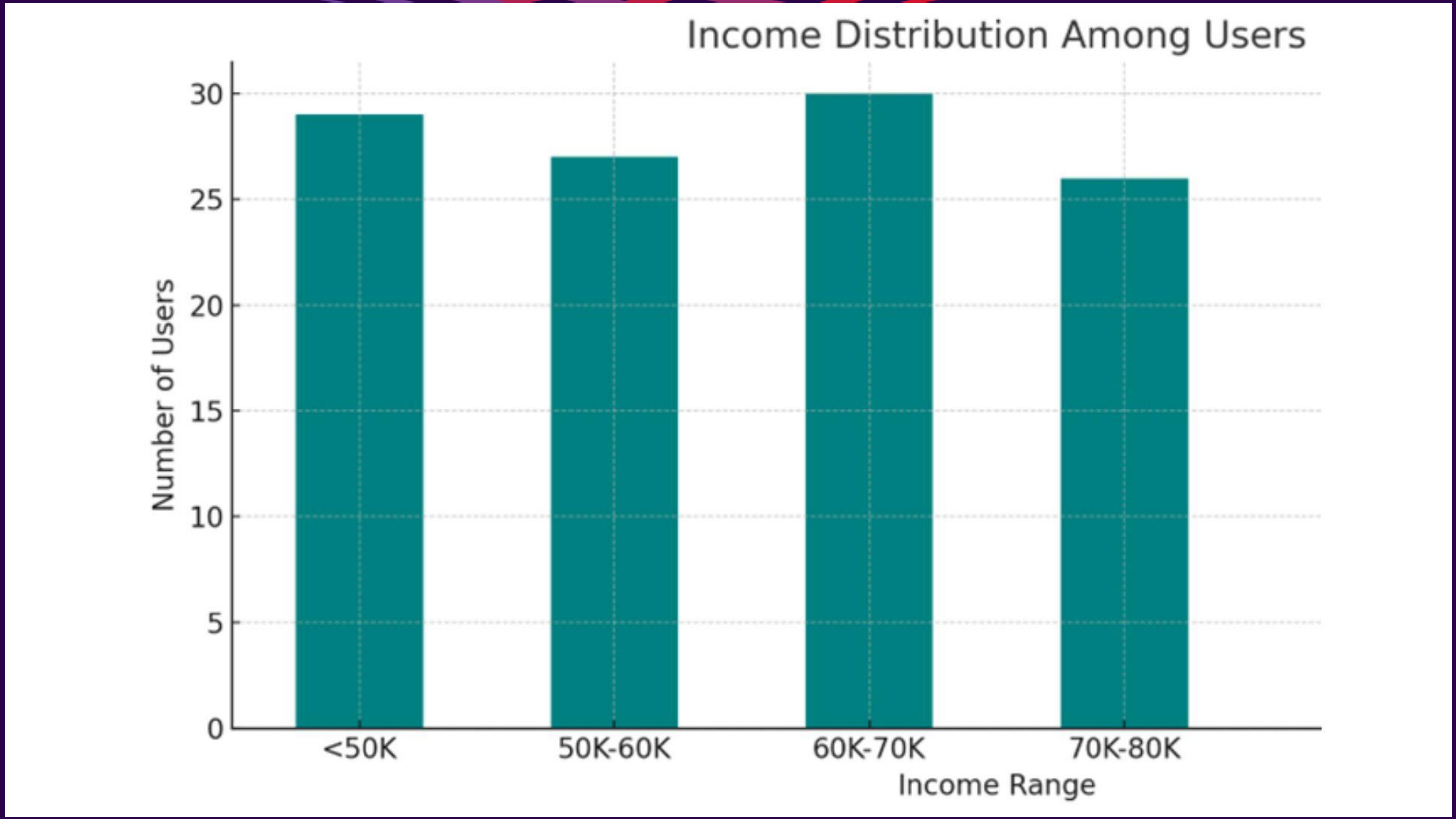


UserID	UserName	TransactionType	AccountNumber	InstitutionName	Amount	Date
1	John Doe	Inflow	12345678	Santander Bank	1500.0000	2024-04-10
11	Laura Martinez	Inflow	12345678	Santander Bank	1500.0000	2024-04-10
1	John Doe	Inflow	12345678	Santander Bank	2000.0000	2024-04-15
11	Laura Martinez	Inflow	12345678	Santander Bank	2000.0000	2024-04-15
1	John Doe	Inflow	12345678	Santander Bank	1800.0000	2024-04-20
11	Laura Martinez	Inflow	12345678	Santander Bank	1800.0000	2024-04-20
1	John Doe	Inflow	123456789	Santander Bank	2500.0000	2024-04-10
1	John Doe	Inflow	123456789	Santander Bank	3000.0000	2024-04-15
1	John Doe	Inflow	123456789	Santander Bank	2800.0000	2024-04-20
3	Sarah Smith	Inflow	98765433	Chase Bank	1800.0000	2024-04-10
3	Sarah Smith	Inflow	98765433	Chase Bank	2200.0000	2024-04-15
3	Sarah Smith	Inflow	98765433	Chase Bank	2000.0000	2024-04-20
3	Sarah Smith	Inflow	98765432	Chase Bank	3000.0000	2024-04-10

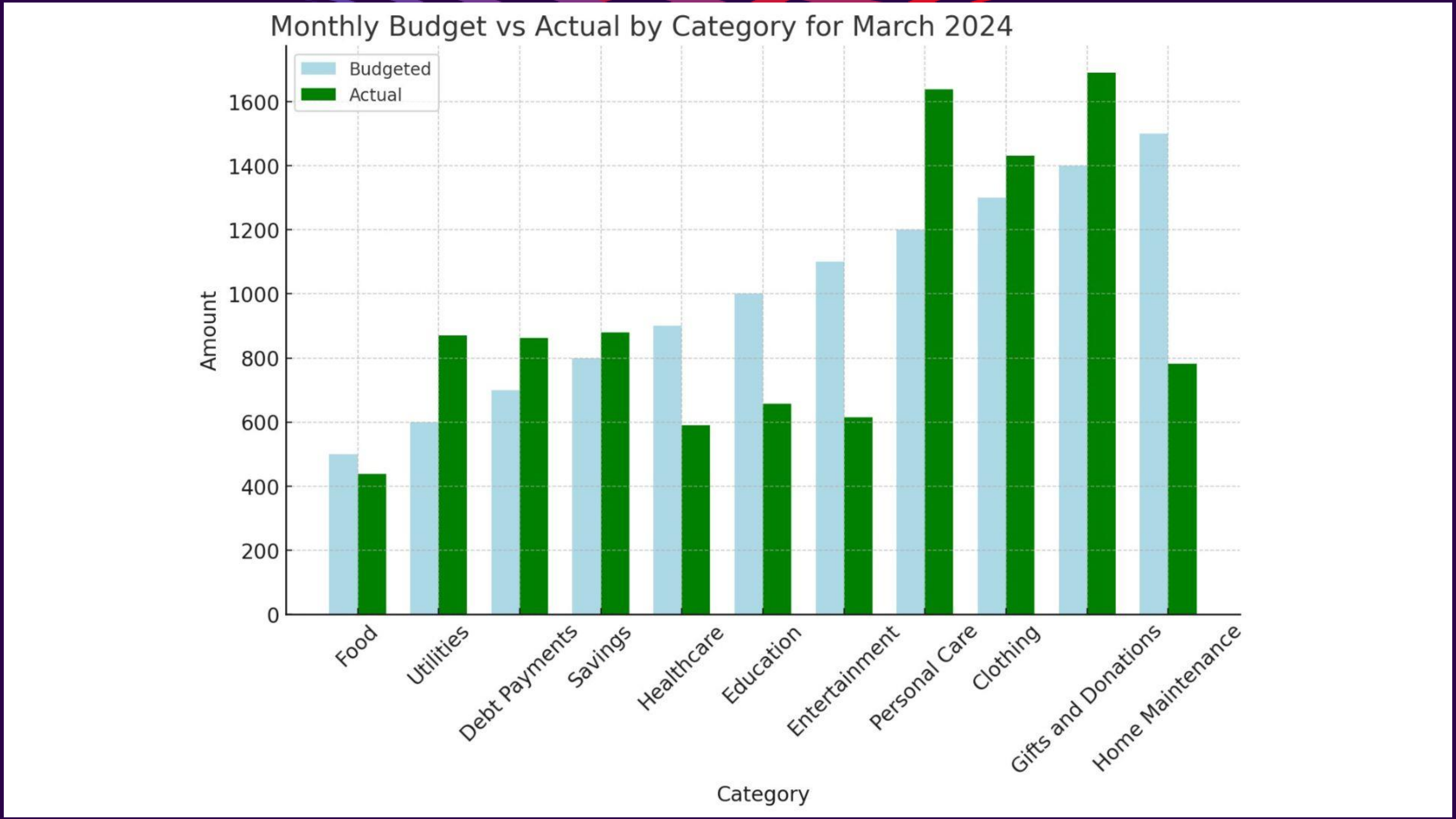


VISUALIZATIONS

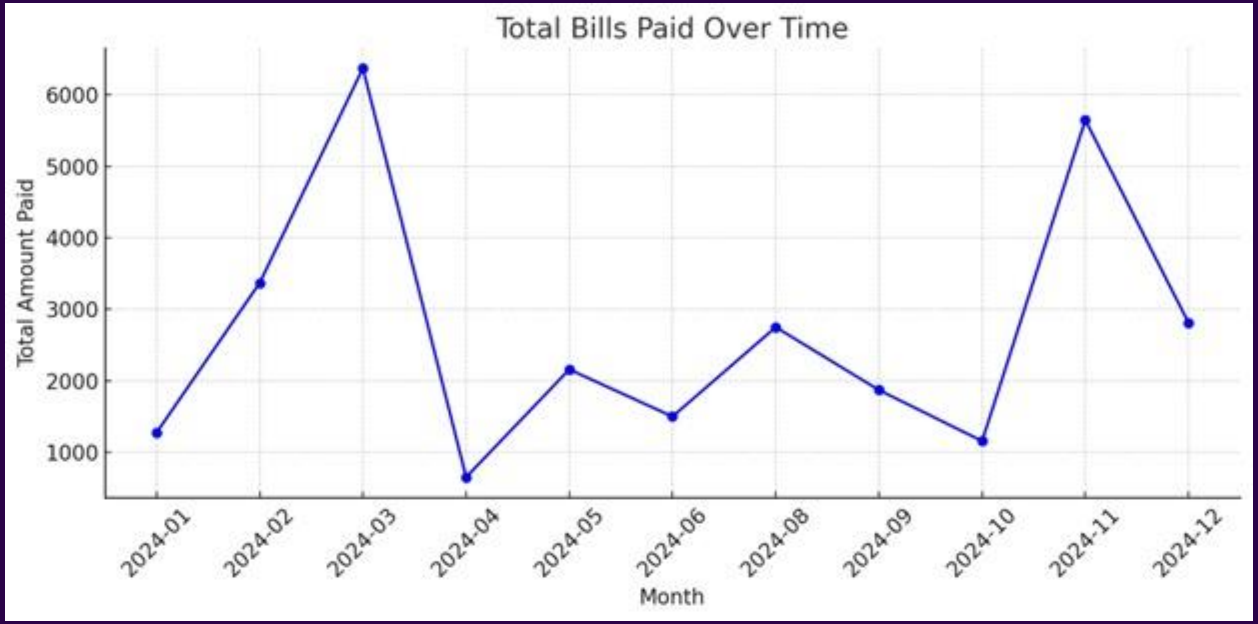
VISUALIZATIONS



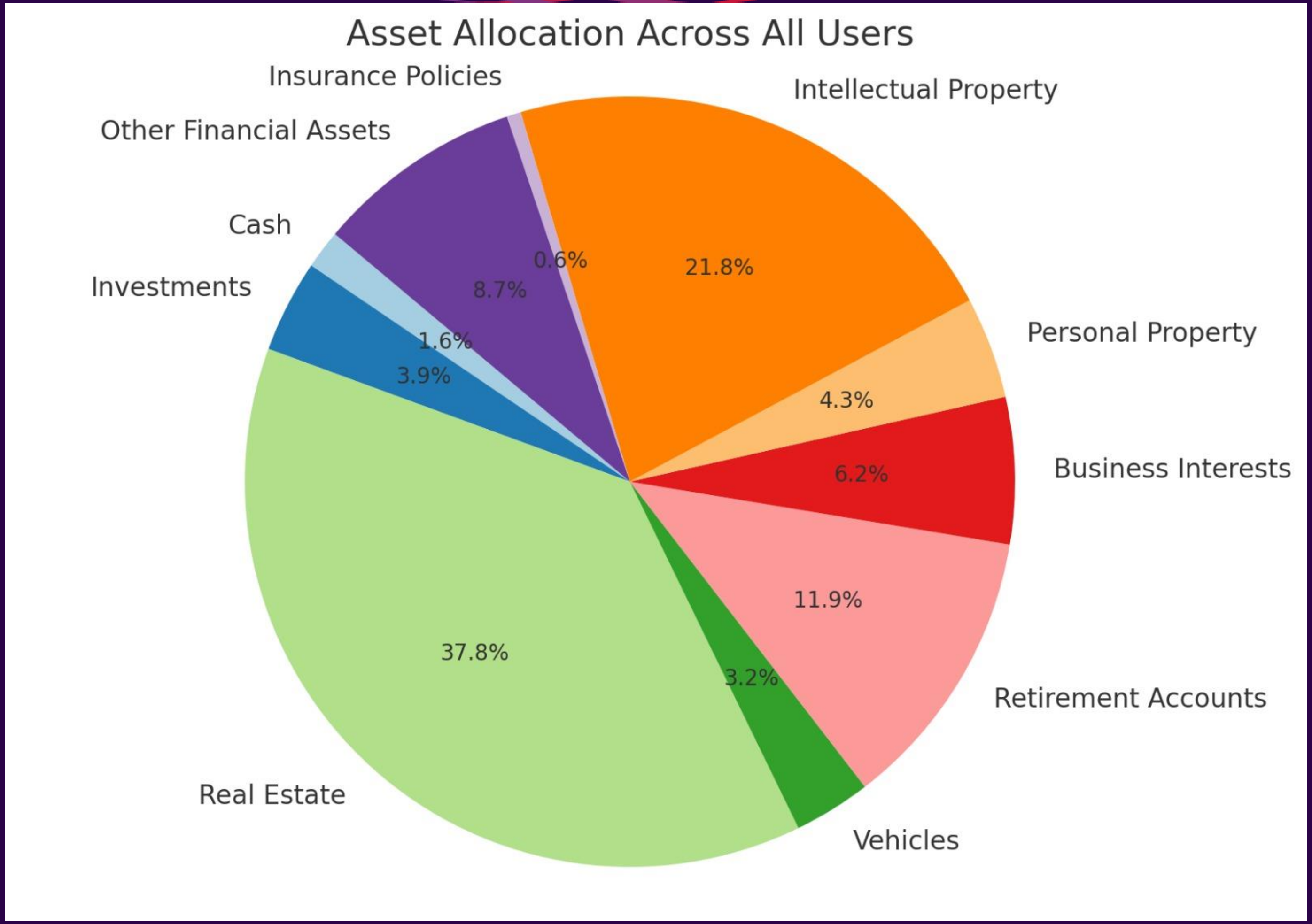
VISUALIZATIONS



VISUALIZATIONS



VISUALIZATIONS



SUMMARY

We strive to promote financial literacy and wellbeing amongst young adults, who are currently living paycheck to paycheck. We have architected a relational database to be able to scale and support a large number of users and their financial transactions, built checks and balances to support our business model, and constructed views to report on the raw data that we have collected. We believe that this business model will support our users in their path to financial wellbeing.

THANK YOU

Questions?