```
-------------------------------------------------------------------------------
--------------------------------
-------------------------------------------------------------------------------
--------------------------------
-------------------------------------------------------------------------------
--------------------------------
--  STORED PROCEDURE
-------------------------------------------------------------------------------
--------------------------------
-------------------------------------------------------------------------------
--------------------------------
-------------------------------------------------------------------------------
--------------------------------

--Transfer Money Between Accounts

CREATE PROCEDURE TransferMoney
    @SenderAccountID INT,
    @ReceiverAccountID INT,
    @Amount DECIMAL(18, 2)
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY
        -- Check if sender has sufficient balance
        IF (SELECT Balance FROM Accounts_tbl WHERE AccountID =
@SenderAccountID) < @Amount
        BEGIN
            THROW 50000, 'Insufficient funds in the sender account.', 1;
        END;

        -- Deduct amount from sender's account
        UPDATE Accounts_tbl
        SET Balance = Balance - @Amount
        WHERE AccountID = @SenderAccountID;

        -- Add amount to receiver's account
        UPDATE Accounts_tbl
        SET Balance = Balance + @Amount
        WHERE AccountID = @ReceiverAccountID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;

-- Top 5 Customers by Transactions

CREATE PROCEDURE TopCustomersByTransactions
    @StartDate DATE,
    @EndDate DATE
```

```
AS
BEGIN
    SELECT TOP 5
        c.CustomerID,
        c.CustomerName,
        SUM(t.Amount) AS TotalTransactionAmount
    FROM Transactions_tbl t
    INNER JOIN Customer_tbl c ON t.CustomerID = c.CustomerID
    WHERE t.TransactionDate BETWEEN @StartDate AND @EndDate
    GROUP BY c.CustomerID, c.CustomerName
    ORDER BY SUM(t.Amount) DESC;
END;


 -- Calculate Loan Interest

 CREATE PROCEDURE CalculateLoanInterest
    @LoanID INT = NULL,
    @Months INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY
        IF @LoanID IS NULL
        BEGIN
            -- Update interest for all loans
            UPDATE Loan_tbl
            SET InterestAmount = (LoanAmount * 0.10 / 12) * @Months
            WHERE LoanStatus = 'Active';
        END
        ELSE
        BEGIN
            -- Update interest for specific loan
            UPDATE Loan_tbl
            SET InterestAmount = (LoanAmount * 0.10 / 12) * @Months
            WHERE LoanID = @LoanID AND LoanStatus = 'Active';
        END;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;


--Log Failed Login Attempts

CREATE PROCEDURE LogFailedLogin
    @UserID INT,
    @IPAddress NVARCHAR(50),
    @Timestamp DATETIME
AS
```

```
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY
        -- Insert failed login attempt
        INSERT INTO Audit_tbl (UserID, IPAddress, AttemptTime, Action)
        VALUES (@UserID, @IPAddress, @Timestamp, 'Failed Login');

        -- Check if failed attempts exceed limit
        IF (
            SELECT COUNT(*)
            FROM Audit_tbl
            WHERE UserID = @UserID
              AND AttemptTime >= DATEADD(HOUR, -1, @Timestamp)
              AND Action = 'Failed Login'
        ) > 5
        BEGIN
            -- Lock user account
            UPDATE User_tbl
            SET AccountStatus = 'Locked'
            WHERE UserID = @UserID;
        END;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;



--Employee Performance

CREATE PROCEDURE TrackEmployeePerformance
    @EmployeeID INT,
    @Year INT
AS
BEGIN
    SELECT
        e.EmployeeID,
        e.EmployeeName,
        COUNT(t.TransactionID) AS TotalTransactions,
        SUM(l.LoanAmount) AS TotalLoansApproved
    FROM Employee_tbl e
    LEFT JOIN Transactions_tbl t ON e.EmployeeID = t.EmployeeID
    LEFT JOIN Loan_tbl l ON e.EmployeeID = l.ApprovedBy
    WHERE e.EmployeeID = @EmployeeID
      AND YEAR(t.TransactionDate) = @Year
    GROUP BY e.EmployeeID, e.EmployeeName;
END;


--Grant or Revoke Permissions
```

```
CREATE PROCEDURE ManagePermissions
    @UserID INT,
    @RoleID INT,
    @PermissionID INT,
    @Action NVARCHAR(10) -- 'Grant' or 'Revoke'
AS
BEGIN
    IF @Action = 'Grant'
    BEGIN
        INSERT INTO Role_Permission_tbl (RoleID, PermissionID, UserID)
        VALUES (@RoleID, @PermissionID, @UserID);
    END
    ELSE IF @Action = 'Revoke'
    BEGIN
        DELETE FROM Role_Permission_tbl
        WHERE RoleID = @RoleID AND PermissionID = @PermissionID AND UserID
= @UserID;
    END
    ELSE
    BEGIN
        PRINT 'Invalid action specified.';
    END;
END;



-------------------------------------------------------------------------
--------------------------------
-------------------------------------------------------------------------
--------------------------------
-------------------------------------------------------------------------
--------------------------------
--  Trigger
-------------------------------------------------------------------------
--------------------------------
-------------------------------------------------------------------------
--------------------------------

--Prevent Overdrafts on Withdrawals

CREATE TRIGGER PreventOverdraft
ON Transactions_tbl
AFTER INSERT
AS
BEGIN
    -- Check if any withdrawal causes an account balance to fall below
zero
    IF EXISTS (
        SELECT 1
        FROM Accounts_tbl a
        INNER JOIN Inserted i ON a.AccountID = i.AccountID
        WHERE i.TransactionType = 'Withdrawal'
```

```
          AND (a.Balance - i.Amount) < 0
    )
    BEGIN
        -- Rollback the transaction
        ROLLBACK TRANSACTION;
        THROW 50000, 'Withdrawal denied: Insufficient funds.', 1;
    END;

    -- Update the account balance for valid withdrawals
    UPDATE Accounts_tbl
    SET Balance = Balance - i.Amount
    FROM Accounts_tbl a
    INNER JOIN Inserted i ON a.AccountID = i.AccountID
    WHERE i.TransactionType = 'Withdrawal';
END;


--Automatically Update Loan Status
CREATE TRIGGER UpdateLoanStatus
ON Transactions_tbl
AFTER INSERT
AS
BEGIN
    -- Check if the loan is fully repaid
    UPDATE Loan_tbl
    SET LoanStatus = 'Closed'
    FROM Loan_tbl l
    INNER JOIN Inserted i ON l.LoanID = i.LoanID
    WHERE l.LoanStatus = 'Active'
      AND l.LoanAmount <= (
          SELECT SUM(t.Amount)
          FROM Transactions_tbl t
          WHERE t.LoanID = l.LoanID
      );
END;


--Log Failed Login Attempts

CREATE TRIGGER LogFailedLogin
ON User_tbl
AFTER UPDATE
AS
BEGIN
    -- Log failed login attempts
    INSERT INTO Audit_tbl (UserID, IPAddress, AttemptTime, Action)
    SELECT i.UserID, i.LastLoginIP, GETDATE(), 'Failed Login'
    FROM Inserted i
    INNER JOIN Deleted d ON i.UserID = d.UserID
    WHERE i.AccountStatus = 'Locked'
      AND d.AccountStatus != 'Locked';
END;

CREATE PROCEDURE sp_UpdateUserAccountStatus
```

```
    @UserID INT,
    @AccountStatus VARCHAR(20),
    @LastLoginIP VARCHAR(50)
AS
BEGIN
    UPDATE User_tbl
    SET AccountStatus = @AccountStatus,
        LastLoginIP = @LastLoginIP
    WHERE UserID = @UserID;
END;

-------------------------------------------------------------------------------
----------------------------------------------
-------------------------------------------------------------------------------
----------------------------------------------

--Banking Database Schema

-------------------------------------------------------------------------------
----------------------------------------------
-------------------------------------------------------------------------------
----------------------------------------------


CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(255),
    Address VARCHAR(255),
    Phone VARCHAR(20)
);

CREATE TABLE Accounts (
    AccountID INT PRIMARY KEY,
    CustomerID INT,
    AccountType VARCHAR(20),
    Balance DECIMAL(18, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

CREATE TABLE Transactions (
    TransactionID INT PRIMARY KEY IDENTITY(1,1),
    AccountID INT,
    TransactionType VARCHAR(20),
    Amount DECIMAL(18, 2),
    TransactionDate DATETIME DEFAULT GETDATE(),
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);

CREATE TABLE AuditLog (
    AuditLogID INT PRIMARY KEY IDENTITY(1,1),
    AccountID INT,
    TransactionType VARCHAR(20),
    Amount DECIMAL(18, 2),
    TransactionDate DATETIME,
```

```
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);
```

```
CREATE PROCEDURE sp_CreateAccount
    @CustomerID INT,
    @AccountType VARCHAR(20),
    @InitialBalance DECIMAL(18, 2)
AS
BEGIN
    INSERT INTO Accounts (CustomerID, AccountType, Balance)
    VALUES (@CustomerID, @AccountType, @InitialBalance);
END;

CREATE PROCEDURE sp_Deposit
    @AccountID INT,
    @Amount DECIMAL(18, 2)
AS
BEGIN
    UPDATE Accounts
    SET Balance = Balance + @Amount
    WHERE AccountID = @AccountID;
END;

CREATE PROCEDURE sp_Withdraw
    @AccountID INT,
    @Amount DECIMAL(18, 2)
AS
BEGIN
    IF (SELECT Balance FROM Accounts WHERE AccountID = @AccountID) >=
@Amount
    BEGIN
        UPDATE Accounts
        SET Balance = Balance - @Amount
        WHERE AccountID = @AccountID;
    END
    ELSE
    BEGIN
        RAISERROR ('Insufficient funds.', 16, 1);
    END
END;

CREATE PROCEDURE sp_Transfer
    @FromAccountID INT,
    @ToAccountID INT,
    @Amount DECIMAL(18, 2)
AS
BEGIN
```

```
    BEGIN TRANSACTION;
    BEGIN TRY
        EXEC sp_Withdraw @FromAccountID, @Amount;
        EXEC sp_Deposit @ToAccountID, @Amount;
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        DECLARE @ErrorMessage NVARCHAR(4000);
        SET @ErrorMessage = ERROR_MESSAGE();
        RAISERROR (@ErrorMessage, 16, 1);
    END CATCH
END;


CREATE TRIGGER tr_AuditLog
ON Accounts
AFTER UPDATE
AS
BEGIN
    IF UPDATE(Balance)
    BEGIN
        INSERT INTO AuditLog (AccountID, TransactionType, Amount,
TransactionDate)
        SELECT i.AccountID,
               CASE
                   WHEN i.Balance > d.Balance THEN 'Deposit'
                   ELSE 'Withdrawal'
               END,
               ABS(i.Balance - d.Balance),
               GETDATE()
        FROM Inserted i
        INNER JOIN Deleted d ON i.AccountID = d.AccountID
        WHERE i.Balance != d.Balance;
    END
END;
```


-- UDF


```
--------------------------------------------------------------------------------
------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------------
```

```
CREATE FUNCTION FunctionName (@Parameter1 DataType, @Parameter2 DataType)
RETURNS ReturnType
AS
BEGIN
    -- Function body
END;
```

```
CREATE FUNCTION GetCustomerName (@CustomerID INT)
RETURNS VARCHAR(255)
AS
BEGIN
    DECLARE @Name VARCHAR(255);
    SELECT @Name = Name FROM Customers WHERE CustomerID = @CustomerID;
    RETURN @Name;
END;
```

```
CREATE FUNCTION GetCustomerOrders (@CustomerID INT)
RETURNS TABLE
AS
RETURN (
    SELECT * FROM Orders WHERE CustomerID = @CustomerID
);
```

```
CREATE FUNCTION TotalBalance ()
RETURNS DECIMAL(18, 2)
AS
BEGIN
```

```
    DECLARE @TotalBalance DECIMAL(18, 2);
    SELECT @TotalBalance = SUM(Balance) FROM Accounts;
    RETURN @TotalBalance;
END;

SELECT dbo.TotalBalance() AS TotalBalance;



-------------------------------------------------------------------------------
----------------------------------------------
-------------------------------------------------------------------------------
----------------------------------------------
-------------------------------------------------------------------------------
----------------------------------------------
-------------------------------------------------------------------------------
----------------------------------------------
-- BUILT IN FUNCTIONS
-------------------------------------------------------------------------------
----------------------------------------------
-------------------------------------------------------------------------------
----------------------------------------------
-------------------------------------------------------------------------------
----------------------------------------------
-------------------------------------------------------------------------------
----------------------------------------------



--STRING

-- LEN() Function
SELECT LEN('Hello World')  -- returns 11

-- UPPER() Function
SELECT UPPER('hello world')  -- returns 'HELLO WORLD'

-- LOWER() Function
SELECT LOWER('HELLO WORLD')  -- returns 'hello world'

-- LTRIM() Function
SELECT LTRIM('   hello world')  -- returns 'hello world'

-- RTRIM() Function
SELECT RTRIM('hello world   ')  -- returns 'hello world'

-- REPLACE() Function
SELECT REPLACE('hello world', 'world', 'universe')  -- returns 'hello
universe'

-- SUBSTRING() Function
SELECT SUBSTRING('hello world', 7, 5)  -- returns 'world'
```

```
--MATH



-- ABS() Function
SELECT ABS(-10)    -- returns 10
SELECT ABS(20)     -- returns 20
SELECT ABS(-30.5)  -- returns 30.5

-- CEILING() Function
SELECT CEILING(10.2)   -- returns 11
SELECT CEILING(20)     -- returns 20
SELECT CEILING(30.7)   -- returns 31

-- FLOOR() Function
SELECT FLOOR(10.2)   -- returns 10
SELECT FLOOR(20)     -- returns 20
SELECT FLOOR(30.7)   -- returns 30

-- ROUND() Function
SELECT ROUND(10.234, 2)  -- returns 10.23
SELECT ROUND(20, 0)      -- returns 20
SELECT ROUND(30.789, 1)  -- returns 30.8

-- SQRT() Function
SELECT SQRT(16)   -- returns 4
SELECT SQRT(25)   -- returns 5
SELECT SQRT(36)   -- returns 6




-- GETDATE() Function
SELECT GETDATE()  -- returns the current date and time

-- DATEADD() Function
SELECT DATEADD(day, 10, '2022-01-01')  -- adds 10 days to the specified
date
SELECT DATEADD(month, 5, '2022-01-01')  -- adds 5 months to the specified
date
SELECT DATEADD(year, 2, '2022-01-01')  -- adds 2 years to the specified
date

-- DATEDIFF() Function
SELECT DATEDIFF(day, '2022-01-01', '2022-01-15')  -- returns the
difference in days
SELECT DATEDIFF(month, '2022-01-01', '2022-06-01')  -- returns the
difference in months
SELECT DATEDIFF(year, '2022-01-01', '2025-01-01')  -- returns the
difference in years
```

```
-- DAY() Function
SELECT DAY('2022-07-25')  -- returns the day of the month (25)

-- MONTH() Function
SELECT MONTH('2022-07-25')  -- returns the month (7)

-- YEAR() Function
SELECT YEAR('2022-07-25')  -- returns the year (2022)



--CONVERSION



-- CAST() Function
SELECT CAST('123' AS INT)  -- converts string to integer
SELECT CAST(123 AS VARCHAR(10))  -- converts integer to string
SELECT CAST(GETDATE() AS DATE)  -- converts datetime to date

-- CONVERT() Function
SELECT CONVERT(INT, '123')  -- converts string to integer
SELECT CONVERT(VARCHAR(10), 123)  -- converts integer to string
SELECT CONVERT(DATE, GETDATE())  -- converts datetime to date
SELECT CONVERT(VARCHAR(10), GETDATE(), 103)  -- converts datetime to
string in specific format




--------------------------------------------------------------------------
----------------------------------------------
```