```
-- ==========================================
-- SETUP TABLES FOR ASSIGNMENT
-- ==========================================

CREATE TABLE AccountBalance (
    AccountId INT PRIMARY KEY,
    AccountName VARCHAR(100),
    Balance DECIMAL(18,2) CHECK (Balance >= 0),
    LastUpdated DATETIME DEFAULT GETDATE()
);
GO

CREATE TABLE TransferHistory (
    TransferId INT IDENTITY(1,1) PRIMARY KEY,
    FromAccountId INT,
    ToAccountId INT,
    Amount DECIMAL(18,2),
    TransferDate DATETIME DEFAULT GETDATE(),
    Status VARCHAR(20),
    ErrorMessage VARCHAR(500)
);
GO
CREATE TABLE AuditTrail (
    AuditId INT IDENTITY(1,1) PRIMARY KEY,
    TableName VARCHAR(100),
    Operation VARCHAR(50),
    RecordId INT,
    OldValue VARCHAR(500),
    NewValue VARCHAR(500),
    AuditDate DATETIME DEFAULT GETDATE(),
    UserName VARCHAR(100) DEFAULT SYSTEM_USER
```

```
);
GO

-- Insert sample data
INSERT INTO AccountBalance (AccountId, AccountName, Balance)
VALUES
    (101, 'Checking Account', 10000.00),
    (102, 'Savings Account', 25000.00),
    (103, 'Investment Account', 50000.00),
    (104, 'Emergency Fund', 15000.00);
GO
```

Question 01 :

Write a simple transaction that transfers $500 from Account 101 to Account 102.
Use BEGIN TRANSACTION and COMMIT TRANSACTION.
Display the balances before and after the transfer.

Question 02 :

Write a transaction that attempts to transfer $1000 from Account 101 to Account 102, but then rolls it back using ROLLBACK TRANSACTION.
Verify that the balances remain unchanged..

Question 03 :

Write a transaction that checks if Account 101 has sufficient balance before transferring $2000 to Account 102.
If insufficient, rollback the transaction.
If sufficient, commit the transaction.

Question 04 :

Write a transaction using TRY...CATCH that transfers money from Account 101 to Account 102. If any error occurs, rollback the transaction and display the error message.

Question 05 :

Write a transaction that uses SAVE TRANSACTION to create a savepoint after the first update. Then perform a second update and rollback to the savepoint if an error occurs.

Question 06 :

Write a transaction with nested BEGIN TRANSACTION statements.
Display @@TRANCOUNT at each level to demonstrate how it changes.

Question 07 :

Demonstrate ATOMICITY by writing a transaction that performs multiple updates.
Show that if one fails, all are rolled back.

Question 08 :

Demonstrate CONSISTENCY by writing a transaction that ensures the total balance across all accounts remains constant.
Calculate total before and after transfer.

Question 09 :

Demonstrate ISOLATION by setting different isolation levels and explaining their effects. Use READ UNCOMMITTED, READ COMMITTED, and SERIALIZABLE.

Question 10 :

Demonstrate DURABILITY by committing a transaction and explaining that the changes will persist even after system restart or failure.

Question 11 :

Write a stored procedure that uses transactions to transfer
- money between two accounts. Include parameter validation,
- error handling, and proper transaction management.

Question 12 :

Write a transaction that uses multiple savepoints to handle
- a multi-step operation. If step 2 fails, rollback to savepoint 1.
- If step 3 fails, rollback to savepoint 2.

QUESTION 13 :

- Write a transaction that handles a deadlock scenario using
- TRY...CATCH. Retry the operation if a deadlock is detected.

QUESTION 14 :

Write a query to check the current transaction count (@@TRANCOUNT)
and demonstrate how it changes within nested transactions.

QUESTION 15 :

Write a transaction that logs all changes to the AuditTrail table.
Include before and after values for updates.

QUESTION 16 :

Write a transaction that demonstrates the difference between COMMIT and ROLLBACK by creating two identical transactions, committing one and rolling back the other.

QUESTION 17 :

Write a transaction that enforces a business rule: "Total withdrawals in a single transaction cannot exceed $5000".
If violated, rollback the transaction.

QUESTION 18 :

Write a transaction that uses explicit locking hints (WITH (UPDLOCK)) to prevent concurrent modifications during a transfer.

QUESTION 19 :

Write a comprehensive error handling transaction that catches specific error numbers and handles them differently.
Handle: Constraint violations, insufficient funds, and general errors.

QUESTION 20:

Write a transaction monitoring query that shows all active transactions in the database, including their status, start time, and session information.