Assignment 3: Explain the ACID properties of a transaction in your own words. Write SQL statements to simulate a transaction that includes locking and demonstrate different isolation levels to show concurrency control.

ACID Properties of a Transaction

ACID properties ensure reliability and integrity in a database transaction:

1. **Atomicity:** A transaction is either fully completed or fully rolled back if any part of it fails.

   Example: Transferring money between two accounts should either update both balances or none.

2. **Consistency:** The database should remain valid before and after the transaction, following defined rules and constraints.

   Example: If money is withdrawn from one account, it must be credited to another, maintaining balance constraints.

3. **Isolation**: Transactions run independently, preventing one transaction from interfering with another. Different isolation levels help manage concurrency.

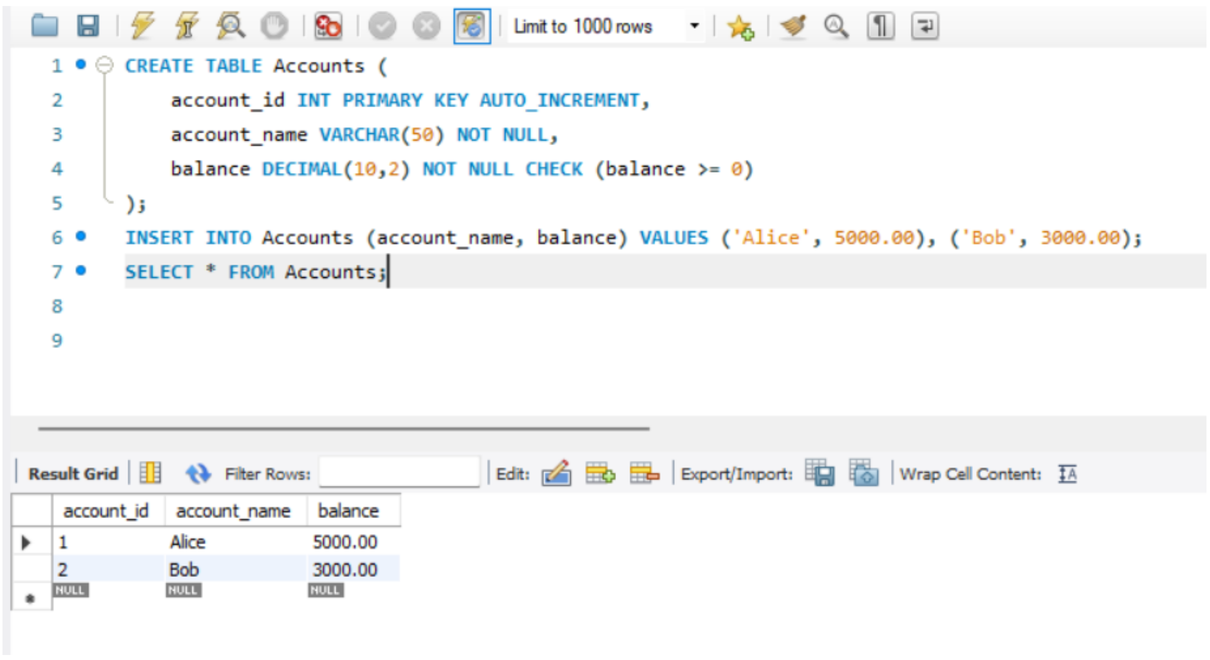
   Example: If two users try to withdraw money from the same account, isolation ensures correct processing.

4. **Durability:** Once a transaction is committed, changes are permanently saved, even in case of system failure.

   Example: A successful money transfer is not lost even if the database crashes immediately after.

Simulating a Transaction with Locking.

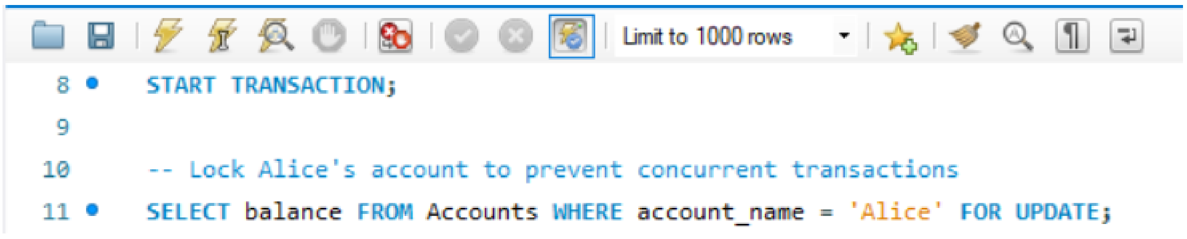
Step 1: Create Tables for a Banking System & Insert Sample Data.



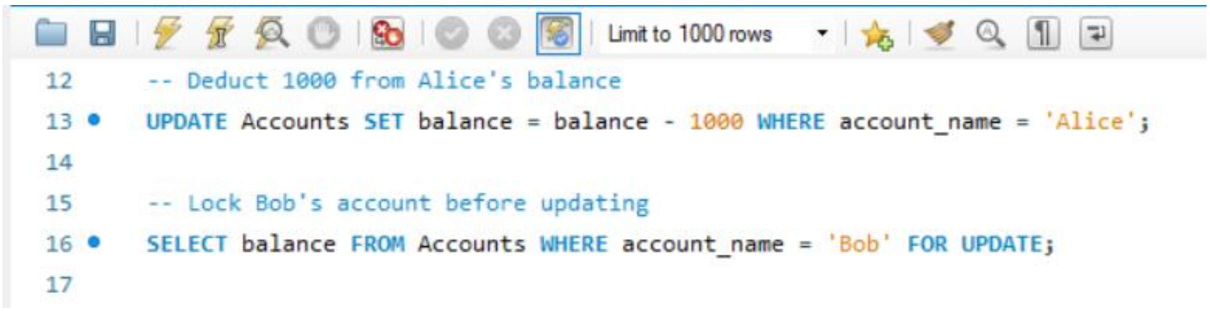
```sql
CREATE TABLE Accounts (
    account_id INT PRIMARY KEY AUTO_INCREMENT,
    account_name VARCHAR(50) NOT NULL,
    balance DECIMAL(10,2) NOT NULL CHECK (balance >= 0)
);
INSERT INTO Accounts (account_name, balance) VALUES ('Alice', 5000.00), ('Bob', 3000.00);
SELECT * FROM Accounts;
```

account_id	account_name	balance
1	Alice	5000.00
2	Bob	3000.00
NULL	NULL	NULL

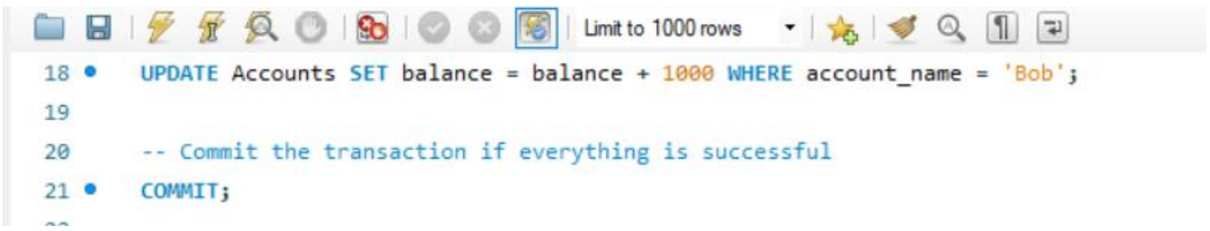
Step 2 : Perform a Transaction with Locking.



```sql
START TRANSACTION;

-- Lock Alice's account to prevent concurrent transactions
SELECT balance FROM Accounts WHERE account_name = 'Alice' FOR UPDATE;
```



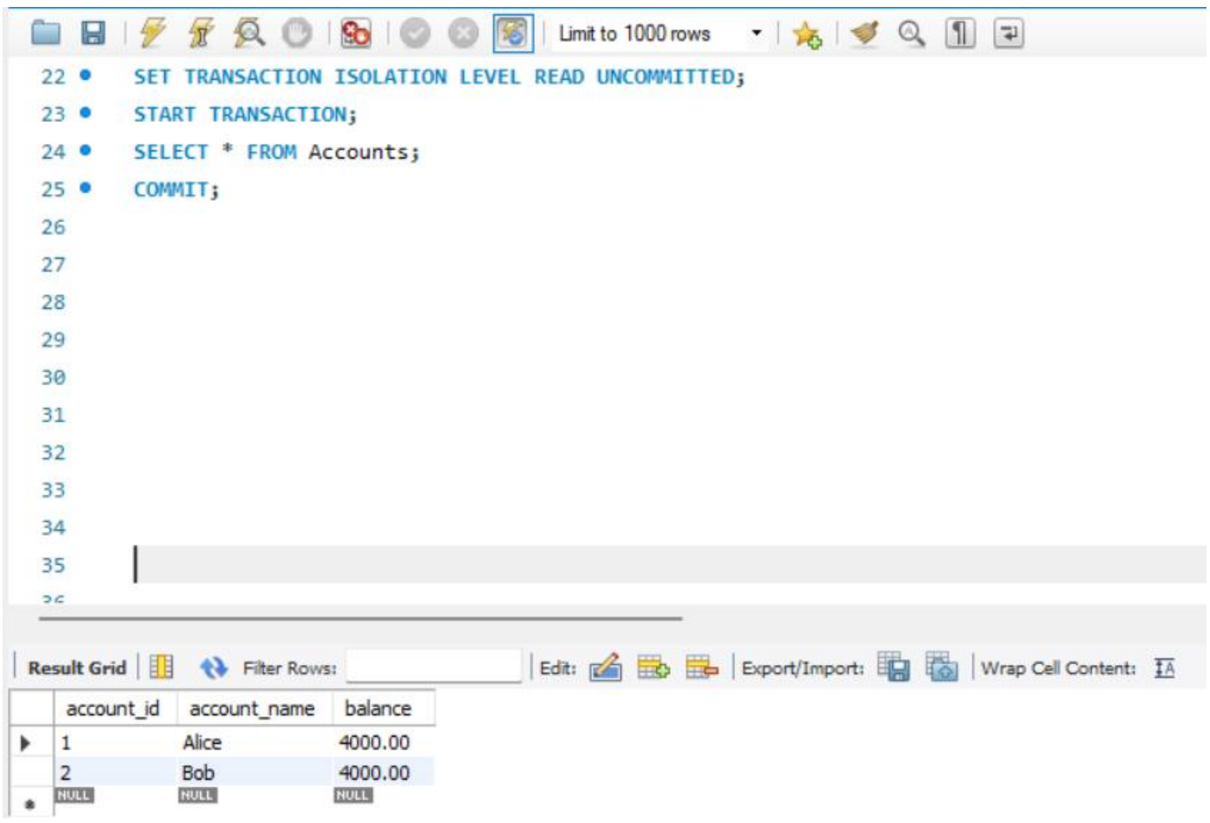
```sql
-- Deduct 1000 from Alice's balance
UPDATE Accounts SET balance = balance - 1000 WHERE account_name = 'Alice';

-- Lock Bob's account before updating
SELECT balance FROM Accounts WHERE account_name = 'Bob' FOR UPDATE;

```



```sql
UPDATE Accounts SET balance = balance + 1000 WHERE account_name = 'Bob';

-- Commit the transaction if everything is successful
COMMIT;
```

Demonstrating Different Isolation Levels

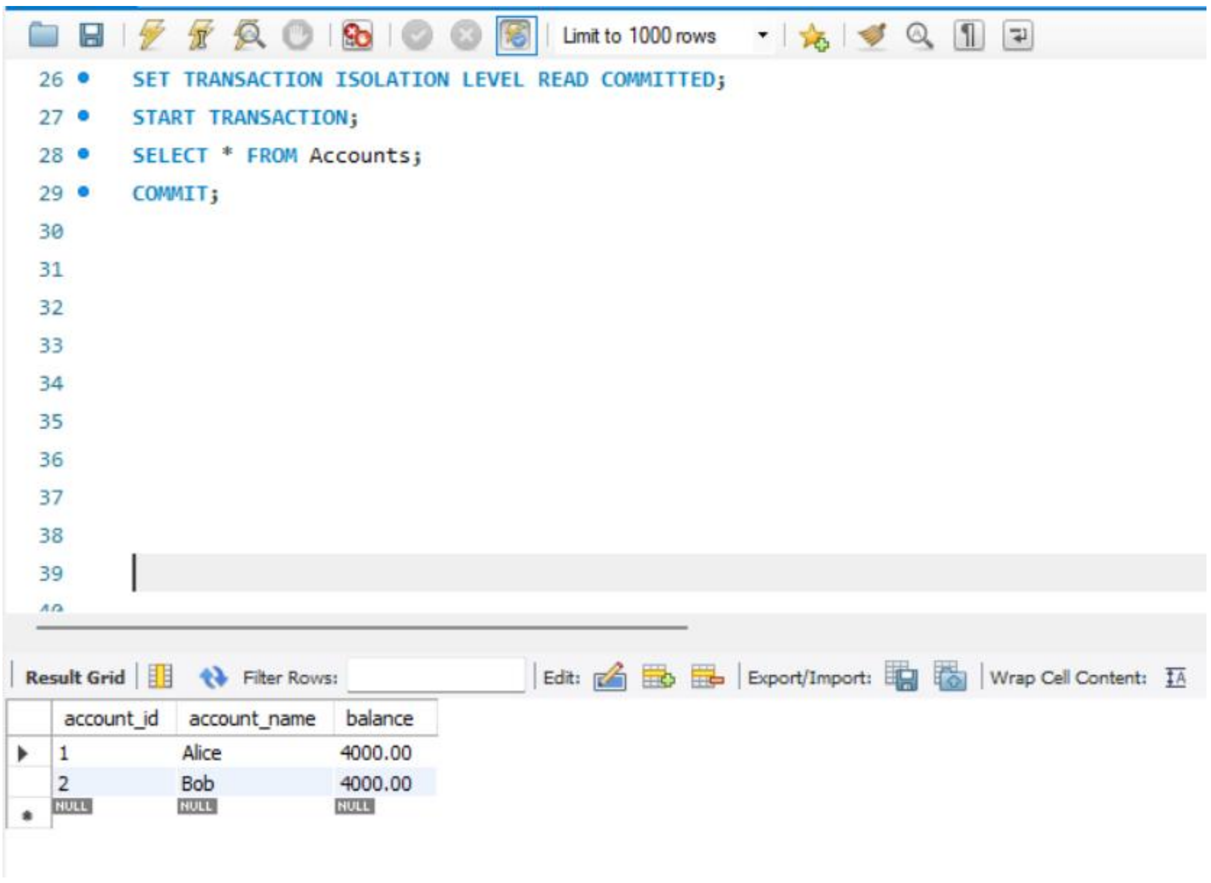
Step 3 : Read Uncommitted (Lowest Isolation).



```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
START TRANSACTION;
SELECT * FROM Accounts;
COMMIT;
```

account_id	account_name	balance
1	Alice	4000.00
2	Bob	4000.00
NULL	NULL	NULL

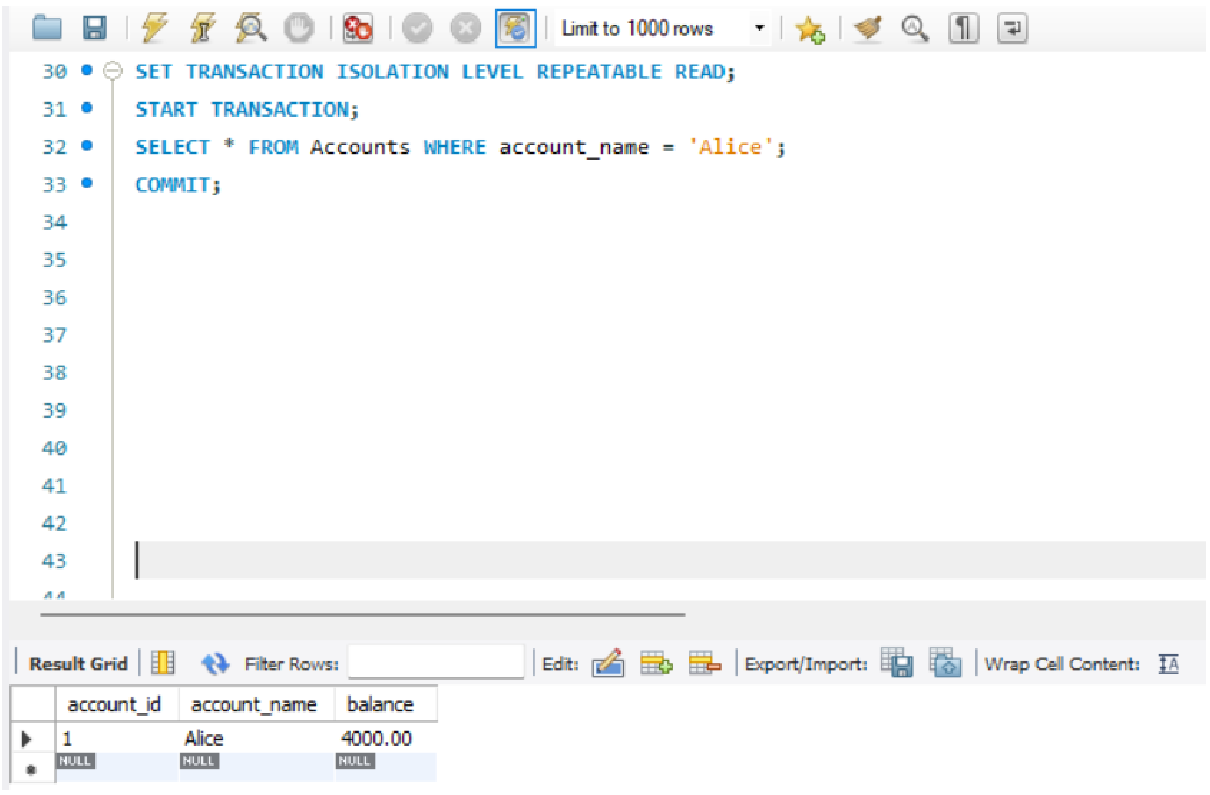
Step 4 : Read Committed (Default in Many Databases).



```sql
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
SELECT * FROM Accounts;
COMMIT;
```

account_id	account_name	balance
1	Alice	4000.00
2	Bob	4000.00
NULL	NULL	NULL

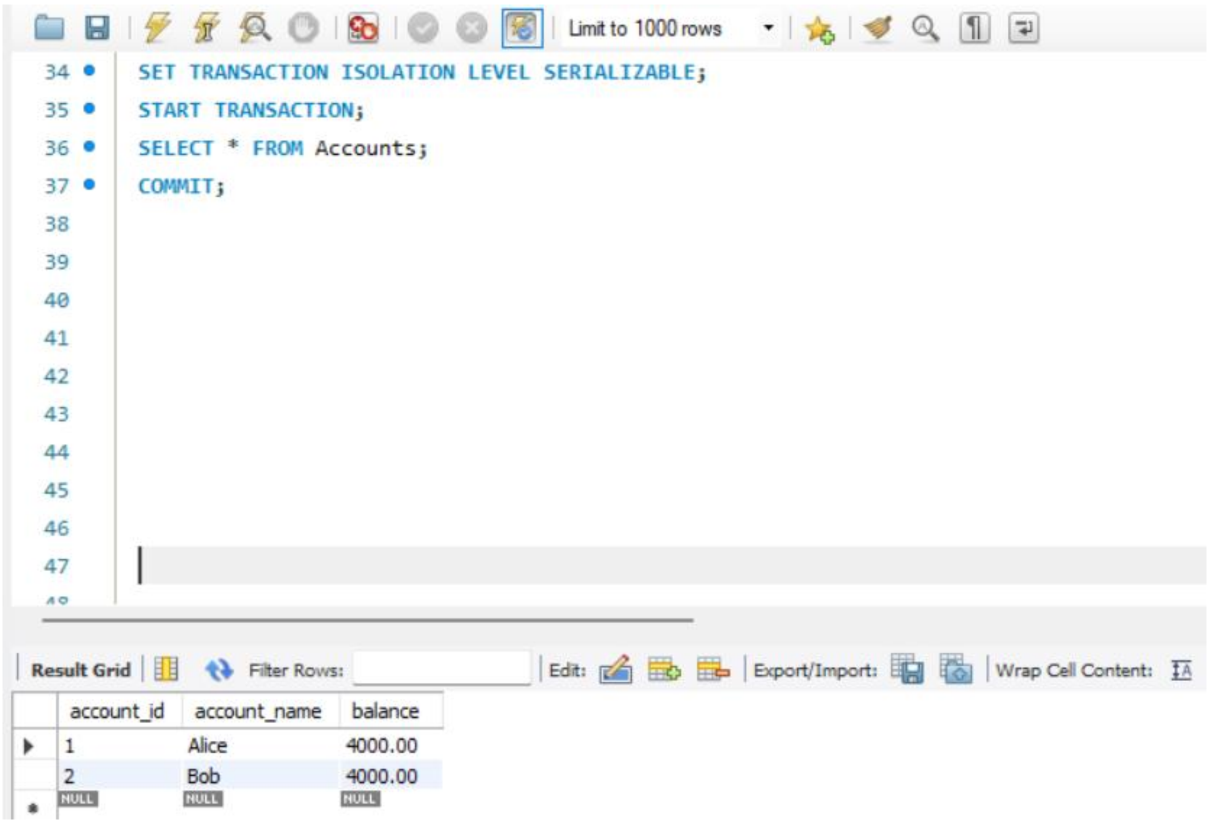
Step 5 : Repeatable Read.



```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;
SELECT * FROM Accounts WHERE account_name = 'Alice';
COMMIT;
```

account_id	account_name	balance
1	Alice	4000.00
NULL	NULL	NULL

Step 6 : Serializable (Highest Isolation).



```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
SELECT * FROM Accounts;
COMMIT;
```

account_id	account_name	balance
1	Alice	4000.00
2	Bob	4000.00
NULL	NULL	NULL