This lecture is **step-by-step walkthrough** for delivering lecture on transaction isolation levels in a DBMS using a bankDummyAccount database. The example involves two transaction sessions to demonstrate different isolation levels (READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE).

**Step 1: Create the Database and Table**

CREATE DATABASE bankDummyAccount;

USE bankDummyAccount;

**CREATE TABLE accounts (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**owner VARCHAR(50),**

**balance DECIMAL(10, 2),**

**currency VARCHAR(10),**

**created\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP**

**);**

**-- Insert initial data**

**INSERT INTO accounts (owner, balance, currency) VALUES**

**('Awais', 1000.00, 'USD'),**

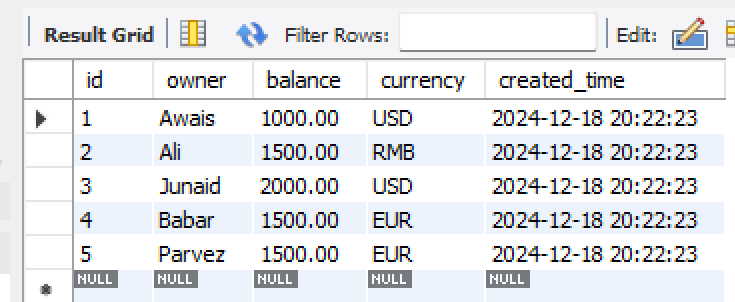
**(Ali, 1500.00, 'RMB');**

**('Junaid', 2000.00, 'USD'),**

**('Babar, 1500.00, 'EUR');**

**(Parvez, 1500.00, 'EUR');**

-- View initial data

**SELECT \* FROM accounts;**

**Initial Data in Table (accounts):**

Step 2: Explain Locking Concepts

1. Shared Locks (S):

* Acquired when a transaction reads a row (e.g., SELECT statement).
* Other transactions can also acquire shared locks on the same row, allowing concurrent reads.

2. Exclusive Locks (X):

* Acquired when a transaction modifies a row (e.g., INSERT, UPDATE, DELETE).
* Prevents other transactions from acquiring any lock on the same row.

3. Escalation:

* Locks may escalate from row-level to table-level if many rows are locked.

Step 3: Walkthrough of Isolation Levels with Locking Examples

Isolation Level 1: READ UNCOMMITTED

Locks: No locks are applied for reading.

Allows: Dirty reads, non-repeatable reads, and phantom reads.

Example 1: Dirty Read

**Session A:**

**SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;**

**START TRANSACTION;**

**UPDATE accounts SET balance = 800.00 WHERE id = 1; -- No lock is applied**

**Session B:**

**SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;**

**START TRANSACTION;**

SELECT \* FROM accounts WHERE id = 1; -- Reads uncommitted balance (800.00)

Output in Session B (Dirty Read):

id owner balance currency

1 Awais 800.00 USD

Isolation Level 2: READ COMMITTED

Locks:

* Shared locks (S) for reading, released after the read.
* Exclusive locks (X) for writing, held until commit.

Example: Prevent Dirty Reads

**Session A:**

**SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;**

**START TRANSACTION;**

UPDATE accounts SET balance = 800.00 WHERE id = 1; -- Exclusive lock applied to id=1.

**Session B:**

**SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;**

**START TRANSACTION;**

UPDATE accounts SET balance = 700.00 WHERE id = 1; -- Blocked until Session A commits.

SELECT \* FROM accounts; -- What you think it will return the current committed values, 700 or 800.?

SELECT \* FROM accounts WHERE id = 1; -- This will also return the current committed value.

Session A:

COMMIT; -- Lock is released.

Session B (After Commit in Session A):

SELECT \* FROM accounts WHERE id = 1; -- Will now see the updated value of 800.00.

Output in Session B (After Commit):

id owner balance currency

1 Awais 800.00 USD

Isolation Level 3: REPEATABLE READ

Locks:

Shared locks (S) are held for the duration of the transaction.

Exclusive locks (X) for writing, held until commit.

Prevents: Dirty reads and non-repeatable reads.

Example 3: Prevent Non-Repeatable Reads

**Session A:**

**SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;**

**START TRANSACTION;**

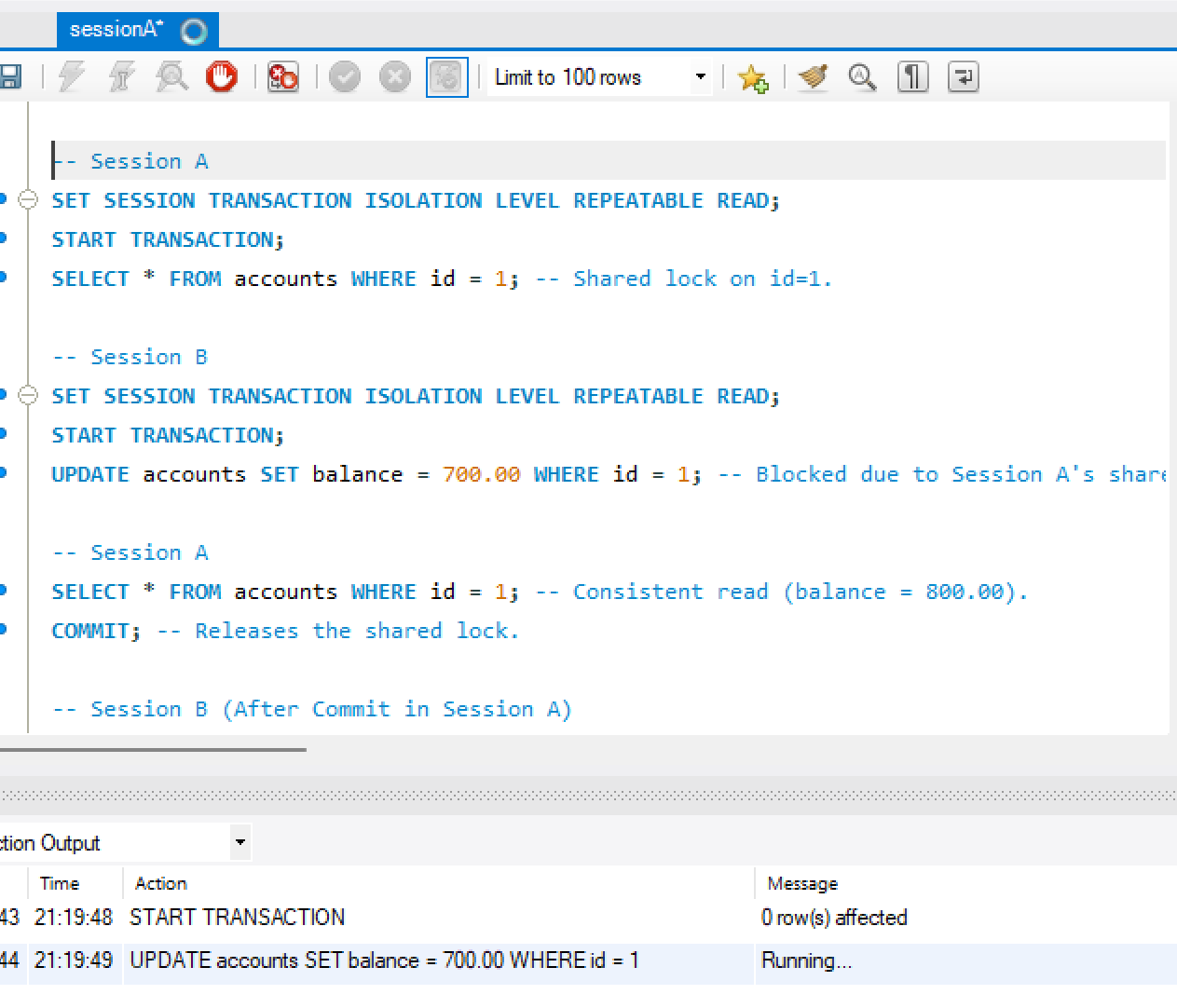
SELECT \* FROM accounts WHERE id = 1; -- Shared lock on id=1.

**Session B:**

**SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;**

**START TRANSACTION;**

UPDATE accounts SET balance = 700.00 WHERE id = 1; -- Blocked due to Session A's shared lock.



Session B is blocked until Session A commits or rolls back.

Session A:

SELECT \* FROM accounts WHERE id = 1; -- Consistent read (balance = 800.00).

COMMIT; -- Releases the shared lock.

Session B (After Commit in Session A):

UPDATE accounts SET balance = 700.00 WHERE id = 1;

COMMIT;

Isolation Level 4: SERIALIZABLE

Locks:

Shared locks (S) for reading, held for the duration of the transaction.

Exclusive locks (X) for writing, held until commit.

Prevents phantom reads by locking entire ranges.

Example 4: Prevent Phantom Reads

**Session A:**

**SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

**START TRANSACTION;**

SELECT \* FROM accounts WHERE balance > 1500.00; -- Shared lock on range balance > 1500.00.

**Session B:**

**SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

**START TRANSACTION;**

INSERT INTO accounts (owner, balance, currency) VALUES ('New User', 1600.00, 'USD'); -- Blocked.

Session B is blocked due to the range lock held by Session A.

Session A:

COMMIT; -- Releases the range lock.

Session B (After Commit in Session A):

INSERT INTO accounts (owner, balance, currency) VALUES ('New User', 1600.00, 'USD');

COMMIT;

Step 4: Summary of Locking in Isolation Levels

* Isolation Level Read Lock (S) Write Lock (X) Range Lock Behavior
* READ UNCOMMITTED None None None Dirty reads, no blocking.
* READ COMMITTED Held during read Held until commit None Prevents dirty reads, allows concurrent updates.
* REPEATABLE READ Held until commit Held until commit None Prevents dirty and non-repeatable reads.
* SERIALIZABLE Held until commit Held until commit Held on ranges Prevents dirty, non-repeatable, and phantom reads.