Assignment 03:

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 stands for Atomicity, Consistency, Isolation, and Durability, which are essential properties that guarantee data integrity and reliability in database transactions. Here's a breakdown of each property:

Atomicity: Imagine a transaction as a single unit of work. Atomicity ensures that either all operations within the transaction succeed, or none of them do. It's like flipping a coin - it can only land on heads or tails, not halfway in between. In the database context, this means either all the data changes in your transaction are applied, or none are, preventing partial updates that could leave the data in an inconsistent state.

Consistency: This property ensures that a transaction transforms the database from one valid state to another. It follows predefined business rules. Imagine a

bank account balance. A transaction that debits the correct amount but forgets to update the available balance would violate consistency. Consistency rules maintain the integrity and accuracy of your data.

Isolation: This property ensures that concurrent transactions executing at the same time don't interfere with each other's data. It's like having separate lanes on a highway to prevent collisions. Even if multiple users are modifying the database simultaneously, isolation guarantees that each transaction sees a consistent view of the data and doesn't get influenced by the changes of others until the transaction is committed.

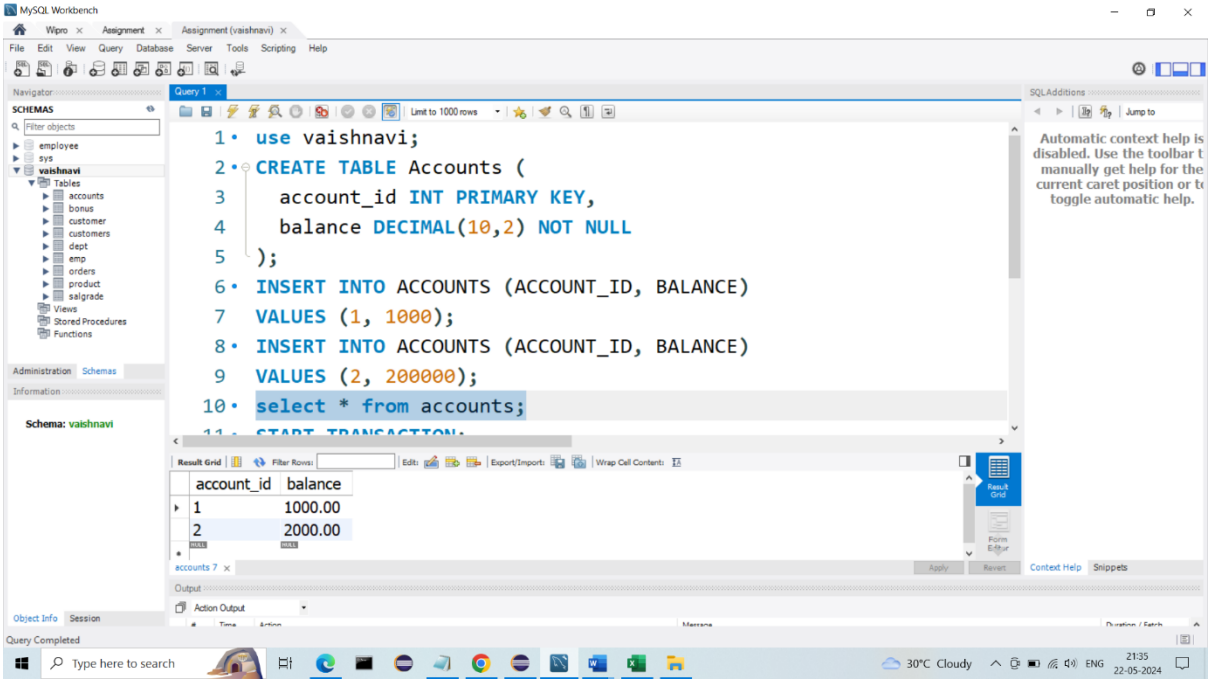
Durability: Once a transaction commits (finishes successfully), the changes to the database become permanent. Even if a system failure occurs after the commit, the changes are persisted and won't be lost. Durability is like writing your final exam answer on a permanent answer sheet - even if the power goes out, your answer is saved.

Transaction with Locking and Isolation Levels

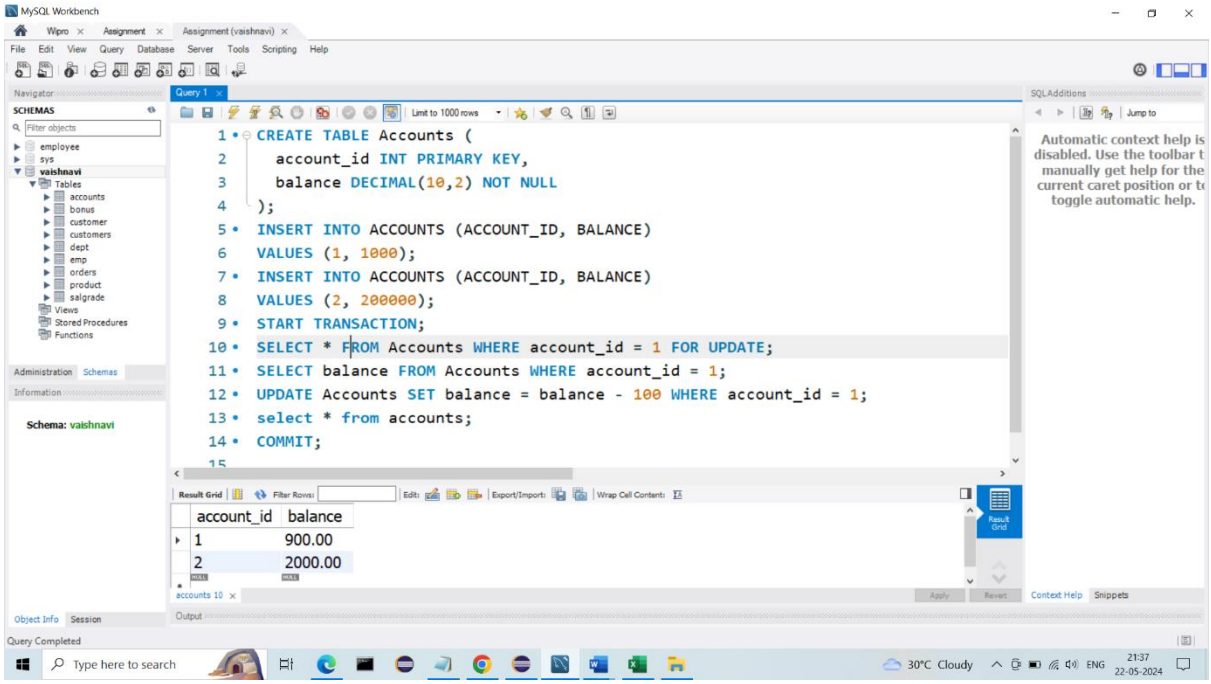
Here's an example transaction that simulates updating a bank account balance with locking and demonstrates different isolation levels:

Scenario: Transferring $100 from account A (account_id = 1) to account B (account_id = 2).

Before locking and updating transaction-:



After locking and updating transaction-:



Isolation Levels:

- **READ COMMITTED (Used in the example):** This level ensures the transaction sees a consistent view of data as of the moment it starts reading each row. It prevents "dirty reads" where a transaction might read uncommitted changes from another transaction.

Other Isolation Levels:

- **READ UNCOMMITTED:** This allows a transaction to see uncommitted changes from other transactions, which can lead to inconsistencies if those transactions are rolled back.
- **REPEATABLE READ:** Ensures the transaction sees a consistent view of data throughout its execution,

preventing "phantom reads" where new rows are inserted by other transactions during the current transaction's execution.

- **SERIALIZABLE:** Provides the strongest isolation level, ensuring transactions are executed one at a time, like a serial queue. This can improve data integrity but can also impact performance due to serialization overhead.