Database Systems

B.Sc. (Hons) in IT

Laboratory Worksheet 08

Transaction and Concurrency Control Simulations

Use MS SQL Server and create a bank database. create two accounts, with account numbers “1234567899”, “1234567898” and place Rs. 100 in each of them. You may choose the accounts to be of any type belonging to a single customer.

Now, write a script to transfer funds from the accounts. The accounts numbers and amount to transfer must be declared as variables @soucreAcc, @destAcc and @fundsToTransfer respectively.

In your fund transfer script, ensure that the destination account is credited prior to deducting from the source account.

You should use appropriate syntax to ensure proper transaction mechanisms for the fund transfer (i.e. BEGIN TRANSACTION/COMMIT TRANSACTION).

Save your script as “Lab8.sql”.

Important: It is important to set the variable **XACT_ABORT** (i.e. SET XACT_ABORT ON) prior to executing transactions in order to simulate “Atomicity” property. Also, ensure that the constraint in balance >= 0 is present in the account table. (Note: You can add constraints using ALTER TABLE).

- **Simulation 1: Atomicity**
  Input: @sourceAcc = 1234567899
  @destAcc = 1234567898
  @ fundsToTransfer = 150.00

  Expected Behavior: The system should abort the transaction (constraint – balance > 0 - is violated). The account balances are not changed. Verify this behavior.

  Briefly explain this simulation result focusing on the reason for its behavior.

- **Simulation 2: Concurrency Control**
  Open two database connections (say A and B). Load the Lab7.sql script to both applications. Use the inputs of simulation 1 for both scripts.

  In Database Connection A:

  Execute only the part of the script till the first update statement

  In Database Connection B:

  Execute only the part of the script till the first update statement.

  Expected Behavior*:* The script in B has not completed execution. The transaction is waiting for the release of locks from A’s script.

  Then complete the rest of script in A.
  Expected Behavior: The update in B completes

  Then execute a “SELECT * FROM Account” in A.
  Expected Behavior: The query executes and waits until B releases the locks

Now execute the rest of B's transaction.
Expected Behavior: A's select statement completes and results are output.

Briefly explain these simulation results discussing the reasons for their behavior.

- **Simulation 3: Deadlock**

  Step 1: Consider a deadlock scenario using two or more transactions
  Step 2: Write SQL script for each transaction
  Step 3: Execute a part of each transaction until the deadlock occurs. (Hint: Use a separate database connection for each transaction).

  Hint: Use *SERIALIZABLE* isolation level for each transaction.

  ***Expected Behavior*:** A victim transaction is aborted by the SQL Server with a similar message:
  "`Msg 1205, Level 13, State 56, Line 1`
  `Transaction (Process ID 53) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction`".

  Explain your experiment in detail.