Lab 9: Concurrency Control Assignment

Utkarsh Ranjan
200050147

Transaction

1. The rollback undo/revert all the queries run after the begin statement. Thus after rollback command, student entry of "Tanak" remains unchanged, and the delete command has no effect.

Concurrency

1. In the second window, another transaction is started using the "begin" command, followed by a select statement on the "student" table to retrieve the row for the student named "Tanaka". The select statement returns the row with the old value of "tot_cred", which is not updated yet by the first transaction.

   When the first transaction is committed using the "commit" command in the first window, the changes made by that transaction are made permanent and visible to other transactions. However, the second window still sees the old value of "tot_cred" because its transaction has already completed, and it is not affected by the changes made by the first transaction.

2. In the second window, another transaction is started using the "begin" command, followed by a select statement to retrieve the minimum value of "tot_cred" for the student named "Tanaka". However, since the first window has already updated the row for "Tanaka" but has not committed yet, the second window is blocked and waiting for the first transaction to complete.

   In the first window, the transaction is committed using the "commit" command. This releases the lock on the updated row and allows the second window to proceed.

   In the second window, the transaction is also committed using the "commit" command. The update statement in the second window sets the value of "tot_cred" to the minimum value of "tot_cred" for the student named "Tanaka" plus 20, which is 75.

3. When both transactions set the isolation level to serializable, they ensure that their execution is equivalent to a serial execution, even though they run concurrently. The first transaction updates the row with a value of 44, and the second transaction waits for it. Once the first transaction is committed, the second transaction retrieves the updated value and tries to update it, but since the first transaction already updated the row, the second transaction encounters a serialization error and rolls back. The serialization mechanism ensures that the final value of "tot_cred" for the student named "Tanaka" reflects the correct order of the transactions, even though they were executed concurrently.

4. In this scenario, two transactions concurrently update the salary of different instructors with each other's salary, which creates a conflict. Since both transactions have the same serializable isolation level, they must be executed serially. After the first transaction commits, the second transaction is unblocked and commits. The final state of the system shows that both instructors have the same salary, which is the salary of the instructor whose salary was used in the first transaction. This demonstrates the importance of ensuring that transactions do not conflict with each other when accessing the same data in a concurrent environment.