CS 775/875 TRANSACTIONS

1. The relevant reading material is mainly from class notes; chapter 20 will give you a wider perspective.
2. The MySQL implementation of isolation levels varies in some aspects from the SQL standard covered in class.
3. This is a lab cum quiz/exam problems. The goal is to help you understand how concurrent transactions perform under various isolation levels. I recommend that you open concurrent MySQL sessions and in each session start a transaction. Try out commands similar to the questions in this assignment. Log into a MySQL session.

To see the default settings:

```
SELECT @@tx_isolation;
SELECT @@autocommit;
```

To start transaction:

```
SET TRANSACTION ISOLATION LEVEL < isolationlevel >;
START TRANSACTION;
SELECT * from < table >
UPDATE/INSERT/DELETE ....
finally COMMIT or ROLLBACK to end transaction.
```

If you do not start a transaction then each individual query is treated as a separate transaction. Alternatively, you could turn autocommit off. Note that isolation levels are relevant only for reads from database; all writes to the database require exclusive locks.

1. (20) Consider the following transactions to the COMPANY database. Assume that each query runs atomically and all transactions complete successfully.

```
T1:
    INSERT INTO EMPLOYEE VALUES
    ('Ahmad','V','Khan','987987487','1959-03-29','980 Dallas TX','M',
     '25000.00','987654321','4');
    Commit;

T2:
    Select avg(Salary) From EMPLOYEE;
    Select avg(Salary) From EMPLOYEE;
    Commit;
```

In each case, state (YES/NO) whether the output of the transactions can show non-serializable (or non-atomic) execution if

(a) T2 is Read Committed.	YES	NO
(b) T2 is Read Uncommitted.	YES	NO
(c) T2 is MySQL's Repeatable Read.	YES	NO
(d) T2 is Repeatable Read with phantom tuples.	YES	NO

```
T1:
    INSERT INTO EMPLOYEE VALUES
    ('Ahmad','V','Khan','987987487','1959-03-29','980 Dallas TX','M',
     '25000.00','987654321','4');
    Rollback;

T2:
    Select avg(Salary) From EMPLOYEE;
    Select avg(Salary) From EMPLOYEE;
    Commit;
```

Reconsider the previous question: Suppose T1 **rollbacks**. In each case, state (YES/NO) whether the output of the transactions can show non-serializable (or non-atomic) execution if

(a) T2 is Read Committed.	YES	NO
(b) T2 is Read Uncommitted.	YES	NO
(c) T2 is MySQL's Repeatable Read.	YES	NO
(d) T2 is Repeatable Read with phantom tuples.	YES	NO

```
T1:
    INSERT INTO EMPLOYEE VALUES
    ('Ahmad','V','Khan','987987487','1959-03-29','980 Dallas TX','M',
     '25000.00','987654321','4');
    Commit;

T2:
    Select * From DEPARTMENT;
    Select avg(Salary) From EMPLOYEE;
    Commit;
```

In each case, state (YES/NO) whether the output of the transactions can show non-serializable (or non-atomic) execution if

(a)	T2 is Read Committed.	YES	NO
(b)	T2 is Read Uncommitted.	YES	NO
(c)	T2 is MySQL's Repeatable Read.	YES	NO
(d)	T2 is Repeatable Read with phantom tuples.	YES	NO

```
T1:
    INSERT INTO EMPLOYEE VALUES
    ('Ahmad','V','Jabbar','987987988','1959-03-29','980 Dallas TX','M',
     '25000.00','987654321','4');
    DELETE FROM EMPLOYEE WHERE Dno=6;
    Commit;

T2:
    Select avg(Salary) From EMPLOYEE;
    Select * From DEPARTMENT;
    Commit;
```

In each case, state (YES/NO) whether the output of the transactions can show non-serializable (or non-atomic) execution if

(a)	T2 is Read Committed.	YES	NO
(b)	T2 is Read Uncommitted.	YES	NO
(c)	T2 is MySQL's Repeatable Read.	YES	NO
(d)	T2 is Repeatable Read with phantom tuples.	YES	NO

```
T1:
    INSERT INTO EMPLOYEE VALUES
    ('Ahmad','V','Shah','9879888987','1959-03-29','980 Dallas TX','M',
     '25000.00','987654321','4');
    DELETE FROM PROJECT WHERE Pno=5;
    Commit;

T2:
    Select avg(Salary) From EMPLOYEE;
    Select count(*) From PROJECT;
    Commit;
```

In each case, state (YES/NO) whether the output of the transactions can show non-serializable (or non-atomic) execution if

(a) T2 is Read Committed.	YES	NO
(b) T2 is Read Uncommitted.	YES	NO
(c) T2 is MySQL's Repeatable Read.	YES	NO
(d) T2 is Repeatable Read with phantom tuples.	YES	NO

2. Consider table R(a) with values (1, 2).

```
T1:
  start transaction;
  Update R Set a = a + 1;
  Insert into R Values (4);
  Commit;
T2:
  Select Sum(a) From R;
  Select Sum(a) From R;
  Commit;
```

Give all possible outputs of T2 when

(a) (2) T2 executes with isolation level Serializable.

(b) (2) T2 executes with isolation level Repeatable-Read where phantom tuples are evaluated. List only those outputs that **do not overlap with level Serializable**.

(c) (2) T2 executes with isolation level Read-Committed. List only those outputs that **do not overlap with level Serializable**.

(d) (4) T2 executes with isolation level Read-Uncommitted; assume that table R is updated after getting a lock on the entire table. List only those outputs that **do not overlap with Read Committed**.

(e) (1) T2 executes with isolation level Repeatable-Read without phantom tuples. List only those outputs that **do not overlap with level Serializable**.

3. Reconsider the above question when transaction T1 **rollbacks**. Consider table R(a) with values (1, 2).

```
T1:
Update R Set a = a + 1;
Insert into R Values (4);
Rollback;
T2:
Select Sum(a) From R;
Select Sum(a) From R;
Commit;
```

Give all possible outputs of T2 when

(a) (1) T2 executes with isolation level Serializable.

(b) (1) T2 executes with isolation level Repeatable-Read where phantom tuples are evaluated. List only those outputs that **do not overlap with level Serializable**.

(c) (1) T2 executes with isolation level Read-Committed. List only those outputs that **do not overlap with level Serializable**.

(d) (3) T2 executes with isolation level Read-Uncommitted; assume that table R is updated after getting a lock on the entire table.

List only those outputs that **do not overlap with READ UNCOMMITTED when T1 commits - Problem 2 (d)**.

4. Consider the following transaction on table R(a).

```
T1:
  update R set a = 2*a;
  Commit
T2:
  update R set a = 2+a;
  Commit
T3:
  select a from R;
  select a from R;
```

Assume that each query executes atomically. Let R have value (1) before the start of transaction. Give all possible outputs of T3 when

(a) (5) T3 execute with isolation level Serializable:

(b) (6) T3 execute with isolation level Read Committed. List only those outputs that **do not overlap with Serializable**.

(c) (1) T3 executes with MySQL's isolation level Repeatable Read. List only those outputs that **do not overlap with Serializable**.

(d) (1) T3 executes with isolation level Repeatable Read, phantom tuples. List only those outputs that **do not overlap with Serializable**.