**Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Reading:**

* Chapter 14 in murach textbook.
* Watch my videos posted on iLearn about transactions and locking.

**Part 1**

Based on your reading answer the following questions.

1. What is the purpose of concurrency control?
2. Explain the trade-off that exist in concurrency control.
3. Explain the difference between concurrent transactions and simultaneous transactions. How many CPUs are required for simultaneous transactions?
4. Define the terms *dirty read, nonrepeatable read*, and *phantom read.* How are these terms related to isolation level?
5. What is lock granularity?
6. Explain the difference between exclusive lock and shared lock.
7. What is deadlock? How can it be avoided? How can it be resolved when it occurs?
8. Explain the difference between optimistic and pessimistic locking.
9. Explain the meaning of the expression *ACID transaction*.

Two important application sql variables related to transactions.

**AUTOCOMMIT:** a single sql statement always has ACID properties. If you have a multiple row update statement, either the entire statement will be successful or nothing will occur. If there is an error during processing of the update, any modifications already performed will be rolled back. But if you want a transaction to span multiple sql statement, then auto commit must be turned OFF. In MySQL this is done with the statement

**SET AUTOCOMMIT = 0;**

You can change auto commit to ON by setting the value to 1. See <https://dev.mysql.com/doc/refman/5.7/en/commit.html> for more information.

**TRANSACTION ISOLATION LEVEL:**

You can set the value of transaction isolation to ***read uncommitted, read committed, repeatable read*** and ***serializable***. The values determine if records will be locked and whether the lock will be held temporarily or until transaction commit. See <https://dev.mysql.com/doc/refman/5.7/en/set-transaction.html>, and <https://dev.mysql.com/doc/refman/5.7/en/innodb-transaction-isolation-levels.html> for more details on MySQL.

Note: MySQL has two database engines MyISAM and INNODB. MyISAM does not support transactions and locking. MyISAM is used in the WORLD schema tables. INNODB is used by default when you create tables. So unless you specify MyISAM, you will have support for transactions.

**Part 2**

You can see the effects of transaction isolation level for yourself by doing the following steps. You must have TWO CONNECTIONS to the DMBS server that that you can do two concurrent transactions.

1. Start mysql workbench.

Open and run **A11\_setup.sql** script to create the cst438 schema and the cst438.city table data.

2. Connect and open a query tab. In the query tab window execute the statements:

**SET AUTOCOMMIT=0;**

**SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;**

3. **IMPORTANT**: **Now go back to the HOME TAB and do a second connection** and open a query tab. You now have 2 query tabs but each tab has a separate connection to the server. Enter the same command in query tab 2

**SET AUTOCOMMIT=0;**

**SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;**

4. Use scripts A11\_client1.sql and A11\_client2.sql to help do the following exercise.

|  |  |  |
| --- | --- | --- |
|  | **Query tab connection 1** | **Query tab connection 2** |
| 1 | ## step 1  set autocommit=0;  set transaction isolation level repeatable read;  select @@autocommit, @@tx\_isolation;  ## pause here. go to step 2 at client 2 |  |
| 2 |  | ## step 2  set autocommit=0;  set transaction isolation level repeatable read;  select @@autocommit, @@tx\_isolation;  select population from cst438.city where id=1820;  ## pause here. go to step 3 at client 1 |
| 3 | ## step 3  ## the next update statement should hang  ## wait several seconds, it will timeout.  ## why? Client#2 has a lock on the record for id=1820.  update cst438.city set population=population+1000 where id=1820; |  |
| 4 |  | ## step 4. change isolation level  ## to Read Commited.  commit;  set transaction isolation level read committed;  select population from cst438.city where id=1820;  ## pause here. Go to step 5 at client 1. |
| 5 | ## step 5. Change isolation to read committed.  rollback;  set transaction isolation level read committed;  ## the update statement will not hang because of read committed.  update cst438.city set population=population+1000 where id=1820;  select population from cst438.city where id=1820;  ## pause here. go to step 6 in client 2. |  |

|  |  |  |
| --- | --- | --- |
| 6 |  | ## step 6. re-read the population data.  ## you should see the same value even though  ## client 1 has updated it.  ## why? client 1 has not commited the change yet.  select population from cst438.city where id=1820;  ## pause here. go to step 7 in client 1 |
| 7 | ## step 7. Now commit the transaction.  commit;  ## go to step 8 in client 2 |  |
| 8 |  | ## step 8. re read the population. This time you see the new value.  select population from cst438.city where id=1820;  commit; |

**Questions**

1. Explain in your own words why step 3 failed and the update statement done by connection 1 timed out?
2. What is the difference between using REPEATABLE READ and READ COMMITTED regarding reading and locking data? what about writing and locking data?

**Part 3. Design your own demo of concurrency and isolation level.**

You have a table that keeps track of inventory of part numbers. Create a table that has id as primary key, partName and inventoryCount as other columns. Load the table with 3 rows of data for part ids 1, 2, and 3 and inventoryCount of 2 for each part.

Client 1 needs to use 2 of part 1 and 2 of part 2. Client one first reads the inventory count of parts 1 and 2 to verify that there is sufficient quantity of hand. Then updates the new count of part 1 and the new count of part 2.

Concurrently client 2 needs to use 2 of part 1 and 2 of part 3.

First run the scripts using autocommit=0 and read committed isolation level. Is the inventory count correct at the end of both client transactions?

Now rerun the scripts but use serializable isolation level. What is the difference in behavior? Do you get the correct inventory count at the end?

Is you rerun the scripts with repeatable read isolation level, do you get the correct behavior?

To make the scripts easier to work with you can use variables in the script

To read current inventory of parts 1 and 2 you can code

select count into @countp1 from inventory where id = 1;

select count into @countp2 from inventory where id = 2;

# pause here so that user can verify sufficient quantity

# and client 2 can run concurrently

update inventory set count=@countp1 - 2 where id = 1;

update inventory set count=@countp2 - 2 where id = 2;