CECS 323 LAB TRANSACTIONS

OBJECTIVE: Get some first-hand experience with transactions and isolation levels.

INTRODUCTION: The Derby database defaults to performing a commit after each DML statement. This behavior helps to make sure that you do not lose data if the database goes offline unexpectedly, but it also means that you do not have a choice on the boundaries of your transactions. In this lab, we are going to show you how the database management system isolates transactions from each other so that they cannot interfere with each other.

See: https://db.apache.org/derby/docs/10.6/devguide/cdevconcepts15366.html for a discussion regarding locking and concurrency control in the Derby database.

By contrast, the MySQL engine that we'll default to uses multi-version concurrency control. A good article on that can be found at: https://medium.com/@ajones1_999/understanding-mysql-multiversion-concurrency-control-6b52f1bd5b7e. The wonderful thing about multi-version concurrency control is that readers never block another transaction and writers never block readers.

PROCEDURE:

Derby database:

Some of you installed Derby using the lib distribution, some used the bin distribution. Depending upon how you installed Derby, follow the appropriate instructions for these first few steps:

Bin	**Lib**
1. Open a command prompt and navigate to where you have installed Derby, and then cd to the **bin** directory within your Derby installation.	1. Open a command prompt and navigate to where you have installed Derby, and then cd to the **lib** directory within your Derby installation.
2. Start the Derby server by typing in "startNetworkServer" at the command line.	2. Start the Derby server by typing in "java -jar derbyrun.jar server start".
Note, the Derby database engine is now running in the foreground in this window. Leave the window open until you are done with this exercise. If there are any exceptions thrown by Derby, you will see them in this window.	
3. Open a **separate** window in the same directory where you launched the Derby server. Enter the command "ij".	3. Open a **separate** window in the same directory where you launched the Derby server. Enter the command "java -jar derbyrun.jar ij"
4. Now, open a **third** window in the same directory where you launched the Derby server. Again enter the command "ij".	4. Open a **third** window in the same directory where you launched the Derby server. Enter the command "java -jar derbyrun.jar ij"
This will put you into a Derby command prompt where we can use Derby commands to give us more control that we would going through a typical IDE.	

5. Type in the command: “connect 'jdbc:derby://localhost:1527/DemoDB;create=true';” This will create a new Derby database named DemoDB[1].
6. In your third window (the second transaction window) connect to the database.
   a. In **this** window, connect to the database by saying “connect ‘jdbc:derby://localhost:1527/DemoDB’;
   b. We do not want to recreate the database at this point, we want two transactions both hitting the same database at the same time.
7. Back in the first ij session, type in the command: “create schema transactions;” This will create a new scheme that you will use for this lab and this lab alone.
8. Then enter the command “set schema transactions;” to make that new schema your default.
9. In the second ij session, issue the command: “set scheme transactions;” to make sure that both sessions are using the same schema as their default.
10. In both transaction windows, enter the command: “call SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY('derby.locks.waitTimeout', '15');” This will make Derby timeout more quickly when it attempts to obtain a lock. This will make the lab go faster for you.
11. Then in the **first transaction window**, run the SQL statements that you will find here.
    a. You can cut/paste those statements into your ij command window, or you can put them into file and run the file using the run command in ij.
    b. Put the file name in single quotes. The run command needs a “;” on the end of it.
12. In **both** windows, enter the following:
    a. Type in the command: “autocommit off;” – This will tell Derby **not** to commit after each DML statement. That will allow us to observe transaction isolation at work.
    b. Type in the command: “set isolation rs;”. This will put your session into the repeatable read isolation level. The default transaction isolation level for Derby is read committed, which only prevents “dirty reads”.
13. Fill out the following table. Designate one of your Derby ij sessions as Transaction 1, and the other as Transaction 2. Perform the following commands in the designated transaction window, and **in the specified order**.
    a. In each case, look at the statement and try to predict what might happen.
    b. Put your prediction (theory) into the proper cell in the spreadsheet.
    c. If something **different** happens, specify that in the “Actual Results” column.
    d. Remember, some of these actions will cause a lock to occur. Give it a minute or two for your transaction to time out, note that, and move on. **Do not abort any** of your transactions.

Step #	Transaction 1	Transaction 2	Expected Results	Actual Results
1.		SELECT COUNT(*) FROM students;		
2.		SELECT AVG(GPA) FROM students;	3.4124999999999996	

[1] The location of this database will be a DemoDB folder that is either under the /bin or the /lib directory, depending on which directory has the Derby jar file that you used to start the network server.

3.	INSERT INTO students (last_name, first_name, major, year_study, GPA) values ('Piggy', 'Miss', 'drama', 'senior', 3.99);			
4.		SELECT COUNT(*) FROM students;[2]		If you do **not** get a timeout waiting for a lock, let me know.
5.		SELECT COUNT(*) FROM students where last_name = 'the Frog';		
6.		select avg(GPA) from students;		
7.		SELECT COUNT(*) FROM (SELECT last_name FROM students WHERE last_name IN ('the Frog', 'Einstein')) abc;		
8.	commit;			
9.		SELECT AVG(GPA) FROM students;		
10.		commit;		
11.		SELECT AVG(GPA) FROM students;		

Bin	**Lib**
12. Use the exit; command in the ij window to get back to the DOS prompt, then issue the command "stopNetworkServer" to shut down the Derby server.	11. Use the exit; command in the ij window to get back to the DOS prompt, then issue the command: "java -jar derbyrun.jar server shutdown" to stop the Derby server.

[2] This will take some time, be patient. The reason will become clear soon enough.

MySQL:

1. Either connect to the campus MySQL instance, or a local MySQL instance on your laptop. We will again create a students table, so be sure that will not conflict with an existing table in your schema. Use the DDL found here to build it in MySQL, it's slightly different from the DDL in Derby. That same file also has an insert statement to put in a little sample data.
2. If you are using MySQL workbench, you can easily create two connections to the same MySQL database. Just create the first connection as you always do, then push the home button in the upper left hand corner and create a second connection. There will now be two tabs at the top of your MySQL window, one for each session. Each tab will have its own set of open SQL windows.
3. In **both** tabs, issue the statement "set transaction isolation level **repeatable read**;" to make sure that you have the same isolation level that you did in Derby. The repeatable read transaction isolation level is the default in MySQL for the InnoDB database engine, which is the default engine.
4. Run the following statements in the order and transaction indicated and fill out the following table, just as you did for Derby:

Step #	Transaction 1	Transaction 2	Expected Results	Actual Results
1.	START TRANSACTION;			
2.		START TRANSACTION;		
3.		SELECT AVG(GPA) FROM students;		
4.		SELECT COUNT(*) FROM students;		
5.	INSERT INTO students (last_name, first_name, major, year_study, GPA) values ('Piggy', 'Miss', 'drama', 'senior', 3.99);			
6.		SELECT COUNT(*) FROM students;		
7.		SELECT COUNT(*) FROM students WHERE last_name = 'the Frog';		
8.		SELECT AVG(GPA) FROM students;		
9.		SELECT COUNT(*) FROM (SELECT last_name FROM students WHERE last_name IN ('the Frog',		

		'Washington')) abc;		
10.	COMMIT;			
11.		SELECT AVG(GPA) FROM students;		
12.		COMMIT;		
13.		SELECT AVG(GPA) FROM students;		

WHAT TO TURN IN:

- Your Word document showing your expectations and the results from each of the above statements for both Derby and MySQL.
- Your team's Lab Collaboration Document. You can find the template for that at BeachBoard | Content | Student Helps | Lab Collaboration Document.