CSCI 3901 Lab 8: Database transactions

Fall 2021

Objective

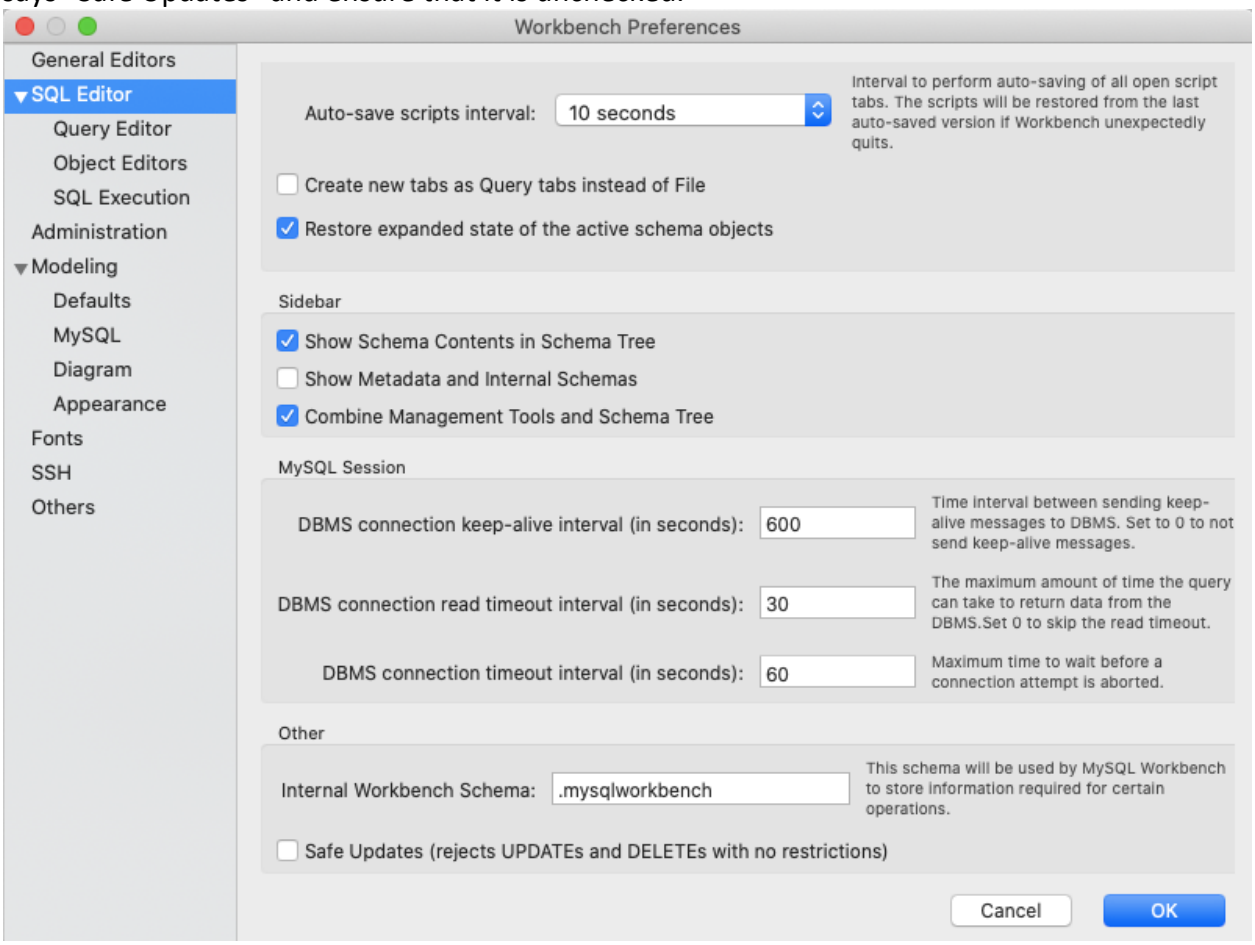
In this lab, you will practice making SQL transactions.

This lab is an individual activity.

This lab should be submitted in Brightspace by 4pm Halifax time on Monday, November 29, 2021.

Preparation

- Make sure that you are using your own private database on db.cs.dal.ca.
- Ensure that MySQLWorkbench is allowing you to do updates and deletions: in the MySQLWorkbench -> Preferences -> SQL Editor screen, scroll to the bottom entry that says “Safe Updates” and ensure that it is unchecked.



Resources

- Commands to work with tables in the database:
    - See the structure of a table
        - Describe <<tablename>>;
    - Create a table
        - create table <<tablename>> (col1Name col1Type, col2Name col2Type, …);
        - Example:
          create table test (id int, name varchar(50));
    - Change the specifications on a column of the table
        - alter table <<tablename>> modify column <<columnName>> <<new column type>>;
        - Example:
          alter table test modify column name varchar(75);
    - Add a column to a table
        - alter table <<tablename>> add column <<columnName>> <column type>>;
        - Example:
          alter table test add column address varchar(50);
    - Delete a column from a table
        - alter table <<tablename>> drop column <<columName>>;
        - Example:
          alter table test drop column name;
    - Delete a table
        - drop table <<tablename>>;
        - Example:
          drop table test;

Procedure

Set-up

1. Open a query tab in MySQLWorkbench to your private database on db.cs.dal.ca
2. Open a putty window to timberlea.cs.dal.ca and then sign in to your mysql database on db.cs.dal.ca:

   mysql -h db.cs.dal.ca -u xxxx -p

   where xxxx is your timberlea CS id.
3. We will use the MySQLWorkbench to make changes to the database and will use the putty window to verify the changes. These two windows will run separate sessions to the database. Two query tabs in MySQLWorkbench would use the same session, so we cannot verify the effects of transactions with just two tabs in MySQLWorkbench.
4. Create a starter table with a few fields in your database:

   create table test_table (id int not null auto_increment primary key, name varchar(50), age int);

5. Add a few rows of sample data into the table:

   insert into test_table values (NULL, “Jenny”, 30), (NULL, “John”, 31), (NULL, “Agnes”, 25);

6. Verify that you can see the data in both MySQLWorkbench and in the putty window:

   select * from test_table;

   Notice that since we made the id “auto_increment” in the line that creates the table, sending a value of NULL in the insert command has the database assign an id automatically to the new data.

Lab steps

1. Execute SQL commands inside a transaction in the MySQLWorkbench window and verify the results in the putty window. Verify the results both before you do the command, after you do the command within the transaction, and then after you rollback the transaction.

   Your steps should look like:
   a. Show the data in the putty window
   b. Start a transaction in the MySQLWorkbench window : start transaction
   c. Execute one SQL action in the MySQLWorkbench window
   d. Show the data in the putty window
   e. Execute the “rollback” command in the MysQLWorkbench window
   f. Show the data in the putty window

2. The SQL commands to try will be a set of CRUD (create, read, update, delete) commands on the tables…well, without the read one since that doesn’t cause race conditions:
   a. Create a table
   b. Create a row of data in a table
   c. Add a column to an existing table
   d. Update a row of data in a table
   e. Update the data type of a column in a table
   f. Delete a row in a table
   g. Delete a column in a table
   h. Delete a table

3. Track and report the outcome from each of the SQL commands. The outcome to observe and report is if or when the changes from the MySQLWorkbench window become visible in the putty window, or if the changes never appear in the putty window.
4. Describe a set of actions you can do to verify whether or not you can have a transaction inside another transaction.

Try your set of actions and report on whether or not mysql will allow nested transactions.

5. Determine whether a transaction will survive beyond your session:
    a. Start a transaction in MySQLWorkbench and make a change to data in the table that you know the transaction will protect from other database users. Do not commit or roll-back the change.
    b. Exit MySQLWorkbench
    c. Restart MySQLWorkbench and return to your same test database. Remember to actually quit the application; don't just close the current query tab or window.
    d. Report if your change remains visible in the database past the restart of MySQLWorkbench.
    e. Report on whether you can make further changes to the same data as in step 5a after the restart.

Broadening question

1. Characterize the kinds of changes that a transaction can let you recover from versus the changes that a transaction doesn't let you recover from? Look for a characterization beyond itemizing the commands that do or don't allow recovery.

Reporting

2. In one file, list
    - The outcomes of each test operation in the lab steps
    - Your test for nesting of transactions and the outcome of the test
    - The results of your test of the transactions extending to a restart of MySQLWorkbench
    - Your answers to the broadening question.
3. Generate a PDF from the document.
4. Submit this PDF file to Brightspace.