Transactions Lab II

Tom Kelliher, CS 417

The purpose of this lab is for you to gain some understanding of how transactions work, see for yourselves how the various SQL isolation levels correspond to the ACID properties described in the textbook, and experiment with concurrent transactions to observe how they interact. "Pair-up" for this lab, using your project team as the "pair."

This is a graded lab. There are several boxed questions throughout the lab. Record your answers to these questions, using LibreOffice Writer or LaTeXon phoenix, or a similar piece of software. Number your answers using the subsection numbers shown in each box. Turn in hard copy of one answers document per pair. Record the names of the members of the pair at the beginning of the answers document.

Remember to either set `AUTOCOMMIT` off or to start every transaction with `BEGIN;` !!!

1 Locks

Continue to use the `movies` table for this Section. Work with your pair, using two `psql` sessions.

1.1 Simultaneous Updates

1. Design and run an experiment to determine what happens if two transactions try to update the same row of a table. Commit both transactions and view the table.

1.1. Describe your experiment. Describe and explain what happened.

1.2 Deadlock

1. Design and run an experiment to determine what happens if two transactions deadlock while attempting to update the same two rows. Attempt to commit each of the transactions and then view the table.

1.2. Describe your experiment. Describe and explain what happened.

1.3 Explicit Locking

1. Read the PostgreSQL documentation on the `SELECT` statement's `FOR` clause, which can be used to explicitly acquire shared and exclusive locks on a set of rows. Determine what happens when:

(a) Two transactions each attempt to acquire a shared lock on the same row.

(b) One transaction attempts to acquire a shared lock on a row, followed by a second transaction attempting to acquire an exclusive lock on the same row.

(c) One transaction attempts to acquire an exclusive lock on a row, followed by a second transaction attempting to acquire a shared lock on the same row.

(d) Two transactions each attempt to acquire an exclusive lock on the same row.

1.3. Describe and explain what happened.

2 Banking Transactions

In some of the following experiments, you'll be running the transactions "step-wise." The means to alternate the execution of individual SQL statements between transactions. You may find it easiest to use *three* `psql` sessions — two sessions started in single-step mode (`psql -s`) to run the two transactions and one session started without the `-s` switch that you can use to reset the `accounts` table between experiments by running the `accounts.sql` script.

2.1 The Transactions

Use the `accounts.sql` script to create and populate the `accounts` tables, which you'll use throughout this Section. Start by writing `.sql` scripts for the following transactions:

- `T1`: Display the sum of all the account balances.
- `T2`: Debit \$200 from account 4, then credit \$200 to account 5. (This simulates a transfer.)
- `T3`: Update the balance for account 4 by increasing it by 10%.
- `T4`: Debit \$200 from account 5, then credit \$200 to account 4. (This reverses `T2`'s transfer.)

I recommend starting each of these transaction scripts with the `BEGIN;` SQL command.

2.1. Include the SQL code for each of your transactions in your report.

2.2 Experiment One

1. Run `T2` and `T3` step-wise. Observe the state of the `accounts` table after both transactions commit.

2.2. What do you observe? Is there a difference between starting `T2` first versus starting `T3` first? Explain your observations.

2.3 Experiment Two

1. Run `T1`, recording the result.
2. Run `T3` in one session, but pause before it commits. Run `T1` to completion. Allow `T3` to commit.

2.3. What do you observe? Explain your observations.

2.4 Experiment Three

1. Run `T2` and `T4` step-wise. Observe what happens as the transactions execute, the result of each transaction committing, and the state of the `accounts` table after both transactions commit.

2.4.a. What do you observe? Is there a difference between starting `T2` first versus starting `T4` first? Explain your observations.

2. How could `T4` be rewritten so that correct behavior results? Revise `T4` and re-run the experiment.

2.4.b. How did you revise `T4`? What do you observe? Is there a difference between starting `T2` first versus starting `T4` first? Explain your observations.

2.5 Experiment Four

1. Revise `T1` so that it computes the sum of the balances twice.
2. Run `T1` in one session, but pause after it computes the sum of the balances the first time. Run `T3` to completion. Allow `T1` to commit.

2.5. What do you observe? Explain your observations. What concurrency problem does this illustrate?