



In this lab, you will need both the PHPMyAdmin web interface and the mysql client you used via PuTTY and login.fos.auckland.ac.nz during the first labs.

Note again that these lab sheets are not assessed, but they may be discussed in tutorials and their content is examinable!

Estimated time to complete this lab: 30 minutes.

A) Getting connected

Connect to the server TWICE, either by opening two PuTTY windows via login.fos.auckland.ac.nz, or one such window and a PHPMyAdmin session, where you'd select the SQL tab for the database.

We'll call the (first) PuTTY window "session 1" and the second PuTTY window (or PHPMyAdmin session) "session 2".

B) Creating a simple table

Using any of the sessions, create a simple table onto which we can unleash all the locking and transactions we wish and that we'll be able to drop and recreate as need be without losing any data:

```
CREATE TABLE Foo (x INT, y INT) ENGINE=InnoDB;
```

Now, put some data in:

```
INSERT INTO Foo VALUES (1, 9);  
INSERT INTO Foo VALUES (2, 8);  
INSERT INTO Foo VALUES (3, 7);
```

OK, we're ready now. Fasten seatbelts please!

C) Getting locked up step-by-step

Using session 1, issue the following statement:

```
LOCK TABLES Foo READ;
```

This gives session 1 the right to read undisturbed from the table. Nobody, including session 1, can now write to the table. Try:

```
INSERT INTO Foo VALUES (4, 6);
```

from session 1 and you will get an error message telling you politely that the table is locked with a READ lock. Now try:

```
SELECT * FROM Foo;
```

from both sessions and convince yourself that it works. Now issue, from session 2:

```
UNLOCK TABLES;
```

and try the insert again. It still doesn't work: The table was locked from session 1, and can only be unlocked from session 1. The unlock from session 2 has no effect.

Now enter

```
INSERT INTO Foo VALUES (4, 6);
```

from session 2. You will see that session 2 "blocks", i.e., it is waiting for the lock to be released before it completes the query. To us, it looks as if the session "hangs".

Now use session 1 to:

```
UNLOCK TABLES;
```

and you'll see the INSERT in session 2 complete all of a sudden. Convince yourself that you can insert again in the table from session 1, too.

Exercise: Issue, from session 1, the command:

```
LOCK TABLES Foo WRITE;
```

And convince yourself that the scenario remains the same, except that session 1 can also write to the table (e.g., INSERT).

D) Transaction in action

From session 1, issue the following statement:

```
START TRANSACTION;  
INSERT INTO Foo VALUES (11,99);
```

Convince yourself with

```
SELECT * FROM Foo;
```

from session 1 that the data shows in the table. This is because session 1 shows its uncommitted data, too.

Now try, this time from session 2, the same query:

```
SELECT * FROM Foo;
```

As our MySQL install runs with isolation level "repeatable read", the uncommitted changes will not show in session 2 ("repeatable read" implies "read committed").

Now issue, in session 1, the statement:

```
COMMIT;
```

Now, the new record will also be visible in session 2. Try it out!

E) One step forward, one back

From session 1, issue the following statement:

```
START TRANSACTION;  
INSERT INTO Foo VALUES (12,98);
```

Convince yourself with

```
SELECT * FROM Foo;
```

from session 1 that the new data shows in the table. Remember it's uncommitted data.

Now try, from session 1, the statement:

```
ROLLBACK;
```

followed by:

```
SELECT * FROM Foo;
```

This time, the uncommitted changes have been rolled back and the new data has disappeared from the table.

F) Getting it all mixed up (hopefully not!)

From session 1, issue the following statement:

```
START TRANSACTION;
```

```
COMPSCI 280 S1 2011
```

```
INSERT INTO Foo VALUES (13,97);
```

Convince yourself with

```
SELECT * FROM Foo;
```

from session 1 that the new data shows in the table. Remember again it's uncommitted data.

Now issue, this time from session 2, the statement:

```
INSERT INTO Foo VALUES (14,96);
```

Check with `SELECT * FROM Foo` in session 2 that this change has made it into the table. As session 2 isn't within a transaction, the change has "autocommitted". Note also that the pair (13,97) does not show in session 2 because it is an uncommitted `INSERT` at this point in time.

Now issue, from session 1, the statement

```
SELECT * FROM Foo;
```

The value pair (14,96) isn't there! This is because session 1, being in transaction isolation level "repeatable read", is working on a snapshot of the table as it was at the beginning of the transaction.

Now do a `COMMIT` in session 1, and a

```
SELECT * FROM Foo;
```

in both sessions. The tables look the same again. Note that the (13,97) shows before the (14,96) indicating that the RDBMS did indeed take notice of the time at which either record was first inserted (although, officially, this doesn't matter, of course!).