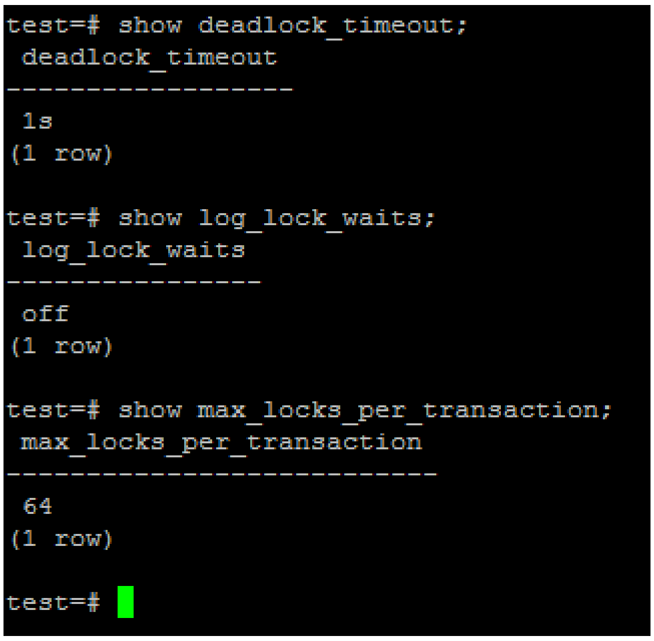
Christopher Ashley Cline
INLS 723: Databases III
Arcot Rajasekar
2012-11-26

Locking Homework

PostgreSQL Settings

- **deadlock_timeout**: 1s
- **log_lock_waits**: off
- **max_locks_per_transaction**: 64

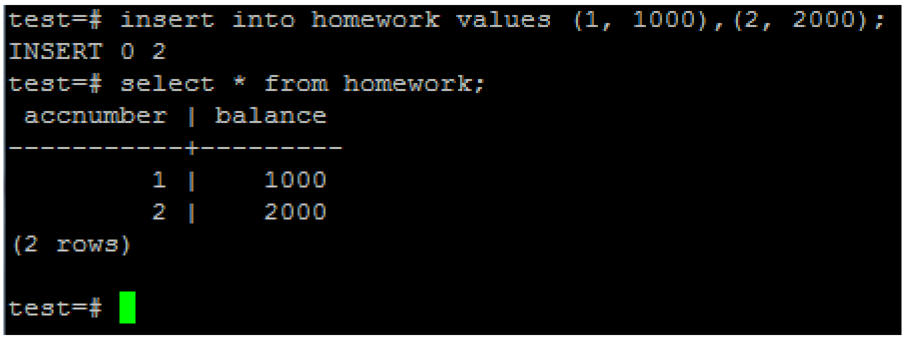


```
test=# show deadlock_timeout;
 deadlock_timeout
------------------
 1s
(1 row)

test=# show log_lock_waits;
 log_lock_waits
----------------
 off
(1 row)

test=# show max_locks_per_transaction;
 max_locks_per_transaction
---------------------------
 64
(1 row)

test=#
```

Transaction Testing

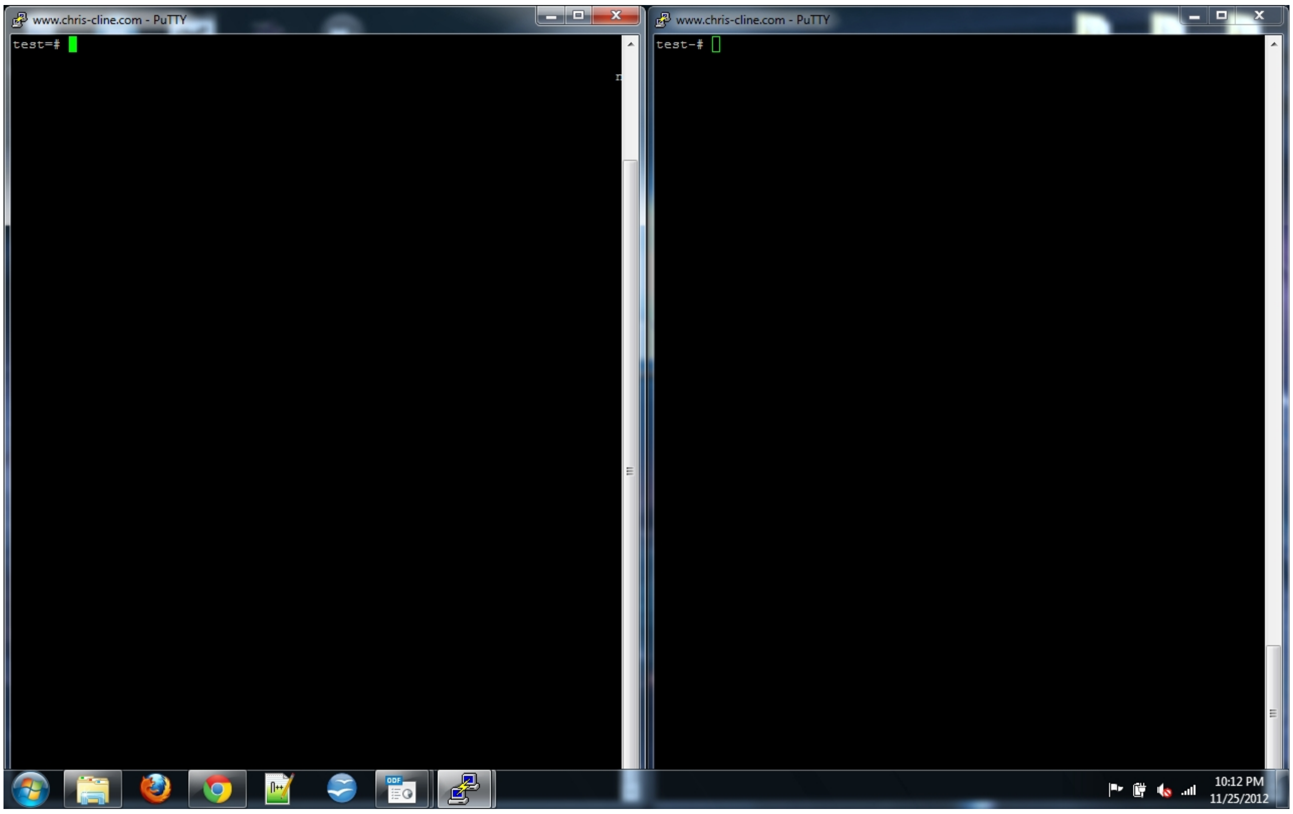
Setup

- Table Creation:



```
test=# insert into homework values (1, 1000),(2, 2000);
INSERT 0 2
test=# select * from homework;
 accnumber | balance
-----------+---------
         1 |    1000
         2 |    2000
(2 rows)

test=#
```

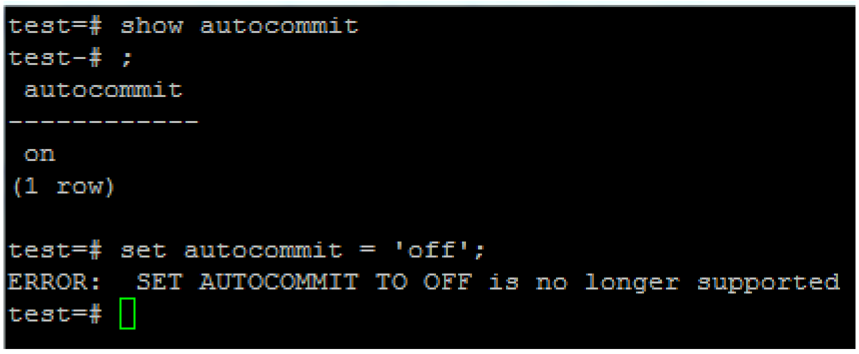
- Two Terminal Windows:



```
test=#
```

```
test-#
```

- Autocommit turned off:



```
test=# show autocommit
test-# ;
 autocommit
------------
 on
(1 row)

test=# set autocommit = 'off';
ERROR:  SET AUTOCOMMIT TO OFF is no longer supported
test=#
```

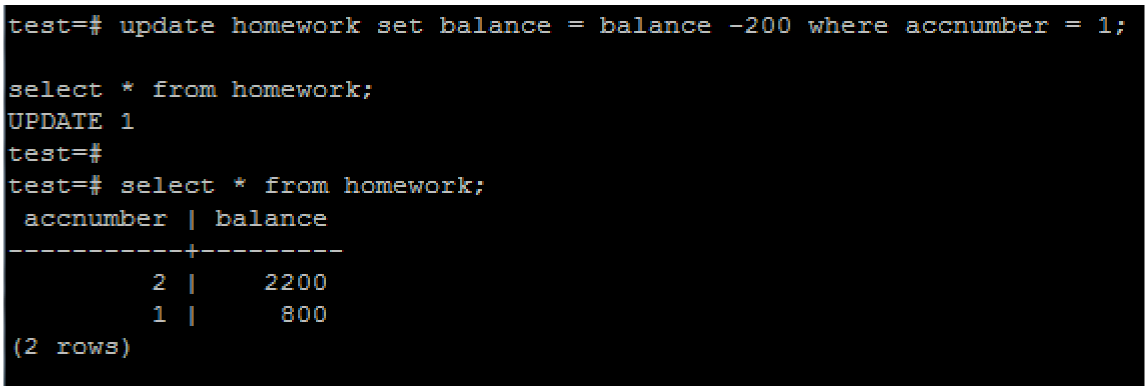
It seems that PostgreSQL no longer allows you to turn off autocommit.

- Alternative Solution: Instead of turning off autocommit, it should be possible to simulate the operations of two interleaved transactions by using *BEGIN* and *END* statements. According to PostgreSQL's online documentation and information found on discussion forums, another way to disable autocommit functionality is to create a *.psqlrc* file (which must reside on the accessing client machine) and create a configuration line which disables autocommit functionality.

Transaction 1	**Transaction 2**
wB1(+100)	
	wB2(+200)
	wB1(-200)
wB2(-100)	

Discussion

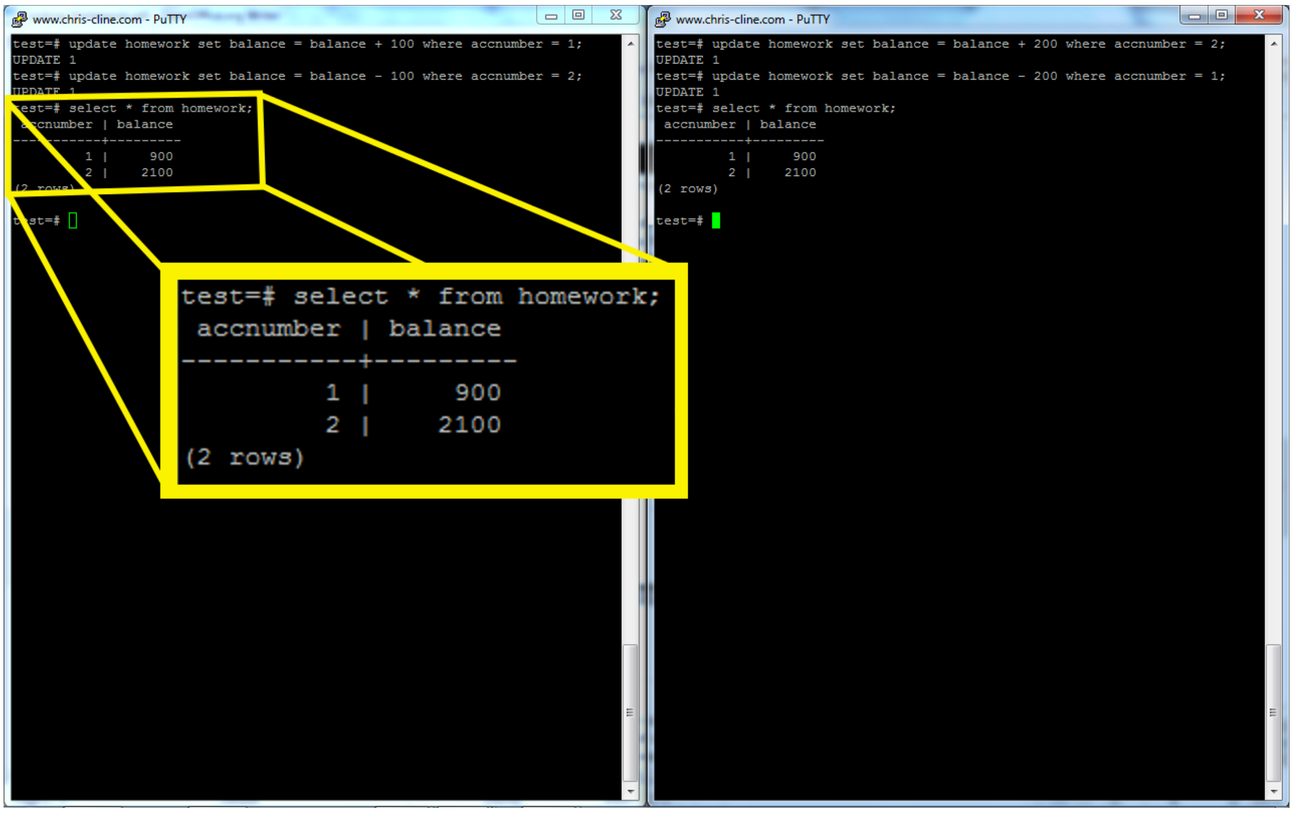
After starting both transactions, there is no issue in performing write operations until the second transaction attempted to write over the balance of account number one. Transaction two's terminal did not report a successful update and instead seemed to be waiting. Transaction one had already written to the balace field of account one, and therefore likely had a write lock on the field. Switching to transaction one, attempting to perform the last update operation resulted in a deadlock error. Entering the *END* command in both terminals results in transaction two committing while transaction one is rolled back. Even though transaction two is the first to attempt to write a value locked by another transaction, transaction one is rolled back because its second write operation would result in a deadlock situation. The end result, of course, is that only transaction two's write operations affect the table data:



```
test=# update homework set balance = balance -200 where accnumber = 1;

select * from homework;
UPDATE 1
test=#
test=# select * from homework;
 accnumber | balance
-----------+---------
         2 |    2200
         1 |     800
(2 rows)
```

Autocommit Reset Discussion

When operations are autocommited, the default situation, each operation is treated as an individual transaction, and every write operation is performed in a serialized fashion without conflict. The resulting table is different from that shown above:



```
test=# update homework set balance = balance + 100 where accnumber = 1;
UPDATE 1
test=# update homework set balance = balance - 100 where accnumber = 2;
UPDATE 1
test=# select * from homework;
 accnumber | balance
-----------+---------
         1 |     900
         2 |    2100
(2 rows)

test=#
```

```
test=# select * from homework;
 accnumber | balance
-----------+---------
         1 |     900
         2 |    2100
(2 rows)
```

```
test=# update homework set balance = balance + 200 where accnumber = 2;
UPDATE 1
test=# update homework set balance = balance - 200 where accnumber = 1;
UPDATE 1
test=# select * from homework;
 accnumber | balance
-----------+---------
         1 |     900
         2 |    2100
(2 rows)

test=#
```