Exercises for Hands-on 8: Databases

Make sure to fill in questions in the space provided underneath the question text.

Let's say that you have two terminals, and create two connections to the database. We will use two colors (blue and red) to indicate the two database sessions. In first terminal (blue), we list all of the accounts in the DB (the command select * from accounts; provides such a list).

```
username=> begin;
BEGIN
username=> select * from accounts;
 username |     fullname     | balance
----------+------------------+---------
 bitdiddl | Ben Bitdiddle    |      65
 alyssa   | Alyssa P. Hacker |      79
 jones    | Alice Jones      |      72
 mike     | Michael Dole     |      83
(4 rows)
```

Now, in the second (red) terminal, we start a transaction and add an account for Chuck (via an insert command):

```
username=> begin;
BEGIN
username=> insert into accounts values ('chuck', 'Charles Robinson', 55);
INSERT 0 1
username=>
```

Question 1. After adding the account for Chuck, we decide to generate a list of all accounts in the first (blue) terminal, and in the second (red) terminal. What output do you expect to get? Are they the same or not? Why? Format your answer as a table with rows and columns.

---

1) Blue will look exactly the same. The red terminal never committed the INSERT and neither has the blue, so it is not reflected in the DB yet. Isolation prevents these 2 terminals, in the middle of transactions, from seeing intermediate data.

```
username=> select * from accounts;
 username |     fullname     | balance
----------+------------------+---------
 bitdiddl | Ben Bitdiddle    |      65
 alyssa   | Alyssa P. Hacker |      79
 jones    | Alice Jones      |      72
 mike     | Michael Dole     |      83
(4 rows)
```

Question 2. Next, we commit the transaction in the second (red) terminal, by issuing the COMMIT statement:

```
username=> commit;
COMMIT
username=>
```

Now we go back to terminal 1 (blue) and regenerate the list of accounts (reproduced below, again by using select). You can see the terminal output below. Why is Chuck's account not there?

```
username=> select * from accounts;
 username |     fullname      | balance
--------------+-------------------+---------
 bitdiddl | Ben Bitdiddle     |      65
 alyssa   | Alyssa P. Hacker  |      79
 jones    | Alice Jones       |      72
 mike     | Michael Dole      |      83
(4 rows)
```

---

2) In this output, Chuck's information is still not available to the blue terminal. This is because the blue terminal has initiated an atomic transaction, which started before the INSERT transaction. Some DBs force updates after 1 process has committed, but here it is evident that the data available at the beginning of the blue terminal's transaction is the data available for the entire transaction. This provides stronger isolation between concurrent processes.

Question 3. Now we commit the transaction in the first (blue) terminal and generate a new list of all accounts again:

```
username=> commit;
COMMIT
username=> select * from accounts;
...
```

What output do you expect to see? Is it different than the output we received in question 2? Why or why not?

---

3) Now, we expect to see chuck's data in the DB. This is different from before, as before we were inside of an atomic transaction which was started before the insert transaction. Now, both transactions have ended, and both terminals can read the updated data. Specifically,

```
username=> select * from accounts;
 username |     fullname     | balance
----------+------------------+---------
 bitdiddl | Ben Bitdiddle    |      65
 alyssa   | Alyssa P. Hacker |      79
 jones    | Alice Jones      |      72
 mike     | Michael Dole     |      83
 chuck    | Charles Robinson |      65
(5 rows)
```

Now, let's try to modify the same account from two different transactions. In the first (blue) terminal, we start a transaction and deposit $5 into Mike's account:

```
username=> begin;
BEGIN
username=> update accounts set balance=balance+5 where username='mike';
UPDATE 1
```

In the second (red) terminal, we start a transaction and withdraw $10 from Mike's account:

```
username=> begin;
BEGIN
username=> update accounts set balance=balance-10 where username='mike';
UPDATE 1
```

Question 4. What would you expect to happen to the execution of the second update? Why?

---

4) The execution of the second update would have to wait. Since the blue terminal began first, it presumably acquired the lock first, and all of the red terminal's operations would be queued to execute as soon as blue commits/rolls back. So, nothing happens, and execution is halted in the second update.

Question 5. Let's say now we ABORT the command in the first (blue) terminal, undoing the $5 deposit:

```
username=> abort;
ROLLBACK
username=>
```

What would you expect would happen now? What happens to the second (red) terminal's transactions? What if we commit in the second (red) terminal? What would you expect to see in Mike's account?

---

5) Now, the blue terminal has ended their transaction in a rollback, undoing the previous operations. Since the terminal's transaction is over, it presumably releases the lock, and the red terminal can now begin the execution of its transaction. Then, if a commit follows from the red terminal, the $10 withdrawal is reflected in the database and the $5 deposit is not. Specifically:

```
username=> select * from accounts;
 username |     fullname     | balance
----------+------------------+---------
 bitdiddl | Ben Bitdiddle    |      65
 alyssa   | Alyssa P. Hacker |      79
 jones    | Alice Jones      |      72
 mike     | Michael Dole     |      73
 chuck    | Charles Robinson |      65
(5 rows)
```

Now let's say that we perform an atomic transfer of $15 from Ben to Alyssa, using two UPDATE statements in a single transaction. In the first (blue) terminal, we list the balances of all accounts. Prior to the update, we check the contents of the DB from the first (blue) terminal and get the following result. (The ? in the below table are placeholders for answers you received in previous questions.)

```
username=> select * from accounts;
 username |     fullname     | balance
----------+------------------+---------
 bitdiddl | Ben Bitdiddle    |      65
 alyssa   | Alyssa P. Hacker |      79
 jones    | Alice Jones      |      72
   ?      |       ?          |      ?
 mike     | Michael Dole     |      ?
(? rows)
```

Then we start a transaction and do a part of a transfer in the second (red) terminal:

```
username=> begin;
BEGIN
username=> update accounts set balance=balance-15 where username='bitdiddl';
UPDATE 1
username=>
```

Question 6. Now we finish the transfer by executing the following commands in the second (red) terminal. At which command do you expect to the effects of the second terminal's transaction to become visible in the first terminal. Why?

```
username=> update accounts set balance=balance+15 where username='alyssa';
UPDATE 1
username=> commit;
COMMIT
username=>
```

---

6) Again, updates made during a transaction are not reflected in the database to other clients until the commit point. The blue terminal would not see either alyssa's or ben's account updates until the commit was made, since they take place in an _atomic_ transaction.
Once the red terminal's `commit;` propagates to the database, both updates get reflected in the DB.