TRANSACTIONS and LOCKING

For each of the following tasks, you have to save on a text file snapshots of your sessions where we can watch:

- the statements executed, and
- the answer the server gives you.

You must identify in which session (session 1 or session 2) has been run each statement. Show your snapshots in a table like this:

Sesion 1	Sessin 2

`mysql> PROMPT s1>`
 	
`s1> SET GLOBAL TRANSACTION`
`-> ISOLATION LEVEL READ COMMITTED;`
 	
`s1> SELECT @@global.tx_isolation;`
`+-----------------------+`
`| @@global.tx_isolation |`
`+-----------------------+`
`| READ-COMMITTED |`
`+-----------------------+`	

	`mysql> PROMPT s2>`
	
	`s2> START TRANSACTION;`
	
	`s2> INSERT INTO City`
	`-> (Name, CountryCode,Population)`
	
	`-> VALUES ('Sakila', 'SWE', 1);`

`s1> SELECT Name, CountryCode`
`-> FROM City`
`-> WHERE Name = 'Sakila';`
`Empty Set (0.0 sec)`	

IMPORTANT (no exercise will be accepted without this): change your mysql prompt for each session:

- in session 1: **`prompt [\D] <your name> s1>`**
- in session 2: **`prompt [\D] <your name> s2>`**

Example:

```
mysql> prompt [\D] alicia s1>
PROMPT set to '[\D] alicia s1>'
[Mon May 17 18:11:33 2021] alicia s1>
[Mon May 17 18:11:36 2021] alicia s1>
[Mon May 17 18:11:37 2021] alicia s1>select now();
+---------------------+
| now()               |
+---------------------+
| 2021-05-17 18:11:59 |
+---------------------+
1 row in set (0.00 sec)

[Mon May 17 18:11:59 2021] alicia s1>
```

EXERCISES

1. In session 1, create the following table (on any DB we have used in class).

```
create table numbers (
     number int PRIMARY KEY
) ENGINE=INNODB;
```

   - Insert several numbers (1,2,3,4,5) in the "**numbers**" table.
   - Disable **autocommit** mode, that is, autocommit = 0.
   - Insert the number 6 and check that it is not visible in session 2 until **commit** it.

2. What level of isolation you must use in session 2 for reproducing the following?:

   In session 1 (where autocommit = 0) run the insertion of number 7. In session 2 we can see this record inserted even before executing the commit in session 1.

3. Execute a **rollback** in session 1 (where you have run the insertion) and check you can see the change in both sessions.

4. What level of isolation you have to use in session 1 to reproduce the following? (REMARK: before changing the level of isolation execute a commit to finish any opened transaction)

   - Run the following query in session 1:

     `select max(number) from numbers;` → retrieve 6

   - Begin a transaction in session 2 and run the following insertion:

     `insert into numbers values(7);`

     `select max(number) from numbers;` → retrieve 7

   - In session 1:

     `select max(number) from numbers;` → retrieve 6

   - Run a `commit` in session 2.

   - In session 1:

     `select max(number) from numbers;` → retrieve 7

     `delete from numbers where number = 7; commit;`

5. What level of isolation you have to use in session 1 for reproducing the following?
    - In session 1:

      `select max(number) from numbers;` → retrieve 6
    - Begin a transaction in session 2 and run the following insertion:

      `insert into numbers values (7);`

      `select max(number) from numbers;` → retrieve 7.
    - Run a `commit` in session 2.
    - In session 1:

      `select max(number) from numbers;` → retrieve 6

      `commit;`

      `select max(number) from numbers;` → retrieve 7
    - In session 2:

      `delete from numbers where number = 7;`

6. Cause a “time out” on a transaction.
    - Run an update on a table row in a transaction in session 1.
    - Run another update on the same table row in a transaction in session 2. Check that remains waiting and returns a "time out". Check also that this attempt to change the row is not executed.

7. Cause a “dead lock”.
    - Run an update on a table row in a transaction in session 1.
    - Run another update on a different row in a transaction in session 2.
    - Run an update on the table row changed in session 1, but now in session 2.
    - In session 1, try to update the row changed (the first one) in session 2.

    Check the message returned (*`deadlock found`*) and that a rollback is executed (in which session?)