Assignment: Transactions and concurrency control

Goal:

The goal of this assignment is to learn to work with transactions and isolation levels.

Instructions:

This is an individual assignment. The deliverables are specified in the end of the assignment.

- Checking the isolation levels:
  `SELECT @@GLOBAL.tx_isolation, @@tx_isolation;`
- Setting the session isolation level (to **READ_UNCOMMITTED** in the example below):
  `SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;`
  The command affects the new transactions.
- Setting the server isolation level (to **READ_UNCOMMITTED**):
  `SET GLOBAL TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;`
  The command affects the new database connections.
- See https://mariadb.com/kb/en/set-transaction/

The database creation script is located in Oma workspace folder **Documents/Data**.

All the tasks can be done with **HeidiSQL**. First, connect to your database server.

Create the bank database and take it into use:

```
DROP DATABASE IF EXISTS Bank;

CREATE DATABASE Bank;

USE Bank;
```

Download the database creation script **Bank_script.sql** and run it. Get acquainted with the resulting database.

First, check the isolation levels and set the server's and the session's isolation levels to **REPEATABLE_READ** if they are something else.

Tasks:

1. In MariaDB you can execute a transaction by giving the operation commands between `START TRANSACTION;` and `COMMIT;` commands as follows:

   ```
   START TRANSACTION;
   SQL-command 1;
   SQL-command 2;
   …
   ```

```
COMMIT;
```

Write and execute a transaction that transfers 150 euros form account 100 to account 103. The transaction consists of two **UPDATE** commands. Check the final balances of accounts 100 and 103 with **SELECT** commands. What are the balances?

2. Repeat the previous, but instead of committing, cancel the transaction with `ROLLBACK;` command. What happens? What are the final balances?

3. Open simultaneously two MariaDB sessions in their own command windows: in HeidiSQL click **File/New window** and create another connection to the database server.

   In the first session write a similar transaction as previously, which transfers 200 euros from account 101 to account 102, but do not commit the transaction. Check the balances in the second session. What do you observe?

4. Now commit the transaction in the first session. Check the balances in both sessions once again. What do you observe?

5. Let's check if phantom reads are possible. Now create two transactions, each in its own connection:

   1. TR1: empty the account number 105 (balance becomes 0)
   2. TR2: Read the balance of account 105. Read the balance of account 105. Read the balance of account 105. Read the balance of account 105.

   Write the operations in different windows in the following order. Pay special attention to the results of reading in rows 3, 5 and 7.

   1. Start TR1.
   2. Start TR2.
   3. TR2: Read balance of account 105. *Result?*
   4. TR1: Decrease the balance of 105 to zero.
   5. TR2: Read balance of account 105 *Result?*
   6. Commit TR1.
   7. Read balance of account 105 *Result?*
   8. Commit TR2.

   What do you observe? How is the reading of uncommitted transaction at row 5 handled?

6. For both connections, change the isolation level to **READ UNCOMMITTED**.

   Change the balance of account 105 to 1000 euros, and then repeat the transactions in task 5 with the new isolation level. How does the action differ from the previous case?

7. Change the isolation level of both sessions to the safest level (**SERIALIZABLE**).

   Restore again the balance of 105 to 1000 euros and repeat the transactions in task 5. What happens?

8. Repeat the previous task so that when transaction TR1 starts waiting, you do not commit transaction TR2, but let time pass at least for 50 seconds. What happens? What is the balance of 105 at the end? Why?

Deliverables

Submit a pdf document that provides answers to the answers specified in the tasks. Add screenshots and document your results in sufficient detail so that it is possible to verify what you have done