NYU Tandon School of Engineering
CS-UY 3083-B
Professor Salim Arfaoui

Assignment 5: Transactions

Due: 11:59pm, Sunday, April 20, 2025

This assignment covers the following topics:

- **SQL**
- **Transactions**

Submission instructions

- You should submit your assignment on Gradescope.
- For this assignment you should turn in 1 pdf file:
  - `(100 pts) Transactions_report.pdf`

For this assignment,

*you may work individually or with at most 3 partners **(max group size = 4)**. We strongly encourage you to work with partner(s) as different query designers typically have different perspectives. Besides, you can learn from each other.*

Please remember that the NYU Code of Conduct is in effect.

Submission and Grading:

- *This assignment is made up of a set of questions collectively worth 100 points.*
- *Homework must be submitted on time.*
- *Late submissions will not be accepted.*
- ***Include all team member names and NetIDs in the report.***
- *Save your report as a PDF (**Transactions_report.pdf**).*
- *Each team submits only **one** copy.*
- *Upload your report as a PDF to the **Assignment 5 on Gradescope**. Make sure you connect your partner to your group on Gradescope so that everyone receives credit.*
- *Making your submission available to the course staff is **your** responsibility; if we cannot access or open your file, we have to assign a zero grade. Be sure to test access to your file before the due date.*

Transactions

To deal with concurrent operations, many SQL databases, including MySQL, support transactions, which allow several SQL statements to execute atomically (often meaning both *before-or-after atomicity* and *all-or-nothing atomicity*). To execute statements as a single transaction, SQL clients first issue a **`START TRANSACTION`** command, then execute some SQL commands, and finally issue either a **`COMMIT`** command, which makes the transaction's changes permanent, or a **`ROLLBACK`** command, which reverts the changes from all of the commands in the transaction.

In this part, you will simulate concurrent database queries by issuing SQL statements over two different connections to the database.

Exercise 1:

A university payment system, which handles payments using ID cards, might construct a table that stores information about accounts, such as the account holder's username, their full name, and the total amount of money in their account. You can visualize the table as containing the following information, with 4 rows and 3 columns:

As a first step, create a new database (**`techDB`**) then use the **`SQL CREATE`** command to create the table shown below, and then execute several **`INSERT`** commands to insert each of the rows into the resulting table.

`accounts:`

username	fullname	balance
jones	Alice Jones	82
bitdiddle	Ben Bitdiddle	65
mike	Michael Dole	73
alyssa	Alyssa P. Hacker	79

Exercise 2:

In this part, you will simulate concurrent database queries by issuing SQL statements over two different connections to the database. **Open up two terminals**, and use the database that contains the `accounts` table in each of them, to create two connections to the database.

We will use two colors (`blue`🔵 and `red`🔴) to indicate the two database sessions.

2.1) Start a transaction in the first (**blue**) terminal, and display a list of all accounts.

2.2) Now, in the second (**red**) terminal, also start a transaction and add an account for Chuck:

```
'chuck', 'Charles Robinson', 55
```

2.3) Generate a list of all accounts in the first (**blue**) terminal, and in the second (**red**) terminal. ***What output do you get? Are they the same or not? Why?***

2.4) Now, commit the transaction in the second (**red**) terminal, by issuing the `COMMIT` statement.

2.5) Generate a list of all accounts from the first (**blue**) terminal again. ***Does it include Chuck? Why or why not?***

2.6) Commit the transaction in the first (**blue**) terminal and generate a new list of all accounts.

What output do you get? Is it different from the output you received in Step 2.5 ? Why or why not?

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

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

2.9) ***What happens to the execution of the second update in step 2.8? Why?***

2.10) Let's try aborting a transaction: enter the `ABORT` command in the first (**blue**) terminal, undoing the $5 deposit:

2.11) What happens to the second (red) terminal's transaction?

2.12) If you now commit the transaction in the second terminal, ***what is the resulting balance in Mike's account?***

2.13) Now let's perform an atomic transfer of $15 from Ben to Alyssa, using two UPDATE statements in a single transaction. In the first (**blue**) terminal, list the balances of all accounts. Then start a transaction and do a part of a transfer in the second (**red**) terminal.

2.14) If you now look at the list of all account balances in the first (**blue**) terminal, ***have the results changed compared to before you started the transaction in the second (red) terminal?***

2.15) Finish the transfer by executing the rest of the operations in the second (**red**) terminal and `COMMIT`. After each command, list all of the account balances in the first (blue) terminal, to see at what point the effects of the second terminal's transaction become visible. ***What is that point, and why?***

What to submit?

- *Perform all the operations outlined above.*
- ***Submit the answers to the questions colored in green.***

Include your answers in the same pdf as part2 (`Transactions_indexing_report.pdf`)