Middle East Technical University | Department of Computer Engineering

CENG 352

Database Management Systems

Spring 2023-2024
Project 2

Due date: May 22nd, 2024, Wednesday, 23:59

1 Project Description

In this project you are asked to develop a simple e-commerce application (e.g. Trendyol, Amazon, etc.). This is a typical application using a database. You will import data to a PostgreSQL database. Database details will be the similar to the previous project, with additions.

Your platform will allow sellers to reach their clients with products. Firstly, you will construct the database and import the data for this database. After those, you will create an application to allow sellers and customers to use the service.

There are 3 important restrictions of this application:

- For each seller, there exists a limit for maximum number of parallel sessions allowed. This number is determined by **the plan to which the seller is subscribed**. Each seller is subscribed to exactly one plan. Once the number of sessions of a seller reaches the maximum, you will deny sign in requests (sign in command) until some sessions are terminated by signing out (sign out command)
- For each customer, there exists a limit for maximum weight per order, which is **15 kilograms**. Once this limit is reached within a shopping cart, you will deny add to cart requests (change_cart command). This application only serves to the individual customers, any order exceeding 15 kilogram is considered as corporate deal.
- When customers decide to add items to the cart or purchase the cart items, you need to validate if the product's stock is available for purchase for particular seller.

2 Database

2.1 Creating the database

The database name will be *ceng352_2023_hw2* for this mini project. You will have 2 steps while creating the database:

- Create tables and insert sample data by running SQL queries in **construct_db.sql**.
- Import data from .csv files under *data.zip* directory. Import the data as-is, without doing any kind of data processing or cleaning. Consider column delimiter as semicolon (,), consider quote character as quote (") and fill empty values with NULL while importing data to tables.

After these steps, you will get these tables in the database:

```
product_category
    category_id, name

products
    product_id, name, category_id, weight, price

customers
    customer_id, name, surname, address, state, gender

orders
    order_id, customer_id, order_time, shipping_time, status

shopping_carts
    order_id, product_id, seller_id, amount

sellers
    seller_id, password, session_count, plan_id

plans
    plan_id, name, max_parallel_sessions

stocks
    seller_id, product_id, stock_count
```

You should keep these information in your mind:

- *Sellers* need to be authenticated to use the application. But the customers don't have to be authenticated, customer-side commands can be called anonymously.
- *Sellers* table holds the seller's current subscription plan. *session_count* is the number of active sessions of sellers. This number is incremented when a seller signs in to the service and decremented when seller signs out.
- *Plans* represents the available plans to subscribe for sellers. You should insert at least 3 plans with different *max_parallel_sessions* values. **max_parallel_sessions is a value to allow sellers to have at most N connections at any moment.**
- *Stocks* represents the number of stocks available for a particular product, on a particular seller's storage. When a new shopping cart is created (order status = CREATED), it should behave regarding the available stocks of the seller.

- *Shopping carts* have now *seller_id* information within the entries. This will allow you to manipulate the seller's stocks.

  If max_parallel_sessions = 2, then the seller who is subscribed to this plan can connect to the platform with at most 2 devices at the same time.

  All sellers must have a plan and one seller is dedicated to one plan only. You are free to invent your own plans.

3 The Software

For helpful tutorials and links, check these websites: **link 1**, **link 2**, **link 3**.

After the database is created, you will implement a simple program for the sellers to connect to the platform. For simulating multiple device connections, you can open multiple terminal windows and sign in from those terminals.

For simplicity, the platform will be a command line-like software written in Python3. You will use **psycopg2** to do PostgreSQL tasks like insert, update, read etc. These are the source files inside **source** directory:

- **main.py:** This file is for retrieving inputs, processing requests and sending response messages back to the user. **main()** function in this file acts like a service layer for the software. You should not modify this file, you will run the software like:

  ```
  >_ python main.py
  ```

  How this software works will be explained shortly.

- **mp2.py: You will ONLY modify this file.** You will implement the functions in this file and make sellers available to sign in, sign out and read contents from the database. You will return success message or error messages, depending on the situation.

- **validators.py:** This file is for validating commands. You don't have to worry about validation, it is written for you and you can try entering incorrect commands to break the program.

- **seller.py:** This file contains the Seller class, which will be used to store currently authenticated seller information.

- **messages.py:** You should **ONLY** use and return messages defined in this file, to get points from this assignment.

- **database.cfg:** This file contains database configurations. Change them with your configurations to connect the database on your local machine.

When the program starts, list of commands are shown and the program waits for command input:

```
_> python main.py

*** Please enter one of the following commands ***
> help
> sign_up <seller_id> <password> <plan_id>
> sign_in <seller_id> <password>
> sign_out
> show_plans
> show_subscription
> change_stock <product_id> <add or remove> <amount>
> subscribe <plan_id>
> ship <order1_id> <order2_id> ... <orderN_id>
> show_cart <customer_id>
> change_cart <customer_id> <product_id> <seller_id> <add or remove> <amount>
> purchase_cart <customer_id>
> quit
ANONYMOUS >
```

At first, there is no signed in seller. Seller is shown as ANONYMOUS and you can only call *help, sign_up, sign_in, quit* commands with anonymous seller. *help* command will remind you what are the available commands provided by this software.

If you implement *sign_in* command correctly and sign in with an existing seller information, the look will change to authenticated seller like below:

```
ANONYMOUS > sign_in <seller_id> <password>
OK
<seller id>>
```

Authenticated sellers can use the other commands too. These commands are available to authenticated sellers only.

4 Tasks

There are programming and written tasks. You should find

```
#TODO: Implement this function
```

comments in the code and replace them with your own implementations for programming tasks. For written tasks, you are required to write a simple report.

4.1 Programming Tasks (72 pts)

4.1.1 Sign Up (6 pts)

```
>_ sign_up <seller_id> <password> <plan_id>
```

You need to implement *sign_up()* function of Mp2Client inside mp2.py. This is the command to create a new seller. The anonymous seller enters seller information (seller id, password) and id of the plan that the new seller will be subscribed to. You should create a new seller with provided information in the database.

When you complete the implementation, the software should give output like this:

```
ANONYMOUS > sign_up johnwick pass123 1
OK
```

If a seller with same seller id exists before, or the program gets any kind of exception during the execution, you should give an error message like this (assuming you have already a seller with 'johnwick'):

```
ANONYMOUS > sign_up johnwick pass123 1
ERROR: Can not execute the given command.
```

4.1.2 Sign In (6 pts)

```
>_ sign_in <seller_id> <password>
```

You need to implement *sign_in()* function of Mp2Client inside mp2.py. This is the command for a seller to sign in to the service. The seller types in seller credentials. You should implement required authentication and session management logic using your seller database.

Remember to increment *session_count* inside subscription table for the seller. Also check whether the seller is out of sessions or not by looking at *max_parallel_sessions* of the seller's plan.

Successful sign in operation should look like this:

```
ANONYMOUS > sign_in johnwick pass123
OK
johnwick >
```

Failed sign in operation should look like this:

```
ANONYMOUS > sign_in johnwick pass123
ERROR: Seller id or password is wrong.
ANONYMOUS >
```

If **session_count** >= **max_parallel_sessions**, then the output should look like this:

```
ANONYMOUS > sign_in johnwick pass123
ERROR: You are out of sessions for signing in.
ANONYMOUS >
```

For simulating multiple device connections, you can open multiple terminal windows and sign in from those terminals.

4.1.3 Sign Out (6 pts)

`>_ sign_out`

You need to implement *sign_out()* function of Mp2Client of Mp2Client inside mp2.py. This is the command for an authenticated seller to sign out from the service. You should implement required authentication and session management logic using your seller database.

Remember to decrement *session_count* inside sellers table for the seller. Also check whether the seller's *session_count* can be at least 0.

Successful sign out operation should look like this:

```
johnwick > sign_out
OK
ANONYMOUS >
```

Failed sign out operation should look like this:

```
johnwick > sign_out
ERROR: Can not execute the given command.
johnwick >
```

4.1.4 Show Plans (3 pts)

`>_ show_plans`

You need to implement *show_plans()* function of Mp2Client inside mp2.py. This is the command to get list of all available plans to subscribe. This command does not have any parameters. You should print all columns of the plans table in the database (e.g. plan id, name, etc.). Follow the pattern below while printing columns:

When there is an authenticated seller, show_plans operation should look like this:

```
johnwick > show_plans
#|Name|Max Sessions
1|Basic|2
2|Advanced|4
3|Premium|6
```

4.1.5 Show Subscription (3 pts)

`>_ show_subscription`

You need to implement *show_subscription()* function of Mp2Client inside mp2.py. This is the command to get the details of the plan to which the authenticated seller is subscribed. This command does not have any parameters. Printing should be similar with the show_plans command.

When there is an authenticated seller, show_subscription operation should look like this:

```
johnwick > show_subscription
#|Name|Max Sessions
1|Basic|2
```

4.1.6 Change Product Stock (9 pts)

```
>_ change_stock <product_id> <operation_type> <amount>
```

You need to implement *change_stock()* function of Mp2Client inside mp2.py. This is the command to change the stock of a product in stocks table. The operation type can be either 'add' or 'remove'. For instance, if the operation type is 'add' and the amount is '10', then you need to increase the stock. If the operation type is 'remove', then you need to decrease the stock.

When there is an authenticated seller, change_stock operation should look like this. Imagine the current stock is 6, then add 2 or remove 5 operations should print and the stock should be 3 at the end:

```
johnwick > change_stock productX add 2
OK
johnwick > change_stock productX remove 5
OK
johnwick >
```

Here is an important rule for changing stock operation: you need to check if the stock amount after the operation is below 0, when the operation type is 'remove'. **The stock value can not be below 0.**

Imagine the current stock is 6, then remove 12 operations should print an error:

```
johnwick > change_stock productX remove 12
ERROR: Can not execute the given command.
johnwick >
```

4.1.7 Subscribe To A New Plan (6 pts)

```
>_ subscribe <new plan's id>
```

You need to implement *subscribe()* function of Mp2Client inside mp2.py. This is the command for authenticated seller to subscribe to another plan. You must ensure that sellers will not subscribe to a new plan with less parallel sessions allowed. You should update the subscription information for the authenticated seller on the seller database.

Assume that *johnwick* has a plan with id=1 and *max_parallel_sessions*=2 and wants to subscribe to a plan with id=2 and *max_parallel_sessions*=4. This operation can be done since new *max_parallel_sessions* is greater than old one. If *max_parallel_sessions* values are same, you should also allow that operation too.

```
johnwick > show_subscription
#|Name|Max Sessions
1|Basic|2
johnwick > subscribe 2
OK
johnwick > show_subscription
#|Name|Max Sessions
2|Advanced|4
```

However, you must reject the command if new *max_parallel_sessions* is smaller than old one. When we try to change johnwick's plan with the old plan id, return SUBSCRIBE_MAX_PARALLEL _SESSIONS_UNAVAILABLE message:

```
johnwick > subscribe 1
ERROR: New plan's max parallel sessions must be greater than or equal to
current plan's max parallel sessions.
johnwick > show_subscription
#|Name|Max Sessions
2|Advanced|4
```

4.1.8 Ship Orders (9 pts)

```
>_ ship <order1_id> <order2_id> ... <orderN_id>
```

You need to implement *ship()* function of Mp2Client inside mp2.py. This is the command to save that sellers shipped orders with given order ids, the stock amounts gets decreased on stocks table. **This command does not require sign in at all.**

```
johnwick > ship orderX orderY orderZ
OK
```

There are rules about this command:

1. You should save **ALL** new stock amounts to the stocks table, or **NONE** of them in case of an error during saving the stocks to the table. The errors could be; non-existing order, insufficient amount of stocks etc.
2. The order status should be changed to SHIPPED and the shipping datetime should be placed for the order.
3. The same stock rules apply from the *change_stock* functionality. If a product is not in stock, then you should halt **whole** shipment and do nothing.

4.1.9 Quit (3 pts)

```
>_ quit
```

You need to implement *quit()* function inside mp2.py. This is the command to quit the application. **You have to sign out before quitting if there is an authenticated seller.** Manage *session_count* column properly.

4.1.10 Show Customer Cart (3 pts)

```
>_ show_cart <customer_id>
```

You need to implement *show_cart()* function of Mp2Client inside mp2.py. This is the command to get list of items that the customer decides to buy. This command has a customer_id parameter. You should print all results from the *shopping_carts* table, with the order id of the 'CREATED' order. Follow the pattern below while printing columns:

```
johnwick > show_cart customerX
Order Id|Seller Id|Product Id|Amount
orderX|sellerX|productX|3
orderX|sellerX|productY|1
orderX|sellerY|productZ|4
```

In Mp2, there can only be 1 order with status='CREATED' for each customer. This is to indicate shopping carts on the fly, the customer can add-remove items to this specific order. The order status will remain as 'CREATED' until the items in the shopping cart is **purchased**.

Another important note about customer-side operations is that they can be called any time in the application. **They do not require sign in at all.**

4.1.11 Change Customer Cart (9 pts)

```
>_ change_cart <customer_id> <product_id> <seller_id> <operation_type> <amount>
```

You need to implement *change_cart()* function of Mp2Client inside mp2.py. This is the command to add/remove items to the shopping cart. This command has customer_id, product_id, seller_id, operation_type amount parameters. You should add/remove given amount of items to the shopping cart, for an order with status='CREATED'. If the item exists on the cart, then you should increment/decrement it with given value. Suppose there are 2 items for 'productX' in the cart:

```
johnwick > change_cart customerX productX sellerX add 3
OK
```

This command will make it 5 items in the cart.

```
johnwick > change_cart customerX productX sellerX remove 1
OK
```

This command will make it 4 items in the cart.

If the operation is 'add', then you also need to check if that amount of items exist in the stocks of the seller. If that is not the case, return STOCK_UNAVAILABLE message.

```
johnwick > change_cart customerX productX sellerX add 30
ERROR: Not enough stocks.
```

More things to consider:

- If there is NO shopping cart defined for the customer before, then you should create a new order with random uuid and status = 'CREATED'. After this, you can use the new order_id. **Do NOT put values to order_time and shipping_time, since this order is for having a temporary, on the fly, shopping cart.**
- If the new amount will be <= 0, you should remove the entry from the shopping_carts table. **Keep the order with status='CREATED' for that customer.**

- If the weight limit (15 kilograms) is reached for the new order, return WEIGHT_LIMIT message. Product weights are in kilograms in the database.

```
johnwick > change_cart customerX productX sellerX add 300
ERROR: Weight limit is exceeded for the order.
```

4.1.12 Purchase Cart (9 pts)

```
>_ purchase_cart <customer_id>
```

You need to implement *purchase_cart()* function of Mp2Client inside mp2.py. This is the command to buy items in the shopping cart. This command has a customer_id parameter. You should change the order time & status to 'RECEIVED' for the customer's newly 'CREATED' order and reduce the seller stocks after this operation.

```
johnwick > purchase_cart customerX
OK
```

You also need to check if that amount of items exist in the stocks of the seller. If that is not the case, return STOCK_UNAVAILABLE message and do NOT change any stocks for any of the items in the shopping cart.

```
johnwick > purchase_cart customerX
ERROR: Not enough stocks.
```

If there is no order with status='CREATED' or no entries for such order in shopping cart table, return EMPTY_CART message.

```
johnwick > purchase_cart customerX
ERROR: There are no items to purchase.
```

4.2 Written Tasks (28 pts)

For tasks below, write down your answers in a file and submit it as a PDF report.

4.2.1 Transaction Isolation Levels (9 pts)

If you were to define transactions in the database level, then how would you define transactions for each of the actions in programming tasks?

For instance, you can say "I would use the isolation level ... and the access mode ... for ... action, because ...". Brief explanations are enough.

4.2.2 Transaction Isolation Levels Experiment (10 pts)

For reading the available subscription plans from plans table, submit 2 scripts named **mp2_transaction_reader.py** and **mp2_transaction_writer.py**. For both reader and writer, set a transaction isolation level as READ COMMITTED, REPEATABLE READ and SERIALIZABLE.

Writer Python script should insert a new plan to the plans table and commit changes. Reader Python script should display plans before writer script commits and after writer script commits.

To imitate sleeps, or long running processes, you can simply put an *input("Hit enter to continue")* statement before commit & after commit. Similarly you can put the same statement between reads from plans table.

Run the reader and the writer on separate terminal windows and observe the output for different isolation levels. Comment about your observations, you will be graded for both scripts and your comments.

4.2.3 Indexes (9 pts)

Write the query for finding statistics about the sellers and the number of products they sold in a particular product category for each month. Introduce indexes to make this query run faster. And report the execution plans with and without your indexes using **"EXPLAIN"**. Compare time differences.

5 Regulations

1. **Implementation:** Implement mp2.py only. If you think you need helper functions for the tasks, please write them to mp2.py. **You should NOT update any other files in the source directory.**
2. **Response messages:** Check messages.py and comments on the functions for response messages to be used. **Different messages for responses are NOT ALLOWED.**
3. **Submission:** Submission will be done via ODTUClass. Please follow the late submission policy announced at the beginning of the semester. You should put your **mp2.py** implementation and your report file (.pdf, .txt) inside a .zip file with following name:

   ```
   e1234567_mp2.zip
       -> mp2.py
       -> mp2_transaction_reader.py
       -> mp2_transaction_writer.py
       -> report PDF
   ```

   Where you should replace "1234567" with your own student number. Please make sure that the .zip file doesn't contain any subdirectories, it should only contain mp2.py and the report file.
4. **Newsgroup:** You must follow ODTUClass for discussions and possible clarifications on a daily basis.