Middle East Technical University | Department of Computer Engineering

CENG 352

Database Management Systems

Spring 2022-2023
Project 2

Due date: June 7, 2023, 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 same as the previous project.

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 following sign in requests (sign in command) until some sessions are terminated by signing out (sign out command).

For each seller, there exists a limit for maximum stock per product. This number is determined by the plan to which the seller is subscribed. Once the number of product stocks of a seller reaches the maximum, you will deny following stock update requests (change_stock command). The stock amount should be between 0 and the maximum limit for the seller's plan.

The stock limit for sellers also applies for the customers. When they decide to add items to the cart or purchase the cart items, you need to validate the seller_stock table entries.

2 Database

2.1 Creating the database

The database name will be **"ceng352_2022_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 following tables in the database:

```
product_category
        product_category_name
        product_category_name_english

customers
        customer_id
        customer_unique_id
        customer_zip_code_prefix
        customer_city
        customer_state

geolocation
        geolocation_zip_code_prefix
        geolocation_lat
        geolocation_lng
        geolocation_city
        geolocation_state

order_payments
        order_id
        payment_sequential
        payment_type
        payment_installments
        payment_value

order_items
        order_id
        order_item_id
        product_id
        seller_id
        shipping_limit_date
        price
        freight_value

orders
        order_id
        customer_id
        order_status
        order_purchase_timestamp
```

```
        order_approved_at
        order_delivered_carrier_date
        order_delivered_customer_date
        order_estimated_delivery_date

sellers
        seller_id
        seller_zip_code_prefix
        seller_city
        seller_state


products
        product_id
        product_category_name
        product_name_length
        product_description_length
        product_photos_qty
        product_weight_g
        product_length_cm
        product_height_cm
        product_width_cm


order_reviews
        review_id
        order_id
        review_score
        review_comment_title
        review_comment_message
        review_creation_date
        review_answer_timestamp

seller_subscription
        seller_id
        subscriber_key
        session_count
        plan_id

subscription_plans
        plan_id
        plan_name
        max_parallel_sessions
        max_stock_per_product

seller_stocks
        seller_id
```

```
product_id
stock_count
```

You should keep following 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 any time in the application. You can assume that seller only gets imaginary "notifications" for customer actions.
- **Seller_subscription** 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 he/she signs out.
- **Subscription_plans** represents the available plans to subscribe for sellers. You should insert at least 3 plans with different **max_parallel_sessions** and **max_stock_per_product** values. **"max_parallel_sessions"** is a value to allow sellers to have **at most N** connections at any moment. **"max_stock_per_product"** is a value to limit sellers to have **at most K** products in stock.
  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.

  If max_stock_per_product = 12, then the seller who is subscribed to this plan can store at most 12 units of a product. This rule applies for all products.

  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
  ```

  only. 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> <subscriber_key> <zip_code> <city> <state> <plan_id>
> sign_in <seller_id> <subscriber_key>
> sign_out
> show_plans
> show_subscription
> change_stock <product_id> <add or remove> <amount>
> show_quota
> subscribe <plan_id>
> ship <product_id_1> <product_id_2> <product_id_3> ... <product_id_n>
> calc_gross
> 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 test_seller_id <subscriber_key>
OK
test_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 (80 pts)

4.1.1 Sign Up (5 pts)

```
>_ sign_up <seller_id> <subscriber_key> <zip_code> <city> <state> <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, subscriber key, zip code, city, state) and id of the plan that the new seller will be subscribed to (plan id). You should create a new seller with provided information in the database. You need to call multiple SQL queries to achieve this functionality.

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

```
ANONYMOUS > sign_up johnwick pass123 70740 brasilia DF 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 70740 brasilia DF 1
ERROR: Can not execute the given command.
```

4.1.2 Sign In (7 pts)

```
>_ sign_in <seller_id> <subscriber_key>
```

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 seller_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 subscriber key 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 (5 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 subscription_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|Max Stocks Per Product
1|Basic|2|4
2|Advanced|4|8
3|Premium|6|12
```

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|Max Stocks Per Product
1|Basic|2|4
```

4.1.6 Change Product Stock (7 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 seller_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 max_stock_per_product = 10, and the current stock is 6, then add 2 or remove 5 operations should print:

```
johnwick > change_stock 92bf5d2084dfbcb57d9db7838bac5cd0 add 2
OK
johnwick > change_stock 92bf5d2084dfbcb57d9db7838bac5cd0 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 above max_stock_per_product or not when the operation type is 'add'. Also you need to check if the stock amount after the operation is below 0 or not when the operation type is 'remove' . **The stock value can not be below 0 or above max_stock_per_product.** The seller needs to change his/her plan to pass the max_stock_per_product limit.

Imagine max_stock_per_product = 10, and the current stock is 6, then add 6 or remove 12 operations should print an error:

```
johnwick > change_stock 92bf5d2084dfbcb57d9db7838bac5cd0 add 6
ERROR: Can not update the stock count.
johnwick > change_stock 92bf5d2084dfbcb57d9db7838bac5cd0 remove 12
ERROR: Can not update the stock count.
johnwick >
```

If there is no product with given product id, reject with not found message:

```
johnwick > change_stock some_product_id add 6
ERROR: Product is not found.
```

4.1.7 Show Stock Quota (5 pts)

```
>_ show_quota
```

You need to implement **show_quota()** function of Mp2Client inside **mp2.py**. This is the command to get the remaining stock quota for all products **in the stock**. This command does not have any parameters.

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

```
johnwick > show_quota
Product Id|Remaining Quota
92bf5d2084dfbcb57d9db7838bac5cd0|2
```

If the seller did not put any products to the stock, it should give this message instead:

```
johnwick > show_quota
Quota limit is not activated yet.
johnwick >
```

4.1.8 Subscribe To A New Plan (5 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 **old_max_parallel_sessions** <= **new_max_parallel_sessions**. If max_parallel_sessions values are same, you should also allow that operation too.

```
johnwick > show_subscription
#|Name|Max Sessions|Max Stocks Per Product
1|Basic|2|4
johnwick > subscribe 2
OK
johnwick > show_subscription
#|Name|Max Sessions|Max Stocks Per Product
2|Advanced|4|8
```

However, you must reject the command if **old_max_parallel_sessions** > **new_max_parallel_sessions**. When we try to change johnwick's plan with the old plan id, following happens:

```
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|Max Stocks Per Product
2|Advanced|4|8
```

If there is no plan with given plan id, reject with not found message:

```
johnwick > subscribe 99
ERROR: Plan is not found.
```

Finally, note that subscribing to a new plan with higher max_stock_per_product allows seller to increase the stocks for the products. So, show_quota will behave differently when you have higher max_stock_per_product. For simplicity, there won't be a case that the new plan has lower max_stock_per_product than the current one.

4.1.9 Ship Products (10 pts)

```
>_ ship <product 1 id> <product 2 id> <product 3 id> ... <product N id>
```

You need to implement **ship_products()** function of Mp2Client inside **mp2.py**. This is the command to save the fact that the authenticated seller has just shipped a product or products. The seller types in ids of the products and the stock amounts gets decreased on seller_stocks table.

```
johnwick > ship 1e9e8ef04db5 6a2fb4dd53d004c 47859fca9dd7474f
OK
```

There are rules about this command:

1. You can send 1, 2, 3, ... product ids as parameters to this command. Your implementation should handle multiple product ids as arguments.
2. You should save **ALL** new stock amounts to the seller_stocks table, or **NONE** of them in case of an error during saving the stocks to the table.
3. The product ids will be distinct, but you should not allow shipping unknown products with product ids that does not exist on the database.
4. The same stock rules apply from the change_stock functionality. If the product is not in stock, then you should halt the shipment.

An example for rule 2 and 3:

```
johnwick > ship 1e9e8ef04db5 6a2fb4dd53d004c test
ERROR: Can not execute the given command.
```

4.1.10 Calculate Gross Income Per Month (3 pts)

```
>_ calc_gross
```

You need to implement **calc_gross()** function of Mp2Client inside **mp2.py**. This is the command to get the gross income of the seller per month by looking at the rows of the order_items and orders tables. Use **order_purchase_timestamp** for year and month. This command does not have any parameters.

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

```
johnwick > calc_gross
Gross Income|Year|Month
123.45|2018|1
67.80|2018|2
johnwick >
```

If the seller hasn't sold any products, it should give this message instead:

```
johnwick > calc_gross
Gross Income: 0
johnwick >
```

4.1.11 Quit (3 pts)

```
>_ quit
```

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

4.1.12 Show Customer Cart (3 pts)

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 customer_carts table. Follow the pattern below while printing columns:

```
johnwick > show_cart 4f2d8ab171c80ec8364f7c12e35b23ad
Seller Id|Product Id|Amount
5b51032eddd242adc84c38acab88f23d|c777355d18b72b67abbeef9df44fd0fd|3
dd7ddc04e1b6c2c614352b383efe2d36|e5f2d52b802189ee658865ca93d83a8f|2
df560393f3a51e74553ab94004ba5c87|ac6c3623068f30de03045865e4e10089|1
```

The 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. That is why they have "customer_id" parameter.

If there is no customer with given customer id, reject with not found message:

```
ANONYMOUS > show_cart test
ERROR: Customer is not found.
johnwick > show_cart test
ERROR: Customer is not found.
```

4.1.13 Change Customer Cart (10 pts)

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 cart. If the item exists on the cart, then you should increment/decrement it with given value. Suppose there are 2 items for 'c777' in the cart:

```
johnwick > change_cart 4f2d8ab c777 add 3
OK
```

This command will make it 5 items in the cart.

```
johnwick > change_cart 4f2d8ab c777 remove 1
OK
```

This command will make it 4 items in the cart.

If there is no customer with given customer id, reject with not found message. If there is no product with given product id, reject with not found message:

```
ANONYMOUS > change_cart test c777 3
ERROR: Customer is not found.
johnwick > change_cart 4f2d8ab test 3
ERROR: Product is not found.
```

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 4f2d8ab c777 add 1
ERROR: Not enough stocks.
```

Finally, if the new amount will be <= 0, you should remove the entry from the customer_carts table.

4.1.14 Purchase Cart (11 pts)

You need to implement **purchase_cart()** function of Mp2Client inside **mp2.py**. This is the command to buy items to the shopping cart. This command has a customer_id parameter. You should clear the customer cart and reduce the seller stocks after this operation. In addition, you should put entries to the orders and order_items table. Leave other fields with dummy values, possibly NULL. The necessary fields are:

```
orders
    order_id
    customer_id
    order_purchase_timestamp

order_items
    order_id
    order_item_id
    product_id
    seller_id

johnwick > purchase_cart 4f2d8ab
OK
```

If there is no customer with given customer id, reject with not found message:

```
ANONYMOUS > purchase_cart 4f2d8ab
ERROR: Customer is not found.
```

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 > purchase_cart 4f2d8ab
ERROR: Not enough stocks.
```

4.2 Written Tasks (20 pts)

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

4.2.1 Transaction Types (10 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 Indexes (10 pts)

Find 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"**.

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
       -> report file
   ```

   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.