Database

Assignment 3: Hanyang Baedal Application Development

Student Name: Bruno Carvacho Yerkovich

Student Number: 2018000300

Index

① Introduction and Goal

Database has become one most used Computer Software areas in different fields such as Education, Medicine, Shopping and so on. Likewise, the urgent use of new technologies and data filtering has engaged developers to keep working on reliable databases that could fulfill the companies and customers' conditions. Moreover, understand the functioning process of a database and its components is a must in terms of how the database will behave and adapt itself to the requirements pre-established in the beginning of the developing process. One of the emerging areas that uses databases as it working method is food delivery service. The main reason is because databases let the developer have freedom when storing big amounts of data and use fast algorithms to handle data processing and simplify orders and users' information.

The goal of this project corresponds to the development of a Database application for delivery service and product filtering with different kind of functionalities such as customers information, seller's products, and restaurants as well as menu data. The application will be developed using Python programming language, SQL and PgAdmin for data visualization and results. It is important to mention as well that the working environment includes a pre-loaded database provided by our teacher assistants. The data will be analyzed, filtered for each of the requirements and used as indicated in the instructions ppt. However, the program does not handle specific input errors as it only detects if the desired action is in the list of available options in the program.

② Theory and Specifications

In terms of theory, Databases are introduced in this project as part of the main developing tool for each of the desired functionalities established at the beginning of the goals in the assignment. Moreover, specific libraries such as “psycopg2” are part of the server connecting process to handle the main database of our program and establish a link between Python and PgAdmin. SQL on the other hand will be the bridge working language to maintain our database updated and not only send commands from Python to the Database but also to keep it secure and safe from external errors that could eventually occur in a hypothetical situation when the program is used by real users.

The program’s structure was developed by following the PgModeler graph designed in the first assignment of this class including the entities and functionalities previously indicated in the homework. Therefore, the program will count with a seller, store, customer, and delivery entities, each of them separated by their respective files with special conditions designed to handle the users’ input. The program will contain then, specific lines of SQL language casted in Python to process the desired piece of data in each case. Finally, the program is developed by getting the user’s input as arguments predefined in the assignment. For this reason, it important for the user to type the correct input every time to avoid errors that could arise due to wrong input information.

③ Code Explanation

3.1 Seller.py

Seller.py corresponds to the code that indicates all the seller's functionalities for the program as well as its pieces of code to handle the database related to this entity.

I) Imported Libraries:

```
1   import time
2   import argparse
3   from helpers.connection import conn
```

The imported libraries are strictly related to the overall application's running time, argument parsing when the user inputs information as the program runs and finally the connecting server for sending and getting data from the database when is necessary in each of the cases created for the seller entity.

II) Show Seller Info:

```
6
7   def showSellerInfo(sellerInfo):
8       print("---------------------------")
9       email = str(sellerInfo[0][3]) + "@" + str(sellerInfo[0][4])
10      print("Name: " + str(sellerInfo[0][1]))
11      print("Phone: " + str(sellerInfo[0][2]))
12      print("Email: " + str(email))
13      print("---------------------------")
14
```

We start our file with the definition of the "Show Seller info" method, which has the task of printing the user's information when changes are made in the database such as adding new sellers or completely removing them from our database.

The method will take as argument the seller's information also known as the entity's information extracted from the main database. Then, the first and last line of the method indicates the delimiter separators when using the method and have an organized and well-printed statement when running the program. The next lines indicate the user's information that will be printed on the screen, this includes the user's name, phone, and email. In the case of the email however, a conjunction between two pieces of data from the two-dimensional array were necessary to get a fully working email on the go.

III) Update Seller Info:

```
16  def updateSellerinfo(args, cur):
17      sellerId = args.property[0]
18      property = args.property[1]
19
20      if property == 'name':
21          print("CHANGE NAME!!")
22          newName = args.property[2]
23          sqlUpdate = "UPDATE seller SET name=%(name)s WHERE id=%(id)s;"
24          cur.execute(sqlUpdate, {"name": newName, "id": sellerId})
25          conn.commit()
26
```

The next method is the "Update Seller Info" function. This method aims to help the user to change the seller's information from the original database without struggling with complicated formularies and just type the necessary information that we need to change to get the user's desired data. The seller's id will be represented as the first element in the arguments' property array whereas the property variable will be the second one in the list. After this data assignment part, we encounter our program to enter the conditions section to check the property indicated by the user when the program started.

The first condition created was the name changing option. For this option we will run an SQL line of code to update the seller's table by re-indicating the new name of a specific seller found by the id typed by the user. The new name is assigned by using the arguments' array created in the beginning of the program. After running the SQL line of code, the information is updated by using the commit command.

```
elif property == 'phone':
    print("CHANGE PHONE!!")
    newPhone = args.property[2]
    sqlUpdate = "UPDATE seller SET phone=%(phone)s WHERE id=%(id)s;"
    cur.execute(sqlUpdate, {"phone": newPhone, "id": sellerId})
    conn.commit()
```

The second condition is to change the user's phone number. This condition follows a similar method as the one used in the change name condition with a slightly difference in the SQL line as now, we are looking for setting a new phone and not a name.

```
34      elif property == 'email':
35          newLocal = args.property[2]
36          newDomain = args.property[3]
37          sqlUpdate = "UPDATE seller SET local=%(local)s WHERE id=%(id)s;"
38          cur.execute(sqlUpdate, {"local": newLocal, "id": sellerId})
39          conn.commit()
40          sqlUpdate = "UPDATE seller SET domain=%(domain)s WHERE id=%(id)s;"
41          cur.execute(sqlUpdate, {"domain": newDomain, "id": sellerId})
42          conn.commit()
```

For this part of the code, two entities were necessary to be changed as the email address is composed by two elements: local and domain. Therefore, I decided to run two different SQL line

codes to assign each of the necessary parameters for the new seller. Of course, after making these changes, a commit call was mandatory to apply our changes.

```
elif property == 'password':
    print("CHANGE PASSWORD!!")
    newPass = args.property[2]
    sqlUpdate = "UPDATE seller SET passwd=%(passwd)s WHERE id=%(id)s;"
    cur.execute(sqlUpdate, {"passwd": newPass, "id": sellerId})
    conn.commit()

else:
    print("ERROR! WRONG OPTION!")
    return 0
```

The last two conditions were necessary to change the user's password or for simply printing an error message if the wrong option was chosen from the user's side.

IV) Update Seller Info:

```
def main(args):
    id = str(args.id)

    try:
        cur = conn.cursor()
        sql = "SELECT * FROM seller WHERE id=%(id)s;"
        cur.execute(sql, {"id": args.property[0]})
        rows = cur.fetchall()

        if id == 'info':
            showSellerInfo(rows)

        elif id == 'update':
            updateSellerinfo(args, cur)
            cur.execute(sql, {"id": args.property[0]})
            newRows = cur.fetchall()
            print("NEW INFO --> " + str(newRows))
        else:
            parser.print_help()
    except Exception as err:
        print(err)
```

The final method in the seller's file is the definition of the main function. During this method we will parse the arguments provided by the user and call the corresponding method. Two options are provided, if the id corresponds to "info" then we print the seller's information by calling the Show Seller's info method. Otherwise, if the choice is "update" then we update the seller's information with the provided new info.

3.2 Store.py

I) Imported Libraries:

```
import time
import argparse
from helpers.connection import conn
from tabulate import tabulate

```

For this case, I imported one extra library compared to the last file to print the content in a more elaborated and special way than we usually do in Python.

II) Show Store Info Method:

```
def showStoreInfo(storeInfo):
    print(storeInfo)
    print("\nStore #" + str(storeInfo[0][0]) + "'s Info:")
    print("--------------------------------------------------")
    print("Store's Name: " + str(storeInfo[0][2]))
    print("Store's Address: " + str(storeInfo[0][1]))
    print("Store's Phone Numbers: ")
    for phoneNumber in storeInfo[0][5]:
        print("#" + str(storeInfo[0][5].index(phoneNumber)) + " --> " + phoneNumber)
    print("Store Seller's ID: " + str(storeInfo[0][7]))
    print("Store's longitude: " + str(storeInfo[0][3]) + " \nStore's Latitude: " + str(storeInfo[0][4]))
    print("Store's Schedule:")
    print(tabulate(storeInfo[0][6], headers="keys", tablefmt='fancy_grid', missingval=' '))
    print("--------------------------------------------------")
```

The first method will print all the store's information in an organized manner showing all the necessary data stored in our server. The method uses as main argument a simple variable defined by the user during the input process. After this, we start printing first the store number, name, address, phone numbers (inside a for loop), store seller's id, store's longitude, latitude, and finally the store's schedule using the tabulate method included in the tabulate library imported at the beginning of the file.

III) Store Menu Method:

```
def storeMenu(storeInfo, cur):
    print("Store #" + str(storeInfo[0][0]) + "'s Menu:")
    print("-------------------------------------------")
    sqlStore = "SELECT * FROM Menu AS Me, Store AS S WHERE Me.sid = S.id AND S.id=%(id)s;"
    cur.execute(sqlStore, {"id": storeInfo[0][0]})
    rows = cur.fetchall()
    for row in rows:
        print("#" + str(rows.index(row) + 1) + ". Menu ID: " + str(row[0]) + ", Menu: " + str(row[1]))

    print("-------------------------------------------")
```

This method will rapidly print the store's menu by first running an SQL command that finds the desired store by using the id provided by the user. After running this command, we parse the information in a for loop and print it as indicated in the ppt instructions.

IV) Add Menu:

```
def addMenu(args, storeInfo, cur):
    #print(args.property[1])
    sqlMax = "SELECT MAX(id) FROM menu;"
    cur.execute(sqlMax)
    maxId = cur.fetchall()
    maxId = maxId[0][0]
    maxId += 1
    #print(maxId)
    sqlAdd = "INSERT INTO menu(menu, sid) VALUES (%(Menu)s, %(Sid)s);"
    cur.execute(sqlAdd, {"Menu": args.property[1], "Sid": args.property[0]})
    conn.commit()
    storeMenu(storeInfo, cur)
```

The next method to define was the Add Menu functionality. With this method we can add a new menu to the menus list defined at the beginning of the program. We take as parameters the arguments provided by the user, Store Information, and the cur (connection) variable. After getting the variables, we define an SQL code sentence to get the highest number in the list of ids

to assign it to the new menu. After this process, we then run another SQL code to insert the new menu and its respective ID.

Finally, the last two steps are to commit the new created content and print the store information to show to the user the applied changes to our database.

V) Main Method

```
def main(args):
    id = str(args.id)

    try:
        cur = conn.cursor()
        sql = "SELECT * FROM store WHERE id=%(id)s;"
        cur.execute(sql, {"id": args.property[0]})
        rows = cur.fetchall()

        if id == 'info':
            showStoreInfo(rows)

        elif id == 'menu':
            storeMenu(rows, cur)

        elif id == 'add_menu':
            addMenu(args, rows, cur)

        else:
            parser.print_help()
    except Exception as err:
        print(err)
```

The main method in this case will find three different options: “info”, “menu” and “add_menu”. All the parameters will be defined inside the arguments provided by the user and later on executed with the cur (connection). It is important to mention though that if the wrong option is typed by the user, then the program will show an error message on the screen to handle the problematic situation.

3.2 Customer.py

I) Imported Libraries:

```
import time
import argparse
from helpers.connection import conn
from tabulate import tabulate
```

The imported libraries in this file were the same as the ones in the Store.py file as we need to do fancy print in here as well.

II) Info Check Parser:

```
def InfoCheck(parser:argparse.ArgumentParser):
    sub_parsers = parser.add_subparsers(dest='function')

    # Informacion del ciente. Agregarla en detalle a la tabla.
    parser_info = sub_parsers.add_parser('info')
    parser_info.add_argument('id', type=int)

    # Direccion exacta del cliente.
    parser_address = sub_parsers.add_parser('address')
    parser_address.add_argument('id', type=int)
    parser_address_mode = parser_address.add_mutually_exclusive_group()
    parser_address_mode.add_argument('-c', '--create')
    parser_address_mode.add_argument('-e', '--edit', nargs=2)
    parser_address_mode.add_argument('-r', '--remove')

    # Sistema de pago del cliente.
    parser_pay = sub_parsers.add_parser('pay')
    parser_pay.add_argument('id', type=int)
    parser_pay_mode = parser_pay.add_mutually_exclusive_group()
    parser_pay_mode.add_argument('--add-card', type=int)
    parser_pay_mode.add_argument('--add-account', nargs=2)
    parser_pay_mode.add_argument('-r', '--remove', type=int)

    # Sistema de busqueda para el cliente.
```

This info Check Parser will be the main method used for information filtering in this file. The method includes not only a well-defined structure but also easy commands that can be identified through the code while filtering and assigning information for each of the variables used in the program. I created different kind of variables for each of the possible cases to simplify the conditioning process in the main method.

III) Customer Info Method:

```
def customerInfo(contenido):
    cur = conn.cursor()
    sql = "SELECT * FROM customer WHERE id=%(id)s;"
    cur.execute(sql, {"id": contenido.id})
    rows = cur.fetchall()
    print(rows)
    print("\nCustomer #" + str(contenido.id) + "'s Information:")
    print("-------------------------------------")
    print("Name: " + str(rows[0][1]))
    print("Phone: " + str(rows[0][2]))
    print("Email: " + str(rows[0][3]) + "@" + str(rows[0][4]))
    print("Password: " + str(rows[0][5]))
    print("Payments:\n" + tabulate(rows[0][6], headers="keys", tablefmt='fancy_grid'))
    print("Latitude & Longitude: " + str(rows[0][7]) + "/" + str(rows[0][8]))
    print("-------------------------------------")
```

Here, we print in a well-organized manner all the customer's information stored in our database: Name, Phone, Email, Password, Payments, Latitude and Longitude. We then use again the tabulate function for printing the customer's payments in order. We always fetch all our information by executing an SQL line of code.

IV) Customer Address Method:

```
def customerAddress(contenido):
    cur = conn.cursor()
    sql = "SELECT * FROM address WHERE cid=%(id)s;"
    cur.execute(sql, {"id": contenido.id})
    rows = cur.fetchall(); rows = sorted(rows)
    print("\nCustomer #" + str(contenido.id) + "'s Address Information:")
    print("-----------------------------------------------------")
    for row in rows:
        print(str(row[0]) + ". " + row[1])
    print("-----------------------------------------------------")
```

Taking as parameters the user's content, we encounter our program to execute a SQL line of code that will select all customer address that have the indicated id stored in their data. We then parse and print the information with a for loop.

V) Customer Payment Method:

```
81  def customerPaymentMethod(contenido):
82      cur = conn.cursor()
83      sql = "SELECT * FROM customer WHERE id=%(id)s;"
84      cur.execute(sql, {"id": contenido.id})
85      rows = cur.fetchall()
86      print("\nCustomer #" + str(contenido.id) + "'s Payment methods:\n")
87      print(tabulate(rows[0][6], headers="keys", showindex="always", tablefmt='fancy_grid'))
88
```

The next method to define is the Customer Payment method. Like the other methods, it executes an SQL line of code to find the desired customer to then use its information to print all the stored payment methods from this customer. As we are not making any changes in the database, we will not use any commit function for this method.

VI) Main Method (Part 1)

```
if contenido.function == "info":
    customerInfo(contenido)

elif contenido.function == "address":
    print("CONTENIDO --> " + str(contenido))
    if contenido.create is not None:
        cur = conn.cursor()
        sqlMax = "SELECT MAX(id) FROM address;"
        cur.execute(sqlMax)
        maxId = cur.fetchall()
        maxId = maxId[0][0]
        if maxId is None:
            maxId = 0
        maxId += 1
        sqlAdd = "INSERT INTO address(direccion, id, cid) VALUES (%(Direccion)s, %(Did)s, %(Cid)s);"
        cur.execute(sqlAdd, {"Direccion": contenido.create, "Did": maxId, "Cid": contenido.id})
        conn.commit()
        customerAddress(contenido)

    elif contenido.edit is not None:
        cur = conn.cursor()
        sqlEdit = "UPDATE address SET direccion = (%(Direccion)s) WHERE cid = %(Cid)s AND id = %(Did)s;"
        cur.execute(sqlEdit, {"Direccion": contenido.edit[1], "Cid": contenido.id, "Did": contenido.edit[0]})
        conn.commit()
        customerAddress(contenido)

    elif contenido.remove is not None:
        cur = conn.cursor()
        sqlEdit = "DELETE FROM address WHERE cid = %(Cid)s AND id = %(Did)s;"
        cur.execute(sqlEdit, {"Cid": contenido.id, "Did": contenido.remove[0]})
        conn.commit()
        customerAddress(contenido)
    else:
        customerAddress(contenido)
```

In order to have a reliable interface that can run all necessary conditions stipulated in the instructions ppt, I had to create a special parser with numerous comparing conditions that could simplify the process and go to the desired method. Then, we start first by checking if our content is part of the “address” condition. If it is so, we then check if the user wants to create, edit, or delete an address from the customer’s table. The process is straightforward and calls three different SQL lines of code to accomplish the objective in each of the tasks. It is important to mention though that in case of adding and address, we need to locate first the maximum ID to

follow an order and add the address to the correct index. On the other hand, we always proceed to apply changes after we run our SQL line either for adding, editing, or removing an address.

VII) Main Method (Part 2)

```
elif contenido.function == "pay":
    print(contenido)
    if contenido.add_card is not None:
        newCard = {"data": {"card_num": None}, "type": "card"}
        (newCard["data"])["card_num"] = contenido.add_card
        cur = conn.cursor()
        sql = "SELECT customer.payments FROM customer WHERE customer.id=%(id)s;"
        cur.execute(sql, {"id": contenido.id})
        rows = cur.fetchall(); rows = sorted(rows)
        rows[0][0].append(newCard)
        rowsNew = str(rows[0][0]).replace("\'", "\"")
        sqlAdd = "UPDATE customer SET payments = (%(Payments)s) WHERE id = %(Cid)s;"
        cur.execute(sqlAdd, {"Payments": rowsNew, "Cid": contenido.id})
        conn.commit()
        customerPaymentMethod(contenido)
    elif contenido.add_account is not None:
        newAccount = {"data": {"acc_num": None}, "type": "account"}
        (newAccount["data"])["acc_num"] = contenido.add_account
        cur = conn.cursor()
        sql = "SELECT customer.payments FROM customer WHERE customer.id=%(id)s;"
        cur.execute(sql, {"id": contenido.id})
        rows = cur.fetchall();
        rows = sorted(rows)
        rows[0][0].append(newAccount)
        rowsNew = str(rows[0][0]).replace("\'", "\"")
        sqlAdd = "UPDATE customer SET payments = (%(Payments)s) WHERE id = %(Cid)s;"
        cur.execute(sqlAdd, {"Payments": rowsNew, "Cid": contenido.id})
        conn.commit()
        customerPaymentMethod(contenido)
```

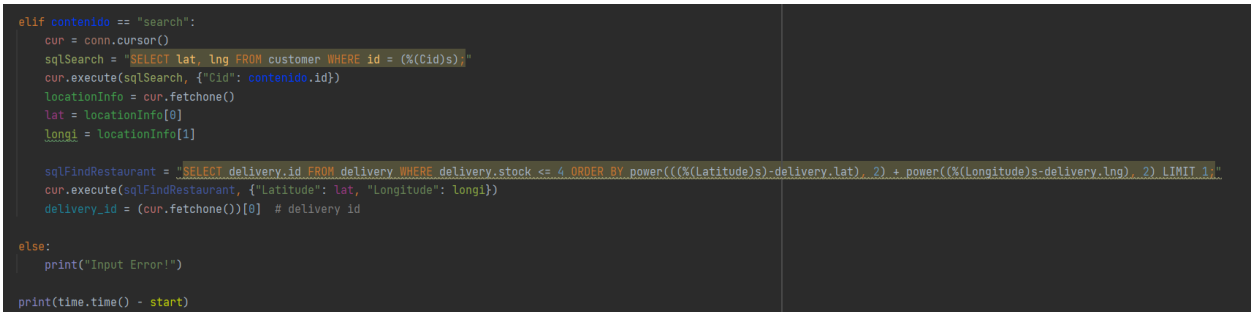
We then jump to the next condition for checking the customer's pay method. Here, the first two options are to add a new card or account. The situation turned a little bit different here as we had to come up with something that was not present in the rest of the conditions in the program. Therefore, the solution was to have a card and account pattern that we could fill up with the user's input and add it to the list of information already stored in the database. I had also to

reformat the array as the original slashes would not match the new ones after adding a new card or bank account.

```
elif contenido.remove is not None:
    cur = conn.cursor()
    sql = "SELECT customer.payments FROM customer WHERE customer.id=%(id)s;"
    cur.execute(sql, {"id": contenido.id})
    rows = cur.fetchall();
    rows = sorted(rows)
    print(rows[0])
    print(len(rows[0][0]))
    rows[0][0].pop(contenido.remove)
    rowsNew = str(rows[0][0]).replace("\'", "\"")
    print(len(rows[0][0]))
    print(rows[0])
    cur = conn.cursor()
    sqlAdd = "UPDATE customer SET payments = (%(Payments)s) WHERE id = %(Cid)s;"
    cur.execute(sqlAdd, {"Payments": rowsNew, "Cid": contenido.id})
    conn.commit()
    customerPaymentMethod(contenido)
else:
    customerPaymentMethod(contenido)
```

To finish with the payment method, we encounter our program to extract the payment method we want to remove by scanning all the payments with the ID indicated by the user. After doing this, we then sort our payments array, we pop the element with the desired position, and finally update the original list with the one that does not contain the old payment method previously deleted. Lastly, if we are not making any changes in the original list of payments, then we call the Customer Pay Method to show an organized list with all the customer's payment methods added to our original database.

VIII) Main Method (Part 3)



```
elif contenido == "search":
    cur = conn.cursor()
    sqlSearch = "SELECT lat, lng FROM customer WHERE id = (%(Cid)s);"
    cur.execute(sqlSearch, {"Cid": contenido.id})
    locationInfo = cur.fetchone()
    lat = locationInfo[0]
    longi = locationInfo[1]

    sqlFindRestaurant = "SELECT delivery.id FROM delivery WHERE delivery.stock <= 4 ORDER BY power(((%(Latitude)s)-delivery.lat), 2) + power((%(Longitude)s-delivery.lng), 2) LIMIT 1;"
    cur.execute(sqlFindRestaurant, {"Latitude": lat, "Longitude": longi})
    delivery_id = (cur.fetchone())[0]  # delivery id

else:
    print("Input Error!")

print(time.time() - start)
```

The last part of the customer's main function was to create a search engine that could be able to fins the stores located near by the customer's original position. In order to do this, we would have first to come up with a special calculation to find the difference between the longitude and latitude of the delivery service and the restaurant. The reason why we use this method is because we must get the closest restaurants located near the desired location previously defined. After finishing with this calculation, we fin the delivery's ID to have a link between the delivery service and our list if restaurants, thing that will help us to accomplish the goal of finding the closest restaurant. The customer's ID here is most that important as we need to find the exact list if restaurants that are near by the solicited customer. However, due to an unrecognizable problem, I could not find the way to finish this tool as the program would keep showing that the selected format for location was not the correct one. Even though the result for this part of the program was not the expected one, I understood all the logic and idea behind the method for looking for the nearest restaurant to our customer's location.

④ Code Results

4.1 Seller

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python seller.py info 2
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321
--------------------------
Name: John
Phone: 01023456789
Email: cdc@hanyang.ac.kr
--------------------------
Running Time: 0.015194416046142578
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python seller.py info 3
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321
--------------------------
Name: 성교식
Phone: 01073033994
Email: ivickb@yale.edu
--------------------------
Running Time: 0.010999917984008789
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python seller.py update 2 phone 01022222223
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321
CHANGE PHONE!!
--------------------------
Name: John
Phone: 01022222223
Email: cdc@hanyang.ac.kr
--------------------------
NEW INFO --> [(2, 'John', '01022222223', 'cdc', 'hanyang.ac.kr', 'hehehe')]
Running Time: 0.014999866485595703
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

4.2 Store

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python store.py info 2
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321

Store #2's Info:
--------------------------------------------------
Store's Name: 고기왕
Store's Address: 제주특별자치도 제주시 일주서로 7329-3
Store's Phone Numbers:
#0 --> 0285434668
Store Seller's ID: 36816
Store's longitude: 33.4934
Store's Latitude: 126.43
Store's Schedule:
```

day	open	closed	holiday
0	1230	0900	False
1	1330	1100	False
2	0300	0000	False
3			True
4	1530	2230	False
5			True
6	2200	2300	False

```
--------------------------------------------------
Running Time: 0.011638402938842773
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python store.py menu 2
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321
Store #2's Menu:
-------------------------------------------
#1. Menu ID: 3, Menu: 밥고로케크림
#2. Menu ID: 4, Menu: 추탕날맥주고추장
#3. Menu ID: 5, Menu: 라바슈잡탕밥홍차
#4. Menu ID: 131211, Menu: pasta
-------------------------------------------
Running Time: 1.7099330425262451
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python store.py add_menu 2 "Pizza"
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321
Store #2's Menu:
-------------------------------------------
#1. Menu ID: 3, Menu: 밥고로케크림
#2. Menu ID: 4, Menu: 추탕날맥주고추장
#3. Menu ID: 5, Menu: 라바슈잡탕밥홍차
#4. Menu ID: 131211, Menu: pasta
#5. Menu ID: 131212, Menu: Pizza
-------------------------------------------
Running Time: 0.029007673263549805
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

4.3 Customer

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python customer.py info 2
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321

Customer #2's Information:
-------------------------------------
Name: 허순이
Phone: 01022215104
Email: nolivdh@cpanel.net
Password: 353dy8
Payments:
╒═══════════════════════════════╤═══════════╕
│ data                          │ type      │
╞═══════════════════════════════╪═══════════╡
│ {'card_num': 709697237096972} │ card      │
├───────────────────────────────┼───────────┤
│ {'card_num': 685540516855405} │ card      │
├───────────────────────────────┼───────────┤
│ {'card_num': 2503984725039847} │ card      │
├───────────────────────────────┼───────────┤
│ {'card_num': 847530338475303} │ card      │
├───────────────────────────────┼───────────┤
│ {'card_num': 1234123456700000} │ card      │
├───────────────────────────────┼───────────┤
│ {'card_num': 1234123456700000} │ card      │
├───────────────────────────────┼───────────┤
│ {'acc_num': ['3', '123456789']} │ account   │
╘═══════════════════════════════╧═══════════╛
Latitude & Longitude: 37.0419/127.93377
-------------------------------------
0.011031150817871094
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python customer.py address 2
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321

Customer #2's Address Information:
-----------------------------------------------------
2. 177B hahahaha
3. 177B Santiago
4. 177B Arica
5. 177B Punta Arenas
-----------------------------------------------------
0.010147809982299805
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python customer.py address 2 -c "177A Bleecker Street"
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321

Customer #2's Address Information:
------------------------------------------------------
2. 177B hahahaha
3. 177B Santiago
4. 177B Arica
5. 177B Punta Arenas
6. 177A Bleecker Street
------------------------------------------------------
0.016338348388671875
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python customer.py address 2 -n 2
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321

Customer #2's Address Information:
------------------------------------------------------
3. 177B Santiago
4. 177B Arica
5. 177B Punta Arenas
6. 177A Bleecker Street
------------------------------------------------------
0.014022350311279297
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python customer.py address 2 -n 3
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321

Customer #2's Address Information:
------------------------------------------------------
4. 177B Arica
5. 177B Punta Arenas
6. 177A Bleecker Street
------------------------------------------------------
0.01393270492553711
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python customer.py address 2 -o 6 "221B Baker Street"
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321

Customer #2's Address Information:
------------------------------------------------------
4. 177B Arica
5. 177B Punta Arenas
6. 221B Baker Street
------------------------------------------------------
0.016325712203979492
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python customer.py pay 2
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321

Customer #2's Payment methods:

╒═══╤═══════════════════════════════════╤═══════════╕
│   │ data                              │ type      │
╞═══╪═══════════════════════════════════╪═══════════╡
│ 0 │ {'card_num': 7096972370969723}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 1 │ {'card_num': 6855405168554051}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 2 │ {'card_num': 2503984725039847}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 3 │ {'card_num': 8475303384753033}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 4 │ {'card_num': 1234123456700000}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 5 │ {'card_num': 1234123456700000}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 6 │ {'acc_num': ['3', '123456789']}   │ account   │
╘═══╧═══════════════════════════════════╧═══════════╛
0.011954307556152344
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python customer.py pay 2 --add-card 1234123456785688
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321

Customer #2's Payment methods:

╒═══╤═══════════════════════════════════╤═══════════╕
│   │ data                              │ type      │
╞═══╪═══════════════════════════════════╪═══════════╡
│ 0 │ {'card_num': 7096972370969723}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 1 │ {'card_num': 6855405168554051}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 2 │ {'card_num': 2503984725039847}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 3 │ {'card_num': 8475303384753033}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 4 │ {'card_num': 1234123456700000}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 5 │ {'card_num': 1234123456700000}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 6 │ {'acc_num': ['3', '123456789']}   │ account   │
├───┼───────────────────────────────────┼───────────┤
│ 7 │ {'card_num': 1234123456785688}    │ card      │
╘═══╧═══════════════════════════════════╧═══════════╛
0.015015602111816406
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python customer.py pay 2 --add-account 3 123456777
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321

Customer #2's Payment methods:

╒═══╤═══════════════════════════════════╤═══════════╕
│   │ data                              │ type      │
╞═══╪═══════════════════════════════════╪═══════════╡
│ 0 │ {'card_num': 7096972370969723}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 1 │ {'card_num': 6855405168554051}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 2 │ {'card_num': 2503984725039847}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 3 │ {'card_num': 8475303384753033}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 4 │ {'card_num': 1234123456700000}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 5 │ {'card_num': 1234123456700000}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 6 │ {'acc_num': ['3', '123456789']}   │ account   │
├───┼───────────────────────────────────┼───────────┤
│ 7 │ {'card_num': 1234123456785688}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 8 │ {'acc_num': ['3', '123456777']}   │ account   │
╘═══╧═══════════════════════════════════╧═══════════╛
0.015709400177001953
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

```
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project> python customer.py pay 2 -r 1
Connecting To:  host=localhost user=postgres dbname=baedal password=1234 port=54321

Customer #2's Payment methods:

╒═══╤═══════════════════════════════════╤═══════════╕
│   │ data                              │ type      │
╞═══╪═══════════════════════════════════╪═══════════╡
│ 0 │ {'card_num': 7096972370969723}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 1 │ {'card_num': 8475303384753033}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 2 │ {'card_num': 1234123456700000}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 3 │ {'card_num': 1234123456700000}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 4 │ {'acc_num': ['3', '123456789']}   │ account   │
├───┼───────────────────────────────────┼───────────┤
│ 5 │ {'card_num': 1234123456785688}    │ card      │
├───┼───────────────────────────────────┼───────────┤
│ 6 │ {'acc_num': ['3', '123456777']}   │ account   │
╘═══╧═══════════════════════════════════╧═══════════╛
0.015998125076293945
PS C:\Users\Bruno\Desktop\Last Semester\DataBase\Assignments\3\21-fall-dbs\homework3\postgres-starter\ApplicationDevelopment\project>
```

⑤ Conclusion and Final Thoughts

After finishing the project, it was possible to understand the main contents related to database and how to develop a simple application with all the necessary functionalities that we would use in a real situation of delivery services in our daily life. One of the most satisfactory parts of this assignment was the fact that all the written code can show a real change in our database or even delete data that we do not want to use anymore. Even though I could not accomplish the goal in its totality, I deeply understood the content proposed for this assignment as well as make a work around of the main functionalities that were introduced in the instructions ppt. My understanding of Database got a great improvement as well as more tools for future development opportunities in this area. As a closing final thought, I enjoyed doing this project as I could see real results when working with the code and the database itself.