University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: ___4___

Date: ___2024/08/05___

Group Number: ____33____

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Kevin Chu (Chao-Wu Chu)	85406312	b0u5x	kevinchu6601@gmail.com
Kaicheng Lu	14723720	o1u1i	kaichenglu2020@gmail.com
Jason Liang	75499566	w3s8d	cajasonliang@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

GitHub Repository Link

https://github.students.cs.ubc.ca/CPSC304-2024S-T2/project_b0u5x_o1u1i_w3s8d

Project Summary

Our project is a food delivering service that manages the process of a customer ordering food from a restaurant and having the food delivered to them. Customers are able to write reviews for both the restaurant and the driver responsible for delivering the food. In addition, customers are able to register, and view their history of orders and payments.

Project Description

Our project is a food delivery service. Users are able to register a new user. There, they enter information about themselves such as their name, address, phone, email, customer account username, credit card, the discount associated with their account, if they have a membership, and their points. During the registration process, users are able to cancel. For the registration to be complete, their phone number, address, name, amount of points, credit card, and username must be filled in. Customers are able to delete a registered customer. They do this by giving the associated phone number of the customer that they want to delete. In addition, customers are able to update any information of a specific registered customer if they know the phone number of the customer. Users are also able to view all the customers that are currently in the database and their data.

The user is able to add drivers to the database. To register a driver, the user must give the name, phone number, driver account username, and the vehicle they drive. In addition, users are also able to delete a driver in the database. This process will require the phone number of the driver they want to delete. In addition, users are able to update any information of a particular driver. They are also able to search for any driver based on specific information. They are able to input any attribute, which will allow them to search for all the drivers that meet their search criteria. Users are also able to view all the drivers that are in the database and their data.

Users are also able to add food items to menus. To add a food item, the user must give the name, price and type of food it is and whether the food item is vegan. Users are also able to delete any food item on the menu. This is done by the user specifying the name of the food item that they want to delete. Users are also able to update the information about any food item existing in the database. Users are also able to view all the food items that are listed in the database.

In addition, users are able to add new food orders to the database. To input an order into the database, the user gives the total price of the order, and the estimated time of the arrival. The user is also able to delete any existing order within the database. To do this, the user will need to give the order number of the order that they want to delete. Users are also able to view the history of their orders. They are able to see the ID of the order, the customer phone number, the restaurant address, the driver phone number, the total price, and the time of arrival associated with each order. as well as their payment history. In addition, the user is able to find the driver and their information of the one delivering their order.

Users are also able to make new payments. To add a payment, the user needs to give the total price of the payment they want to make, and the status of their payment to indicate whether the payment was successful or not. In addition, users are able to view their history of payments. They will be able to see the ID, status, price, restaurant address, and the driver phone associated with each of their payments. In addition, users are able to find the number of successful payments.

Users are also able to add menus into the database. To add a menu, the user will need to give the name and cuisine of their menu. In addition, they are able to delete an existing menu in the database. To do this, they are required to give the menu number of the menu they want to delete. In addition, users are able to view the list of menus in the database as well as the list of food items.

Users are also able to make reviews for a driver or a restaurant. Users are able to see a list of all the reviews for drivers in the database or a list of all the reviews for restaurants in the database. When viewing the list of restaurant reviews, users are able to see the phone number of the customer, time stamp, title, image, comment, rating type, restaurant address, food quality rating and portion size rating associated with their restaurant review. Users are able to make new restaurant or driver reviews. To make a new restaurant review, users are prompted to give the title of their review, timestamp of when the review was made, image, comment, food quality rating, and a portion size rating of their restaurant review. Users are able to delete restaurant reviews in the database. To do this, they must give the timestamp of when the review was made. Users are also able to make a driver review. To do this, they need to give the title of the review, the timestamp of their review, image, comment, a package handling rating, and the delivery time rating of their driver review. In addition, users are able to view only driver reviews that have a rating greater than 3. Users are also able to find reviews for a specific driver by giving the driver number that they want to see reviews for. Users can also find reviews to find reviews for a specific restaurant by giving the restaurant number of the restaurant they want to see reviews for. Users are also able to choose to view only restaurant reviews with a rating greater than 3.

The program also enables users to choose to do a projection on different categories in the database including customers, restaurants, menu advanced info, menu basic info, food advanced info, food basic info, orders, drivers, review advanced info, review basic info, food reviews, driver reviews, payment advanced info, and payment basic info. The user can choose a category and a combination of attributes that they want to filter to view a table for. For customers, the attributes they can choose to view are name, address, phone, email, customer account, credit card, membership, points, and discount. For restaurants, users can choose to view the attributes name, address, open hours, and restaurant account. For menu advanced info, the user can choose to view the attributes name and cuisine. For menu basic info, the user can choose to view the attributes menu id and name. For food advanced info, users can choose to view the attributes, food type and is it vegan. For food basic info, users can choose to view the attributes, food name, food price, and food type. For orders, users can choose to view the attributes order id, total price, time arrival, customer phone, restaurant address, and driver phone. For drivers, users can choose to view the attributes phone, name, vehicle, and driver account. For review basic info, users can choose to view the attributes, customer phone, time stamp, title, image, comment, food quality rating and portion size rating. For review advanced info, users can choose to view the attributes, comment and rating type. For food reviews, users can view the attributes, customer phone, timestamp, restaurant address, food quality rating, and portion size rating. For driver reviews, the user can choose to view the attributes, customer phone, timestamp, driver phone, package handling rating, and delivery time rating. For payment advanced info, the user can view the attributes, order ID, price, restaurant address, and driver phone. For payment basic info, users can see the attributes payment ID, status, and order ID.

Schema Changes

We changed the domain for the attribute timestamp for the entity DriverReview from datetime to timestamp. We changed the domain for the attribute timestamp for the entity FoodReivew from date to timestamp. We changed the domain for the attribute time arrival for the entity Orders from date to timestamp. We changed the domain for the attribute timestamp for the Review from date to timestamp.

We changed the domain for these attributes because we thought that in the context of those entities to be better suited with the domain timestamp. This is because the domain timestamp is more detailed compared to datetime, and it makes more sense in all of those entities. This can make it provide more accurate data for users. This is because precision on the time when the order can be vital for delivery. In addition, for Reviews, it helps in identifying each review because it is more precise, and thus significantly less likely for two different reviews to have the same timestamp. For orders, we thought users might also find it useful to know the precise time that their delivery will arrive. At the same time, it can help users differentiate between orders.

We renamed the attribute MID in the Own relationship and in the Contain relationship to menuID. This is because the attribute name MID is unclear, and it is difficult to determine that it refers to menuID. We renamed the fName attribute in the Contain and the Ordered relationship to foodName. Similarly, it is because the attribute name is unclear and it can be difficult to know that it means foodName. Thus, we decided to rename it to foodName so the attribute name is clear and easy to determine what it means.
We renamed the entities food1 and food2 to foodBasicInfo and foodAdvancedInfo, menu1 and menu2 to menuBasicInfo and menuAdvancedInfo, payment1 and payment2 to paymentBasicInfo and paymentAdvancedInfo, and review1 and review2 to reviewBasicInfo and reviewAdvancedInfo. This is because the names of the entities did not give any information on what it represented. The names before were more ambiguous, but with the current names, it makes it more clear the type of attributes that the entities would contain.

We also changed the name of all the attributes to have the entity in its name. This is because a lot of our attributes had the same name in our program such as timestamp that was in driverReviews and foodReview. By changing the name of the attributes to include the entity where it is from, it makes it clear where it is from and makes the names of each attribute unique. This helps when we use foreign keys in our project in making it clear which attribute from which entity we are using.

We changed the name of attributes that had abbreviations to have the full name like customerAcc to be customer_account. This is because having the abbreviations made the name less representative of what the attribute was. By having the full name rather than an abbreviation, it makes it immediately clear what the attribute refers to and eliminates any ambiguity that was there from having an abbreviation.

Schema

Customers (customer_phone: varchar[15], customer_name: varchar[255], customer_address: varchar[255], customer_email:varchar[255], customer_account: varchar[20], customer_paymentInfo: varchar[255], customer_membership: varchar[3], customer_points: int, customer_discount: varchar[4]),
customer_email, customer_account UNIQUE; customer_name, customer_address, customer_email, customer_account, customer_paymentInfo, customer_points NOT NULL; customer_membership DEFAULT ‘No’
CK: customer_phone, customer_email, customer_account

Restaurants (restaurant_address: varchar[255], restaurant_name: varchar[255], restaurant_openHours: varchar[255], restaurant_account: varchar[20]),
restaurant_account UNIQUE; restaurant_name, restaurant_openHours, restaurant_account NOT NULL
CK: restaurant_address, restaurant_account

Own (**restaurant_address**: varchar[255], **menu_id**: int)
CK: {restaurant_address, menu_id}

MenuBasicInfo (menu_id: int, **menu_name**: varchar[255]),
menu_name NOT NULL
CK: menu_id, menu_name

MenuAdvancedInfo (menu_name: varchar[255], menu_cuisine: varchar[255])
CK: {menu_name}

Contain (**menu_id**: int, **food_name**: varchar[255])
CK: {menu_id, food_name}

FoodBasicInfo (food_name: varchar[255], food_price: decimal[5, 2], **food_type**: varchar[255]),
food_price, food_typeNOT NULL
CK: {food_name, food_type}

FoodAdvancedInfo (food_hasVeganDiet: varchar[3], food_type: varchar[255]),
food_type, food_hasVeganDiet NOT NULL
CK: {food_type}

Ordered (**food_name**: varchar[255], **order_id**: int, orderedQuantity: int),
orderedQuantity NOT NULL
CK: {food_name, order_id}

Orders (order_id: int, order_totalPrice: decimal[5, 2], order_timeArrival: timestamp, **customer_phone**: varchar[15], **restaurant_address**: varchar[255], **driver_phone**: varchar[15]),
order_totalPrice, order_timeArrival, customer_phone, restaurant_address, driver_phone NOT NULL
CK: {order_id}

Drivers (driver_phone: varchar[15], driver_name: varchar[255], driver_account: varchar[20], driver_vehicle: varchar[255]),
driver_account UNIQUE; driver_name, driver_account, driver_vehicle NOT NULL
CK: driver_phone, driver_account

ReviewBasicInfo (**customer_phone**: varchar[15], **review_timeStamp**: timestamp, review_title: varchar[50], review_comment: varchar[255], review_image: varchar[255]),
review_title, **review_comment** NOT NULL
CK: {customer_phone, review_timeStamp}

ReviewAdvancedInfo (review_comment: varchar[255], review_ratingType: varchar[8]),
review_ratingType NOT NULL
CK: {review_comment}

FoodReviews (**customer_phone**: varchar[15], **review_timeStamp**: timestamp, **restaurant_address**: varchar[255], foodReviews_foodQualityRating: int, foodReviews_portionSizeRating: int), restaurant_address NOT NULL
CK: {customer_phone, review_timeStamp}

DriverReviews (**customer_phone**: varchar[15], **review_timeStamp**: timestamp, **driver_phone**: varchar[15], driverReviews_packageHandlingRating: int, driverReviews_deliveryTimeRating: int), driver_phone NOT NULL
CK: {customer_phone, review_timeStamp}

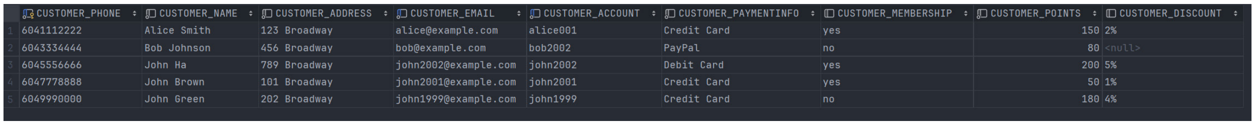
PaymentBasicInfo (payment_id: int, payment_status: varchar[7], **order_id**: int), status, order_id NOT NULL
CK: {payment_id}

PaymentAdvancedInfo (payment_price: decimal[5, 2], **order_id**: int, **restaurant_address**: varchar[255], **driver_phone**: varchar[15]), price, restaurant_address, driver_phone NOT NULL
CK: {order_id}

SQL Initialization

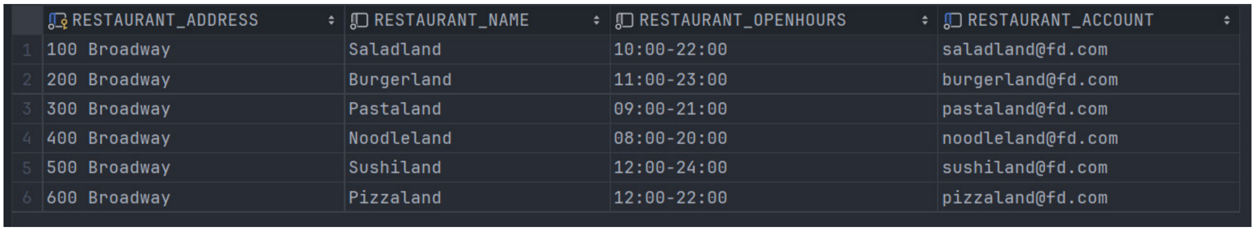
All SQL statements for initialization is included in (src/sql/scripts/DatabaseScripts.sql), the screenshots below are the tables after initialization on SQL*PLUS.

Customers:



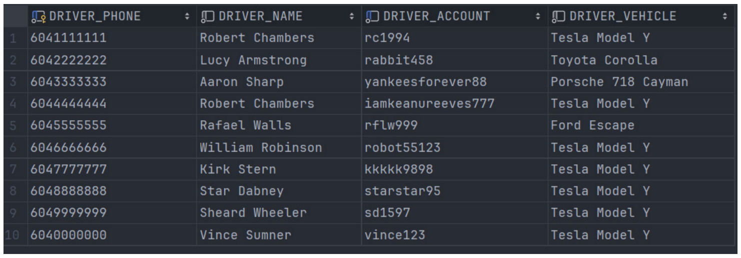
CUSTOMER_PHONE	CUSTOMER_NAME	CUSTOMER_ADDRESS	CUSTOMER_EMAIL	CUSTOMER_ACCOUNT	CUSTOMER_PAYMENTINFO	CUSTOMER_MEMBERSHIP	CUSTOMER_POINTS	CUSTOMER_DISCOUNT
6041112222	Alice Smith	123 Broadway	alice@example.com	alice001	Credit Card	yes	150	2%
6043334444	Bob Johnson	456 Broadway	bob@example.com	bob2002	PayPal	no	80	<null>
6045556666	John Ha	789 Broadway	john2002@example.com	john2002	Debit Card	yes	200	5%
6047778888	John Brown	101 Broadway	john2001@example.com	john2001	Credit Card	yes	50	1%
6049990000	John Green	202 Broadway	john1999@example.com	john1999	Credit Card	no	180	4%

Restaurants



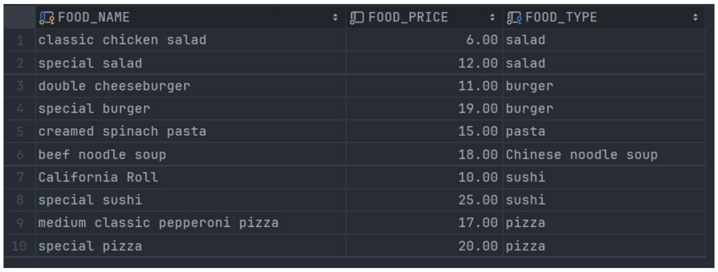
RESTAURANT_ADDRESS	RESTAURANT_NAME	RESTAURANT_OPENHOURS	RESTAURANT_ACCOUNT
100 Broadway	Saladland	10:00-22:00	saladland@fd.com
200 Broadway	Burgerland	11:00-23:00	burgerland@fd.com
300 Broadway	Pastaland	09:00-21:00	pastaland@fd.com
400 Broadway	Noodleland	08:00-20:00	noodleland@fd.com
500 Broadway	Sushiland	12:00-24:00	sushiland@fd.com
600 Broadway	Pizzaland	12:00-22:00	pizzaland@fd.com

Drivers:



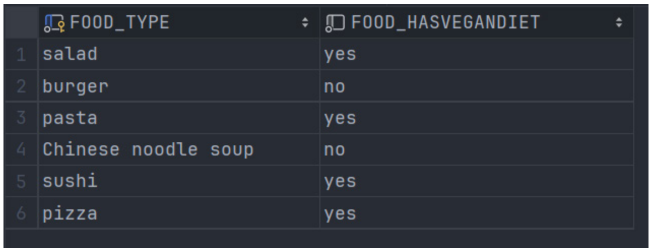
DRIVER_PHONE	DRIVER_NAME	DRIVER_ACCOUNT	DRIVER_VEHICLE
6041111111	Robert Chambers	rc1994	Tesla Model Y
6042222222	Lucy Armstrong	rabbit458	Toyota Corolla
6043333333	Aaron Sharp	yankeesforever88	Porsche 718 Cayman
6044444444	Robert Chambers	iamkeanureeves777	Tesla Model Y
6045555555	Rafael Walls	rflw999	Ford Escape
6046666666	William Robinson	robot55123	Tesla Model Y
6047777777	Kirk Stern	kkkkk9898	Tesla Model Y
6048888888	Star Dabney	starstar95	Tesla Model Y
6049999999	Sheard Wheeler	sd1597	Tesla Model Y
6040000000	Vince Sumner	vince123	Tesla Model Y

FoodBasicInfo:



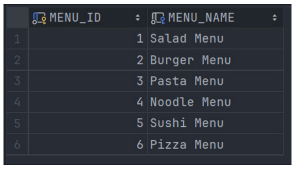
FOOD_NAME	FOOD_PRICE	FOOD_TYPE
classic chicken salad	6.00	salad
special salad	12.00	salad
double cheeseburger	11.00	burger
special burger	19.00	burger
creamed spinach pasta	15.00	pasta
beef noodle soup	18.00	Chinese noodle soup
California Roll	10.00	sushi
special sushi	25.00	sushi
medium classic pepperoni pizza	17.00	pizza
special pizza	20.00	pizza

FoodAdvancedInfo:



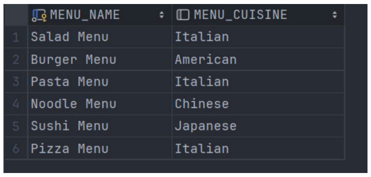
FOOD_TYPE	FOOD_HASVEGANDIET
salad	yes
burger	no
pasta	yes
Chinese noodle soup	no
sushi	yes
pizza	yes

MenuBasicInfo:



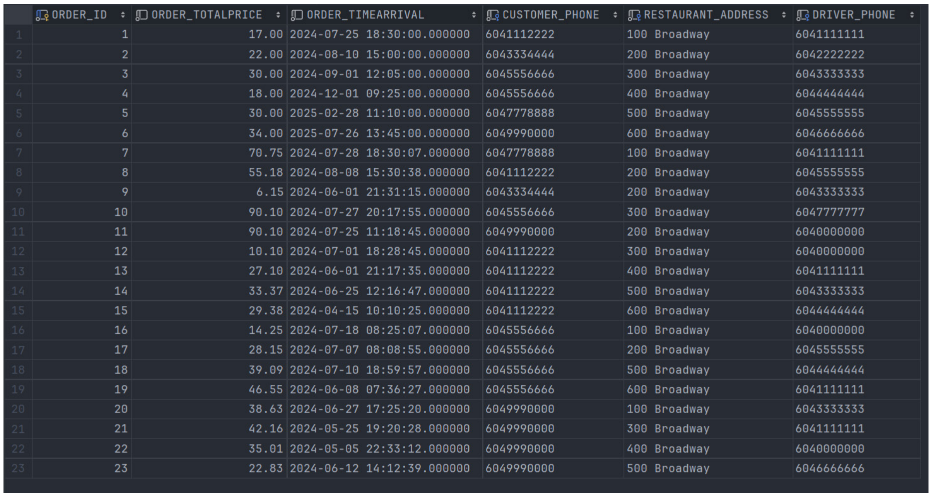
MENU_ID	MENU_NAME
1	Salad Menu
2	Burger Menu
3	Pasta Menu
4	Noodle Menu
5	Sushi Menu
6	Pizza Menu

MenuAdvancedInfo:



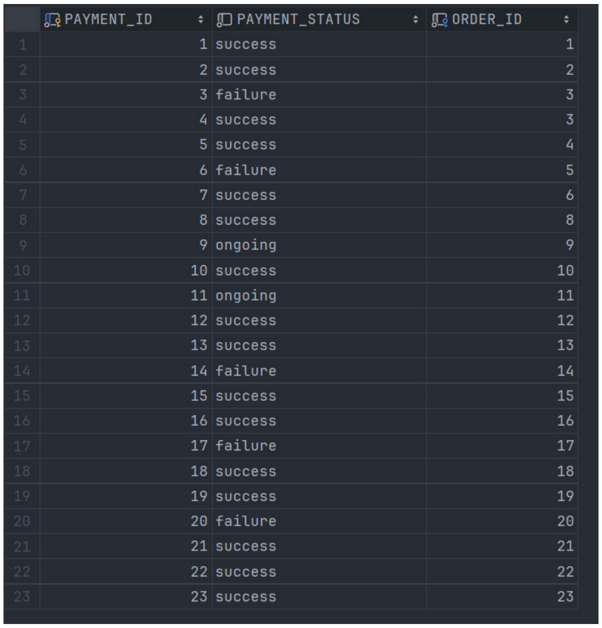
MENU_NAME	MENU_CUISINE
Salad Menu	Italian
Burger Menu	American
Pasta Menu	Italian
Noodle Menu	Chinese
Sushi Menu	Japanese
Pizza Menu	Italian

Orders:



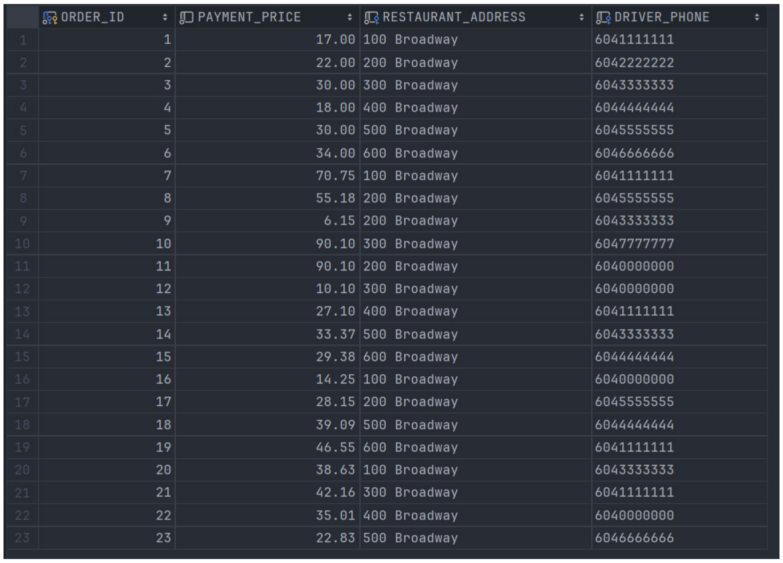
ORDER_ID	ORDER_TOTALPRICE	ORDER_TIMEARRIVAL	CUSTOMER_PHONE	RESTAURANT_ADDRESS	DRIVER_PHONE
1	17.00	2024-07-25 18:30:00.000000	6041112222	100 Broadway	6041111111
2	22.00	2024-08-10 15:00:00.000000	6043334444	200 Broadway	6042222222
3	30.00	2024-09-01 12:05:00.000000	6045556666	300 Broadway	6043333333
4	18.00	2024-12-01 09:25:00.000000	6045556666	400 Broadway	6044444444
5	30.00	2025-02-28 11:10:00.000000	6047778888	500 Broadway	6045555555
6	34.00	2025-07-26 13:45:00.000000	6049990000	600 Broadway	6046666666
7	70.75	2024-07-28 18:30:07.000000	6047778888	100 Broadway	6041111111
8	55.18	2024-08-08 15:30:38.000000	6041112222	200 Broadway	6045555555
9	6.15	2024-06-01 21:31:15.000000	6043334444	200 Broadway	6043333333
10	90.10	2024-07-27 20:17:55.000000	6045556666	300 Broadway	6047777777
11	90.10	2024-07-25 11:18:45.000000	6049990000	200 Broadway	6040000000
12	10.10	2024-07-01 18:28:45.000000	6041112222	300 Broadway	6040000000
13	27.10	2024-06-01 21:17:35.000000	6041112222	400 Broadway	6041111111
14	33.37	2024-06-25 12:16:47.000000	6041112222	500 Broadway	6043333333
15	29.38	2024-04-15 10:10:25.000000	6041112222	600 Broadway	6044444444
16	14.25	2024-07-18 08:25:07.000000	6045556666	100 Broadway	6040000000
17	28.15	2024-07-07 08:08:55.000000	6045556666	200 Broadway	6045555555
18	39.09	2024-07-10 18:59:57.000000	6045556666	500 Broadway	6044444444
19	46.55	2024-06-08 07:36:27.000000	6045556666	600 Broadway	6041111111
20	38.63	2024-06-27 17:25:20.000000	6049990000	100 Broadway	6043333333
21	42.16	2024-05-25 19:20:28.000000	6049990000	300 Broadway	6041111111
22	35.01	2024-05-05 22:33:12.000000	6049990000	400 Broadway	6040000000
23	22.83	2024-06-12 14:12:39.000000	6049990000	500 Broadway	6046666666

PaymentBasicInfo



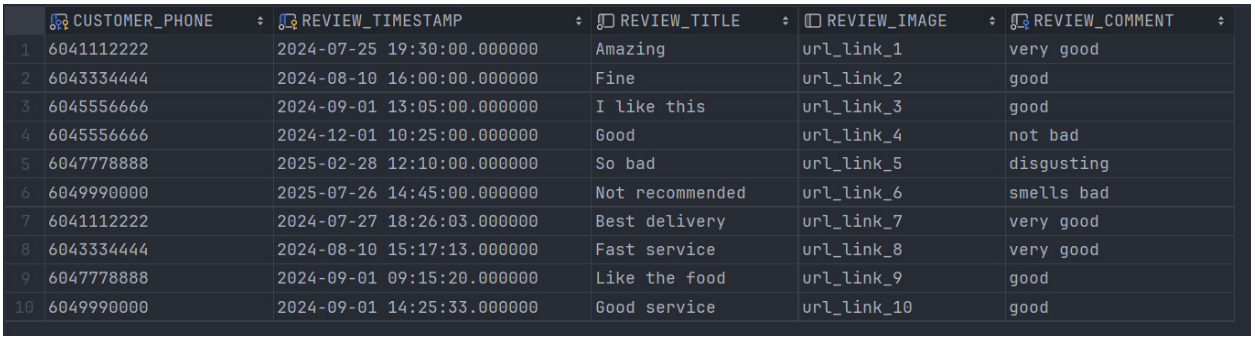
PAYMENT_ID	PAYMENT_STATUS	ORDER_ID
1	success	1
2	success	2
3	failure	3
4	success	3
5	success	4
6	failure	5
7	success	6
8	success	8
9	ongoing	9
10	success	10
11	ongoing	11
12	success	12
13	success	13
14	failure	14
15	success	15
16	success	16
17	failure	17
18	success	18
19	success	19
20	failure	20
21	success	21
22	success	22
23	success	23

PaymentAdvancedInfo:



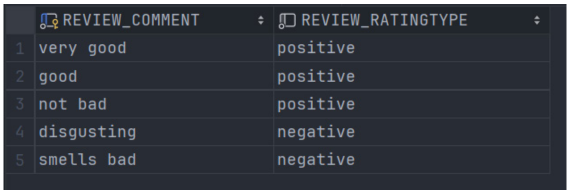
ORDER_ID	PAYMENT_PRICE	RESTAURANT_ADDRESS	DRIVER_PHONE
1	17.00	100 Broadway	6041111111
2	22.00	200 Broadway	6042222222
3	30.00	300 Broadway	6043333333
4	18.00	400 Broadway	6044444444
5	30.00	500 Broadway	6045555555
6	34.00	600 Broadway	6046666666
7	70.75	100 Broadway	6041111111
8	55.18	200 Broadway	6045555555
9	6.15	200 Broadway	6043333333
10	90.10	300 Broadway	6047777777
11	90.10	200 Broadway	6040000000
12	10.10	300 Broadway	6040000000
13	27.10	400 Broadway	6041111111
14	33.37	500 Broadway	6043333333
15	29.38	600 Broadway	6044444444
16	14.25	100 Broadway	6040000000
17	28.15	200 Broadway	6045555555
18	39.09	500 Broadway	6044444444
19	46.55	600 Broadway	6041111111
20	38.63	100 Broadway	6043333333
21	42.16	300 Broadway	6041111111
22	35.01	400 Broadway	6040000000
23	22.83	500 Broadway	6046666666

ReviewBasicInfo:



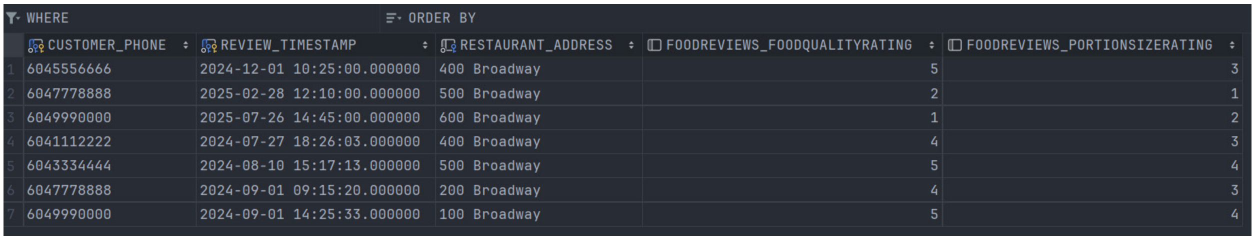
CUSTOMER_PHONE	REVIEW_TIMESTAMP	REVIEW_TITLE	REVIEW_IMAGE	REVIEW_COMMENT
6041112222	2024-07-25 19:30:00.000000	Amazing	url_link_1	very good
6043334444	2024-08-10 16:00:00.000000	Fine	url_link_2	good
6045556666	2024-09-01 13:05:00.000000	I like this	url_link_3	good
6045556666	2024-12-01 10:25:00.000000	Good	url_link_4	not bad
6047778888	2025-02-28 12:10:00.000000	So bad	url_link_5	disgusting
6049990000	2025-07-26 14:45:00.000000	Not recommended	url_link_6	smells bad
6041112222	2024-07-27 18:26:03.000000	Best delivery	url_link_7	very good
6043334444	2024-08-10 15:17:13.000000	Fast service	url_link_8	very good
6047778888	2024-09-01 09:15:20.000000	Like the food	url_link_9	good
6049990000	2024-09-01 14:25:33.000000	Good service	url_link_10	good

ReviewAdvancedInfo:



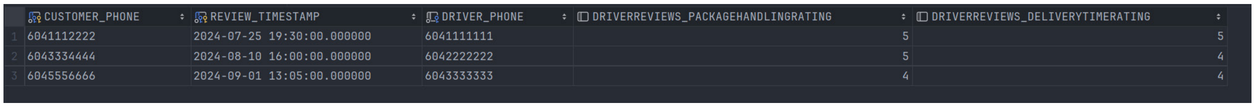
REVIEW_COMMENT	REVIEW_RATINGTYPE
very good	positive
good	positive
not bad	positive
disgusting	negative
smells bad	negative

FoodReviews:



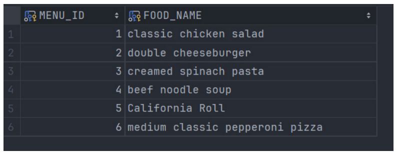
CUSTOMER_PHONE	REVIEW_TIMESTAMP	RESTAURANT_ADDRESS	FOODREVIEWS_FOODQUALITYRATING	FOODREVIEWS_PORTIONSIZERATING
6045556666	2024-12-01 10:25:00.000000	400 Broadway	5	3
6047778888	2025-02-28 12:10:00.000000	500 Broadway	2	1
6049990000	2025-07-26 14:45:00.000000	600 Broadway	1	2
6041112222	2024-07-27 18:26:03.000000	400 Broadway	4	3
6043334444	2024-08-10 15:17:13.000000	500 Broadway	5	4
6047778888	2024-09-01 09:15:20.000000	200 Broadway	4	3
6049990000	2024-09-01 14:25:33.000000	100 Broadway	5	4

DriverReviews:



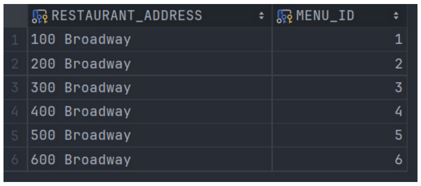
CUSTOMER_PHONE	REVIEW_TIMESTAMP	DRIVER_PHONE	DRIVERREVIEWS_PACKAGEHANDLINGRATING	DRIVERREVIEWS_DELIVERYTIMERATING
6041112222	2024-07-25 19:30:00.000000	6041111111	5	5
6043334444	2024-08-10 16:00:00.000000	6042222222	5	4
6045556666	2024-09-01 13:05:00.000000	6043333333	4	4

Contain:



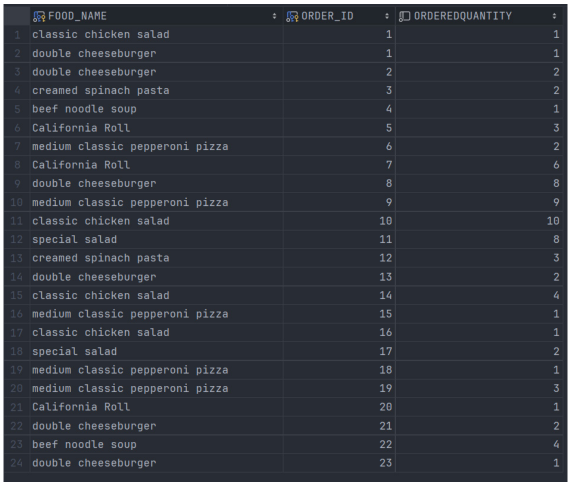
MENU_ID	FOOD_NAME
1	classic chicken salad
2	double cheeseburger
3	creamed spinach pasta
4	beef noodle soup
5	California Roll
6	medium classic pepperoni pizza

Own:



RESTAURANT_ADDRESS	MENU_ID
100 Broadway	1
200 Broadway	2
300 Broadway	3
400 Broadway	4
500 Broadway	5
600 Broadway	6

Ordered:



FOOD_NAME	ORDER_ID	ORDEREDQUANTITY
classic chicken salad	1	1
double cheeseburger	1	1
double cheeseburger	2	2
creamed spinach pasta	3	2
beef noodle soup	4	1
California Roll	5	3
medium classic pepperoni pizza	6	2
California Roll	7	6
double cheeseburger	8	8
medium classic pepperoni pizza	9	9
classic chicken salad	10	10
special salad	11	8
creamed spinach pasta	12	3
double cheeseburger	13	2
classic chicken salad	14	4
medium classic pepperoni pizza	15	1
classic chicken salad	16	1
special salad	17	2
medium classic pepperoni pizza	18	1
medium classic pepperoni pizza	19	3
California Roll	20	1
double cheeseburger	21	2
beef noodle soup	22	4
double cheeseburger	23	1

University of British Columbia, Vancouver
Department of Computer Science

SQL Queries

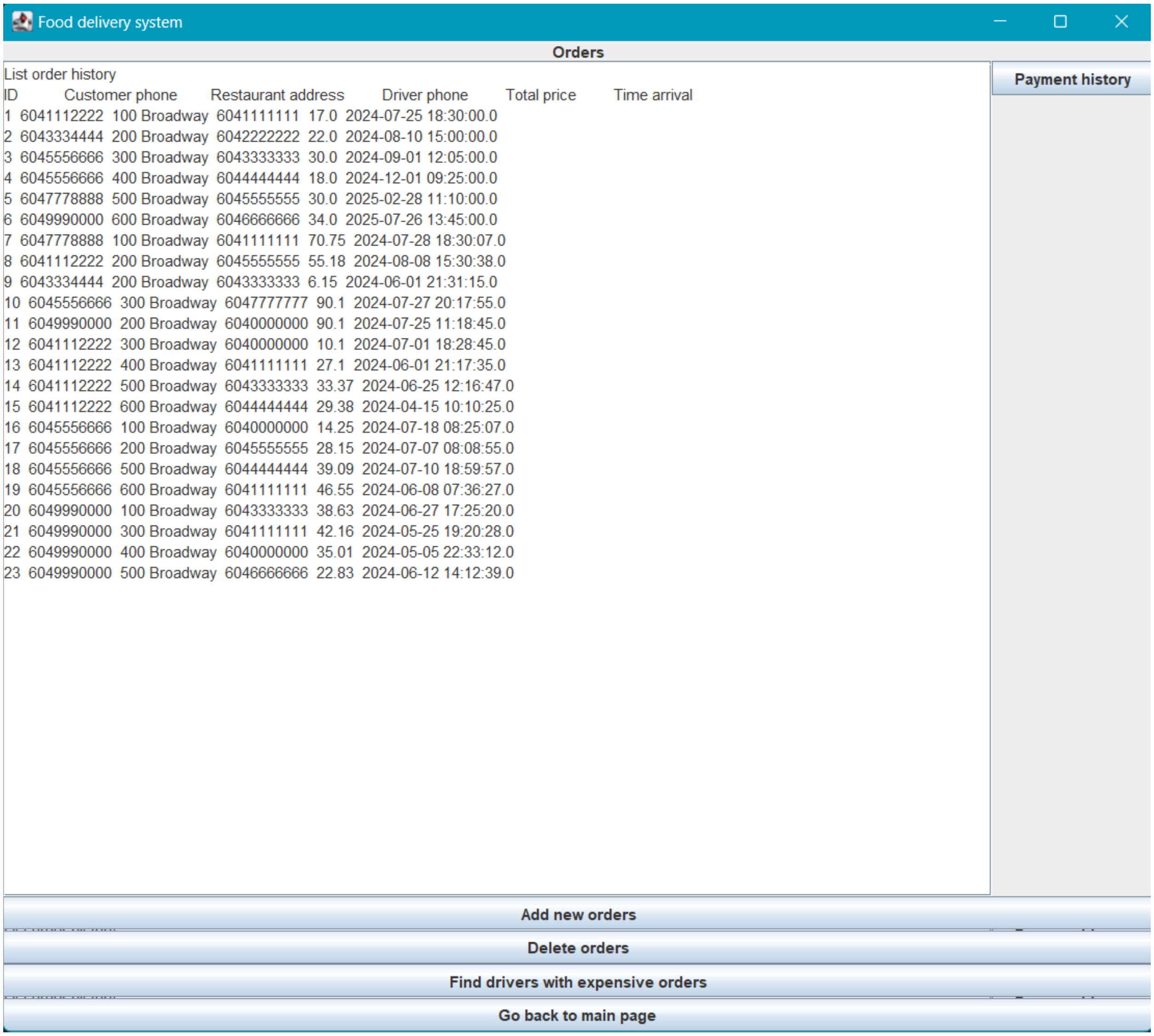
1. Insert Operation

We do the INSERT operation on the Orders Table (with foreign keys that will be related to other tables).

SQL statement:

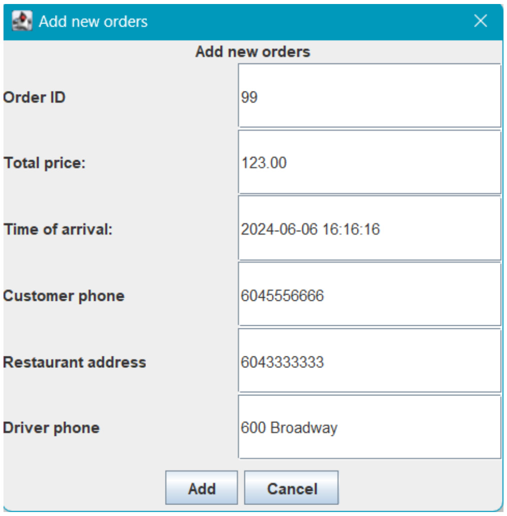
INSERT INTO Orders VALUES (99,123.0,'2024-06-06 16:16:16.0','6045556666','600 Broadway','6043333333');

Step 1: We choose the “Add new orders” button on the GUI.



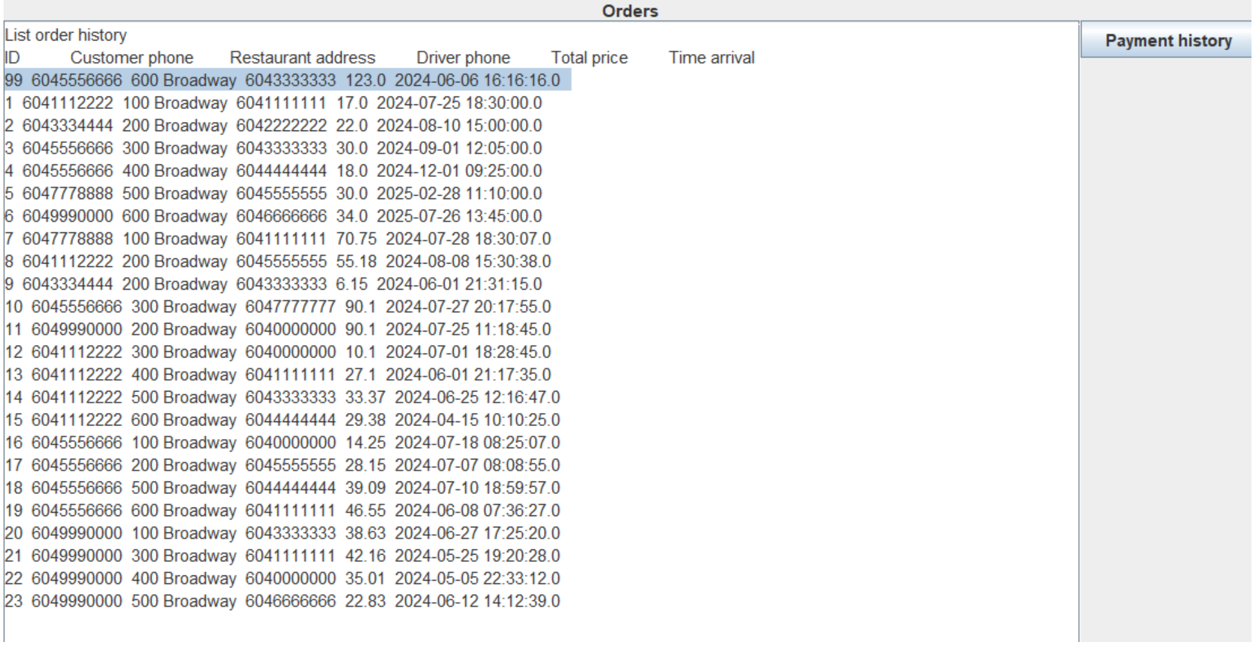
```
List order history
ID        Customer phone     Restaurant address      Driver phone      Total price       Time arrival
1  6041112222  100 Broadway  6041111111  17.0  2024-07-25 18:30:00.0
2  6043334444  200 Broadway  6042222222  22.0  2024-08-10 15:00:00.0
3  6045556666  300 Broadway  6043333333  30.0  2024-09-01 12:05:00.0
4  6045556666  400 Broadway  6044444444  18.0  2024-12-01 09:25:00.0
5  6047778888  500 Broadway  6045555555  30.0  2025-02-28 11:10:00.0
6  6049990000  600 Broadway  6046666666  34.0  2025-07-26 13:45:00.0
7  6047778888  100 Broadway  6041111111  70.75  2024-07-28 18:30:07.0
8  6041112222  200 Broadway  6045555555  55.18  2024-08-08 15:30:38.0
9  6043334444  200 Broadway  6043333333  6.15  2024-06-01 21:31:15.0
10  6045556666  300 Broadway  6047777777  90.1  2024-07-27 20:17:55.0
11  6049990000  200 Broadway  6040000000  90.1  2024-07-25 11:18:45.0
12  6041112222  300 Broadway  6040000000  10.1  2024-07-01 18:28:45.0
13  6041112222  400 Broadway  6041111111  27.1  2024-06-01 21:17:35.0
14  6041112222  500 Broadway  6043333333  33.37  2024-06-25 12:16:47.0
15  6041112222  600 Broadway  6044444444  29.38  2024-04-15 10:10:25.0
16  6045556666  100 Broadway  6040000000  14.25  2024-07-18 08:25:07.0
17  6045556666  200 Broadway  6045555555  28.15  2024-07-07 08:08:55.0
18  6045556666  500 Broadway  6044444444  39.09  2024-07-10 18:59:57.0
19  6045556666  600 Broadway  6041111111  46.55  2024-06-08 07:36:27.0
20  6049990000  100 Broadway  6043333333  38.63  2024-06-27 17:25:20.0
21  6049990000  300 Broadway  6041111111  42.16  2024-05-25 19:20:28.0
22  6049990000  400 Broadway  6040000000  35.01  2024-05-05 22:33:12.0
23  6049990000  500 Broadway  6046666666  22.83  2024-06-12 14:12:39.0
```

Step 2: We enter the information to insert a new tuple (there is a little bug that Restaurant Address and Driver phone should be in opposite text box).



Order ID	99
Total price:	123.00
Time of arrival:	2024-06-06 16:16:16
Customer phone	6045556666
Restaurant address	6043333333
Driver phone	600 Broadway

Step 3: We can see that the new tuple is inserted into the Orders table.



```
List order history
ID       Customer phone      Restaurant address       Driver phone      Total price        Time arrival
99  6045556666  600 Broadway  6043333333  123.0  2024-06-06 16:16:16.0
1  6041112222  100 Broadway  6041111111  17.0  2024-07-25 18:30:00.0
2  6043334444  200 Broadway  6042222222  22.0  2024-08-10 15:00:00.0
3  6045556666  300 Broadway  6043333333  30.0  2024-09-01 12:05:00.0
4  6045556666  400 Broadway  6044444444  18.0  2024-12-01 09:25:00.0
5  6047778888  500 Broadway  6045555555  30.0  2025-02-28 11:10:00.0
6  6049990000  600 Broadway  6046666666  34.0  2025-07-26 13:45:00.0
7  6047778888  100 Broadway  6041111111  70.75  2024-07-28 18:30:07.0
8  6041112222  200 Broadway  6045555555  55.18  2024-08-08 15:30:38.0
9  6043334444  200 Broadway  6043333333  6.15  2024-06-01 21:31:15.0
10  6045556666  300 Broadway  6047777777  90.1  2024-07-27 20:17:55.0
11  6049990000  200 Broadway  6040000000  90.1  2024-07-25 11:18:45.0
12  6041112222  300 Broadway  6040000000  10.1  2024-07-01 18:28:45.0
13  6041112222  400 Broadway  6041111111  27.1  2024-06-01 21:17:35.0
14  6041112222  500 Broadway  6043333333  33.37  2024-06-25 12:16:47.0
15  6041112222  600 Broadway  6044444444  29.38  2024-04-15 10:10:25.0
16  6045556666  100 Broadway  6040000000  14.25  2024-07-18 08:25:07.0
17  6045556666  200 Broadway  6045555555  28.15  2024-07-07 08:08:55.0
18  6045556666  500 Broadway  6044444444  39.09  2024-07-10 18:59:57.0
19  6045556666  600 Broadway  6041111111  46.55  2024-06-08 07:36:27.0
20  6049990000  100 Broadway  6043333333  38.63  2024-06-27 17:25:20.0
21  6049990000  300 Broadway  6041111111  42.16  2024-05-25 19:20:28.0
22  6049990000  400 Broadway  6040000000  35.01  2024-05-05 22:33:12.0
23  6049990000  500 Broadway  6046666666  22.83  2024-06-12 14:12:39.0
```

- **Function Reference**

  1. SQL Query String:

     Where: OrderService.java - line 45

     Method: public String[] insert(OrdersModel model)

  2. Query Executed:

     Where: AppDOA.java - line 1174

     Method: public String[] insertOrder(OrdersModel model, String query)

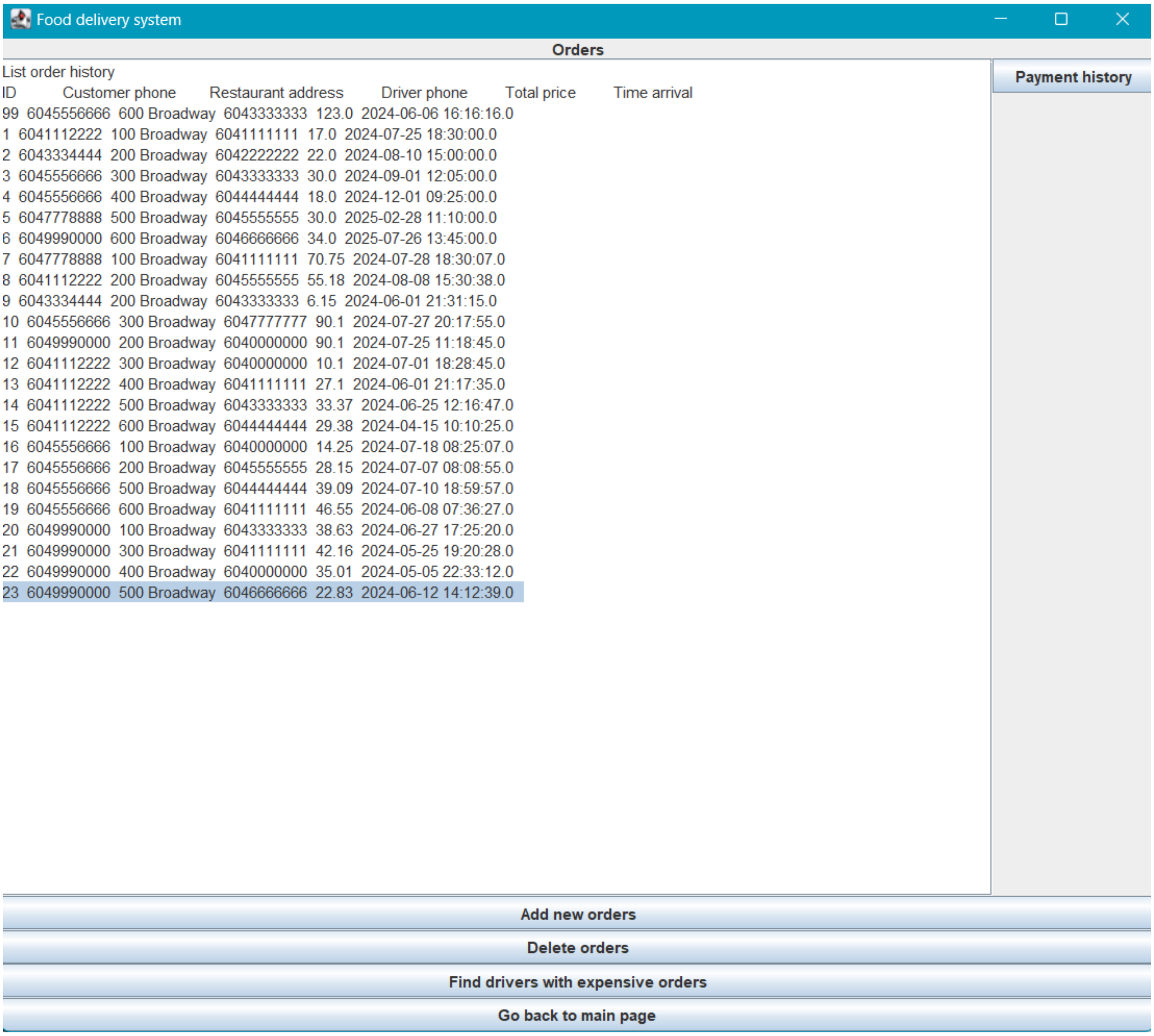
2. Delete Operation

We do the DELETE operation on the Orders Table.

SQL statement:

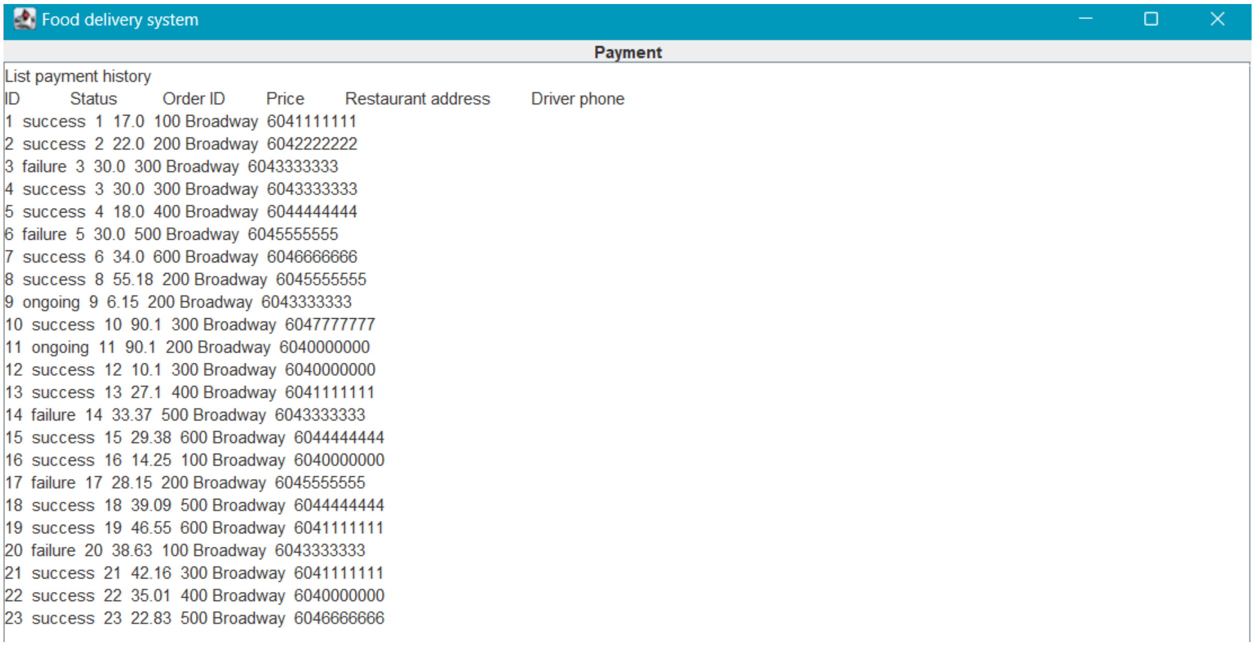
DELETE FROM Orders WHERE order_id = 23;

Step 1: We pick which tuple to delete, here we pick orderID with 23, this will also effect the tuples with a foreign key of 23 in PaymentAdvancedInfo and PaymentBasicInfo tables.



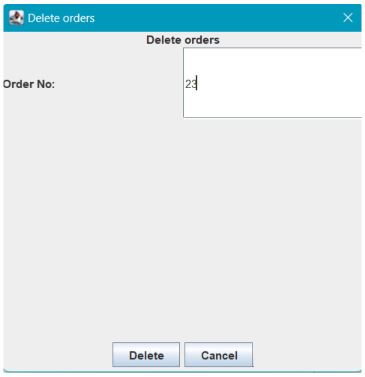
```
List order history
ID       Customer phone     Restaurant address     Driver phone     Total price     Time arrival
99  6045556666  600 Broadway  6043333333  123.0  2024-06-06 16:16:16.0
1  6041112222  100 Broadway  6041111111  17.0  2024-07-25 18:30:00.0
2  6043334444  200 Broadway  6042222222  22.0  2024-08-10 15:00:00.0
3  6045556666  300 Broadway  6043333333  30.0  2024-09-01 12:05:00.0
4  6045556666  400 Broadway  6044444444  18.0  2024-12-01 09:25:00.0
5  6047778888  500 Broadway  6045555555  30.0  2025-02-28 11:10:00.0
6  6049990000  600 Broadway  6046666666  34.0  2025-07-26 13:45:00.0
7  6047778888  100 Broadway  6041111111  70.75  2024-07-28 18:30:07.0
8  6041112222  200 Broadway  6045555555  55.18  2024-08-08 15:30:38.0
9  6043334444  200 Broadway  6043333333  6.15  2024-06-01 21:31:15.0
10  6045556666  300 Broadway  6047777777  90.1  2024-07-27 20:17:55.0
11  6049990000  200 Broadway  6040000000  90.1  2024-07-25 11:18:45.0
12  6041112222  300 Broadway  6040000000  10.1  2024-07-01 18:28:45.0
13  6041112222  400 Broadway  6041111111  27.1  2024-06-01 21:17:35.0
14  6041112222  500 Broadway  6043333333  33.37  2024-06-25 12:16:47.0
15  6041112222  600 Broadway  6044444444  29.38  2024-04-15 10:10:25.0
16  6045556666  100 Broadway  6040000000  14.25  2024-07-18 08:25:07.0
17  6045556666  200 Broadway  6045555555  28.15  2024-07-07 08:08:55.0
18  6045556666  500 Broadway  6044444444  39.09  2024-07-10 18:59:57.0
19  6045556666  600 Broadway  6041111111  46.55  2024-06-08 07:36:27.0
20  6049990000  100 Broadway  6043333333  38.63  2024-06-27 17:25:20.0
21  6049990000  300 Broadway  6041111111  42.16  2024-05-25 19:20:28.0
22  6049990000  400 Broadway  6040000000  35.01  2024-05-05 22:33:12.0
23  6049990000  500 Broadway  6046666666  22.83  2024-06-12 14:12:39.0
```

Payment Information (including PaymentAdvancedInfo and PaymentBasicInfo tables)



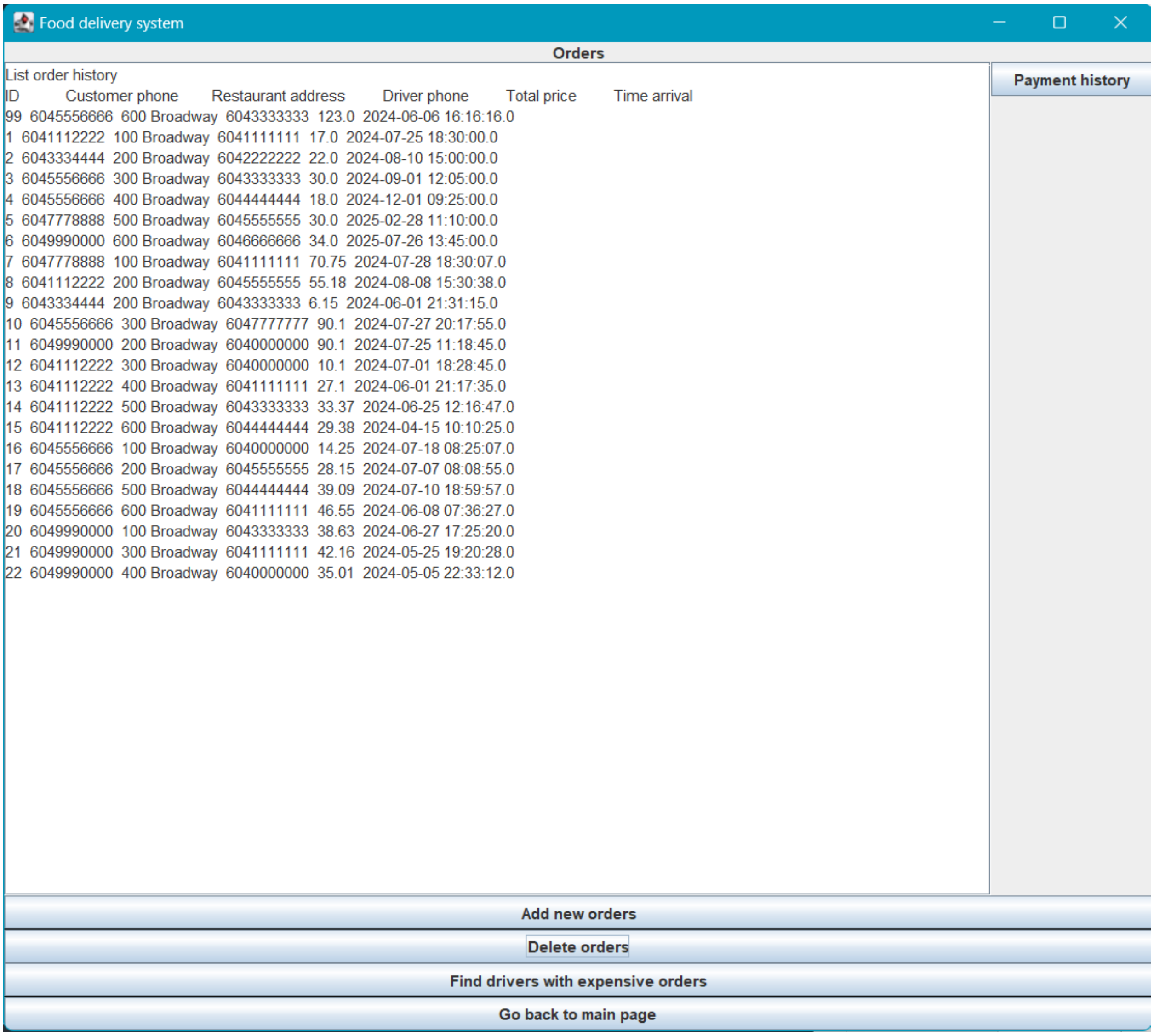
```
List payment history
ID        Status      Order ID      Price      Restaurant address      Driver phone
1  success  1  17.0  100 Broadway  6041111111
2  success  2  22.0  200 Broadway  6042222222
3  failure  3  30.0  300 Broadway  6043333333
4  success  3  30.0  300 Broadway  6043333333
5  success  4  18.0  400 Broadway  6044444444
6  failure  5  30.0  500 Broadway  6045555555
7  success  6  34.0  600 Broadway  6046666666
8  success  8  55.18  200 Broadway  6045555555
9  ongoing  9  6.15  200 Broadway  6043333333
10  success  10  90.1  300 Broadway  6047777777
11  ongoing  11  90.1  200 Broadway  6040000000
12  success  12  10.1  300 Broadway  6040000000
13  success  13  27.1  400 Broadway  6041111111
14  failure  14  33.37  500 Broadway  6043333333
15  success  15  29.38  600 Broadway  6044444444
16  success  16  14.25  100 Broadway  6040000000
17  failure  17  28.15  200 Broadway  6045555555
18  success  18  39.09  500 Broadway  6044444444
19  success  19  46.55  600 Broadway  6041111111
20  failure  20  38.63  100 Broadway  6043333333
21  success  21  42.16  300 Broadway  6041111111
22  success  22  35.01  400 Broadway  6040000000
23  success  23  22.83  500 Broadway  6046666666
```

Step 2: We enter Order ID as 23 to delete.



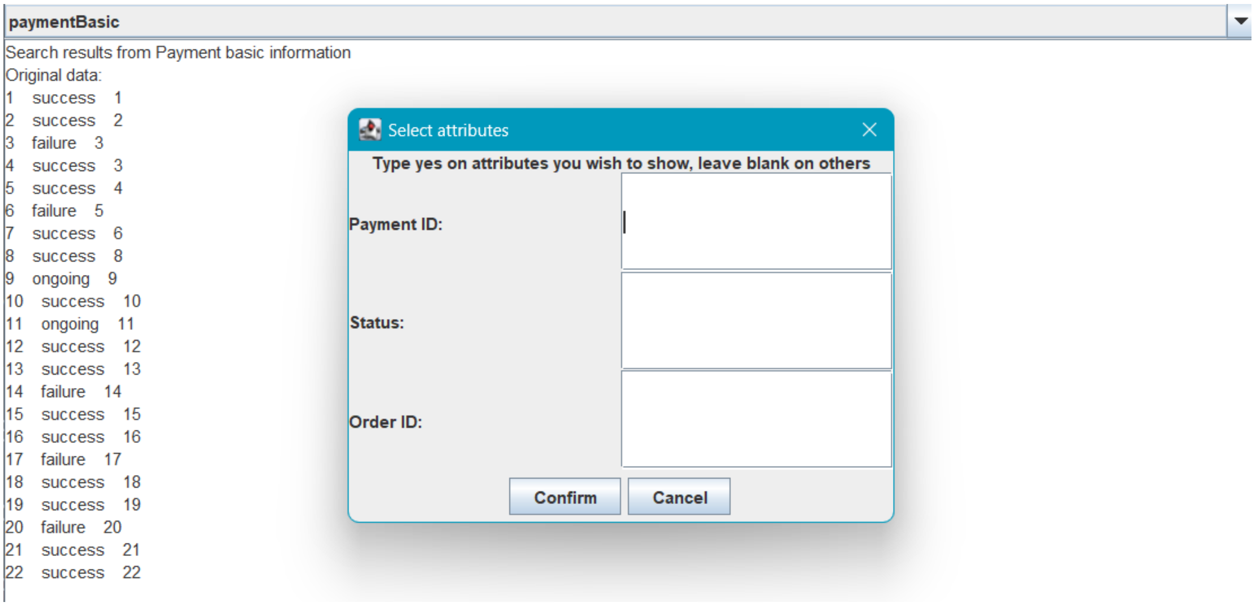
Order No: 23

Step 3: We can now see that the tuple with Order ID = 23 is deleted, and so is the related tuples in PaymentAdvancedInfo and PaymentBasicInfo tables.



```
List order history
ID        Customer phone      Restaurant address        Driver phone        Total price        Time arrival
99  6045556666  600 Broadway  6043333333  123.0  2024-06-06 16:16:16.0
1  6041112222  100 Broadway  6041111111  17.0  2024-07-25 18:30:00.0
2  6043334444  200 Broadway  6042222222  22.0  2024-08-10 15:00:00.0
3  6045556666  300 Broadway  6043333333  30.0  2024-09-01 12:05:00.0
4  6045556666  400 Broadway  6044444444  18.0  2024-12-01 09:25:00.0
5  6047778888  500 Broadway  6045555555  30.0  2025-02-28 11:10:00.0
6  6049990000  600 Broadway  6046666666  34.0  2025-07-26 13:45:00.0
7  6047778888  100 Broadway  6041111111  70.75  2024-07-28 18:30:07.0
8  6041112222  200 Broadway  6045555555  55.18  2024-08-08 15:30:38.0
9  6043334444  200 Broadway  6043333333  6.15  2024-06-01 21:31:15.0
10  6045556666  300 Broadway  6047777777  90.1  2024-07-27 20:17:55.0
11  6049990000  200 Broadway  6040000000  90.1  2024-07-25 11:18:45.0
12  6041112222  300 Broadway  6040000000  10.1  2024-07-01 18:28:45.0
13  6041112222  400 Broadway  6041111111  27.1  2024-06-01 21:17:35.0
14  6041112222  500 Broadway  6043333333  33.37  2024-06-25 12:16:47.0
15  6041112222  600 Broadway  6044444444  29.38  2024-04-15 10:10:25.0
16  6045556666  100 Broadway  6040000000  14.25  2024-07-18 08:25:07.0
17  6045556666  200 Broadway  6045555555  28.15  2024-07-07 08:08:55.0
18  6045556666  500 Broadway  6044444444  39.09  2024-07-10 18:59:57.0
19  6045556666  600 Broadway  6041111111  46.55  2024-06-08 07:36:27.0
20  6049990000  100 Broadway  6043333333  38.63  2024-06-27 17:25:20.0
21  6049990000  300 Broadway  6041111111  42.16  2024-05-25 19:20:28.0
22  6049990000  400 Broadway  6040000000  35.01  2024-05-05 22:33:12.0
```

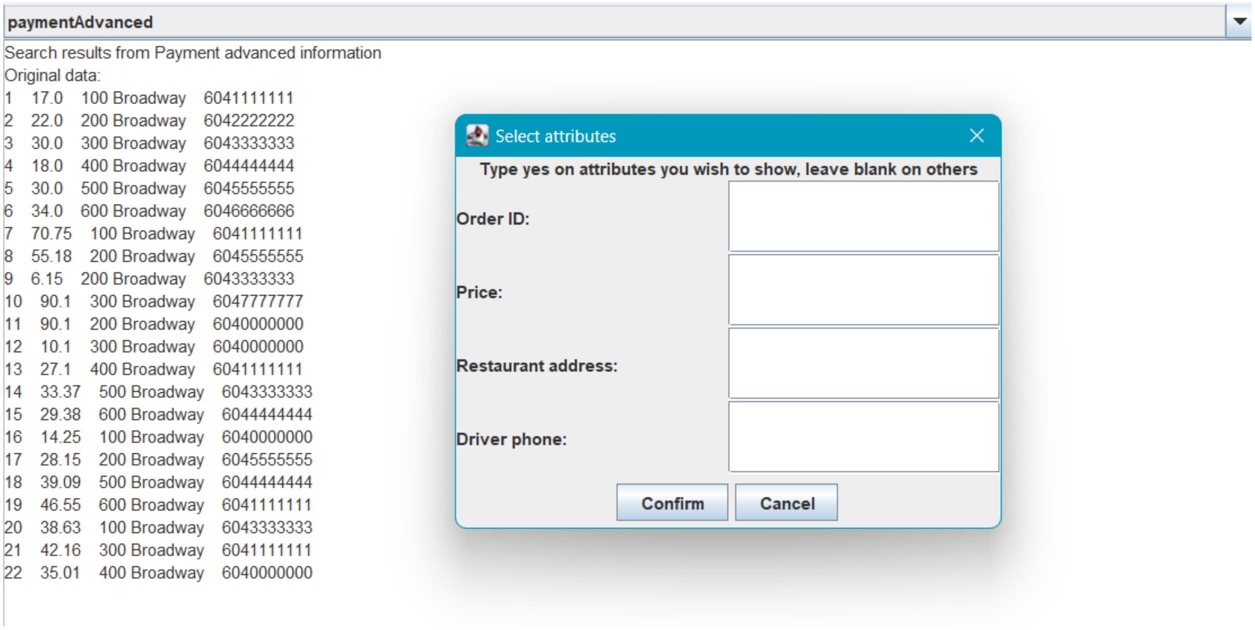
PaymentBasic(PaymentID, Status, OrderID)



```
Search results from Payment basic information
Original data:
1   success   1
2   success   2
3   failure   3
4   success   3
5   success   4
6   failure   5
7   success   6
8   success   8
9   ongoing   9
10   success   10
11   ongoing   11
12   success   12
13   success   13
14   failure   14
15   success   15
16   success   16
17   failure   17
18   success   18
19   success   19
20   failure   20
21   success   21
22   success   22
```

Type yes on attributes you wish to show, leave blank on others

PaymentAdvanced(OrderID, Price, Restautant Address, Driver Phone)



```
Search results from Payment advanced information
Original data:
1   17.0   100 Broadway   6041111111
2   22.0   200 Broadway   6042222222
3   30.0   300 Broadway   6043333333
4   18.0   400 Broadway   6044444444
5   30.0   500 Broadway   6045555555
6   34.0   600 Broadway   6046666666
7   70.75   100 Broadway   6041111111
8   55.18   200 Broadway   6045555555
9   6.15   200 Broadway   6043333333
10   90.1   300 Broadway   6047777777
11   90.1   200 Broadway   6040000000
12   10.1   300 Broadway   6040000000
13   27.1   400 Broadway   6041111111
14   33.37   500 Broadway   6043333333
15   29.38   600 Broadway   6044444444
16   14.25   100 Broadway   6040000000
17   28.15   200 Broadway   6045555555
18   39.09   500 Broadway   6044444444
19   46.55   600 Broadway   6041111111
20   38.63   100 Broadway   6043333333
21   42.16   300 Broadway   6041111111
22   35.01   400 Broadway   6040000000
```

Type yes on attributes you wish to show, leave blank on others

- **Function Reference**

  1. SQL Query String:

     Where: OrderService.java - line 54

     Method: public String[] delete(OrdersModel model)

2. Query Executed:
   Where: AppDOA.java - line 1225
   Method: public String[] deleteOrder(OrdersModel model, String query)

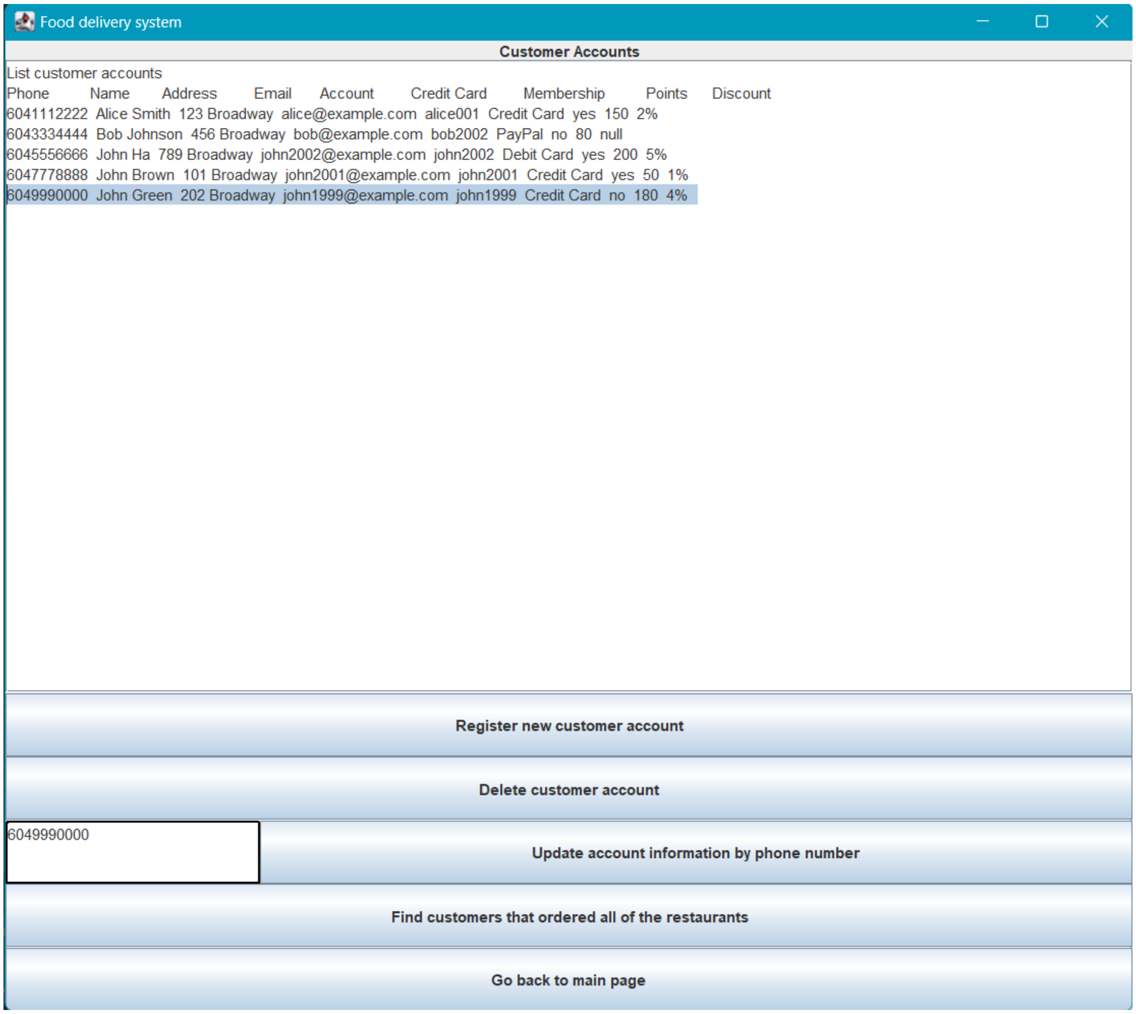
3. Update Operation

We do the UPDATE operation on the Customers Table.

SQL statement:

UPDATE Customers SET customer_name = 'Kevin Lee', customer_address = '303 Broadway', customer_email = 'KL123@example.com', customer_account = 'KLee2024', customer_paymentInfo = 'PayPal', customer_membership = 'yes', customer_points = 99, customer_discount = '1%' WHERE customer_phone = '6049990000';

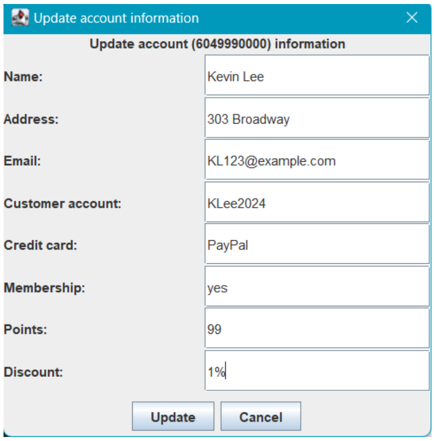
Step 1: We pick which tuple to update, here we pick customer_phone with 6049990000.


Food delivery system
Customer Accounts
Register new customer account
Delete customer account
Update account information by phone number
Find customers that ordered all of the restaurants
Go back to main page

```
List customer accounts
Phone       Name       Address       Email      Account       Credit Card       Membership       Points      Discount
6041112222  Alice Smith  123 Broadway  alice@example.com  alice001  Credit Card  yes  150  2%
6043334444  Bob Johnson  456 Broadway  bob@example.com  bob2002  PayPal  no  80  null
6045556666  John Ha  789 Broadway  john2002@example.com  john2002  Debit Card  yes  200  5%
6047778888  John Brown  101 Broadway  john2001@example.com  john2001  Credit Card  yes  50  1%
6049990000  John Green  202 Broadway  john1999@example.com  john1999  Credit Card  no  180  4%
```

6049990000

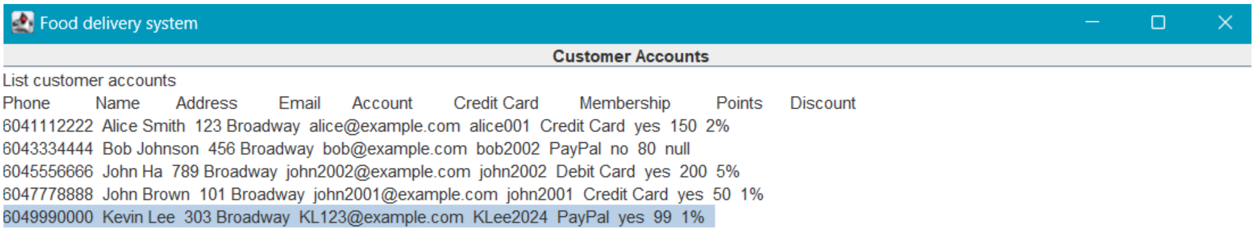
Step 2: We pick which tuple to update, here we pick customer_phone with 6049990000.



Update account (6049990000) information

Name:	Kevin Lee
Address:	303 Broadway
Email:	KL123@example.com
Customer account:	KLee2024
Credit card:	PayPal
Membership:	yes
Points:	99
Discount:	1%

Step 3: We see the corresponding changes in the GUI.



Customer Accounts

```
List customer accounts
Phone      Name      Address      Email      Account      Credit Card      Membership      Points      Discount
6041112222  Alice Smith  123 Broadway  alice@example.com  alice001  Credit Card  yes  150  2%
6043334444  Bob Johnson  456 Broadway  bob@example.com  bob2002  PayPal  no  80  null
6045556666  John Ha  789 Broadway  john2002@example.com  john2002  Debit Card  yes  200  5%
6047778888  John Brown  101 Broadway  john2001@example.com  john2001  Credit Card  yes  50  1%
6049990000  Kevin Lee  303 Broadway  KL123@example.com  KLee2024  PayPal  yes  99  1%
```

- **Function Reference**

  1. SQL Query String:

     Where: CustomerAccountService.java - line 65

     Method: public String[] update(CustomersModel model)

  2. Query Executed:

     Where: AppDOA.java - line 176

     Method: public String[] updateCustomer(CustomersModel model, String query)

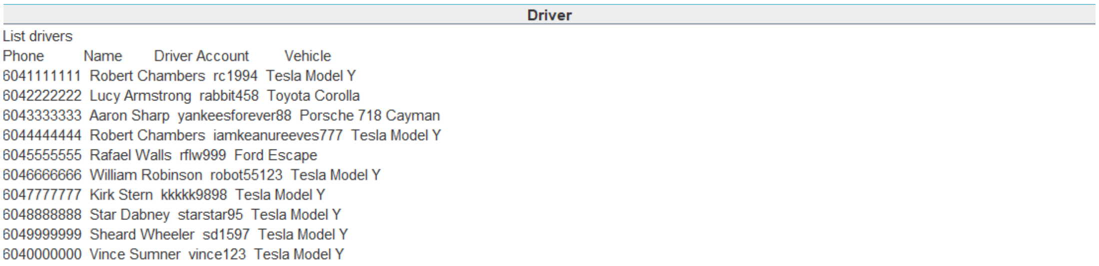
4. Selection Operation

We do the selection operation on the Drivers Table.

SQL statement:

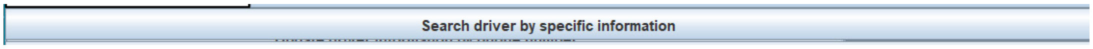
```
SELECT * FROM Drivers
WHERE driver_vehicle = 'Tesla Model Y';
```

Step 1: We see that the original table is as below, and click at the search driver button.

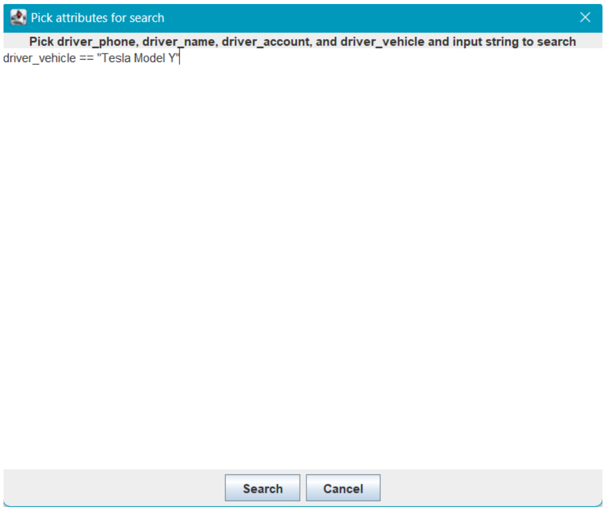


Driver

```
List drivers
Phone      Name      Driver Account      Vehicle
6041111111  Robert Chambers  rc1994  Tesla Model Y
6042222222  Lucy Armstrong  rabbit458  Toyota Corolla
6043333333  Aaron Sharp  yankeesforever88  Porsche 718 Cayman
6044444444  Robert Chambers  iamkeanureeves777  Tesla Model Y
6045555555  Rafael Walls  rflw999  Ford Escape
6046666666  William Robinson  robot55123  Tesla Model Y
6047777777  Kirk Stern  kkkkk9898  Tesla Model Y
6048888888  Star Dabney  starstar95  Tesla Model Y
6049999999  Sheard Wheeler  sd1597  Tesla Model Y
6040000000  Vince Sumner  vince123  Tesla Model Y
```



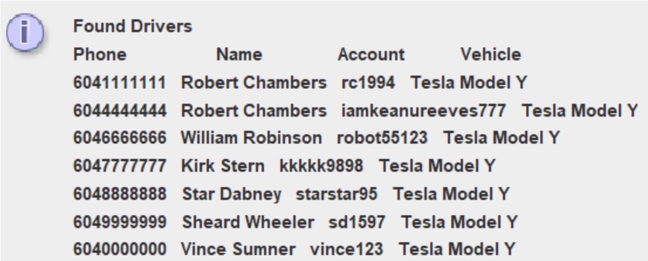
Step 2: We follow the instructions, and enter what we want kind of drivers we want to search.



Pick driver_phone, driver_name, driver_account, and driver_vehicle and input string to search

```
driver_vehicle == "Tesla Model Y"
```

Step 3: We get a list of tuples with the drivers that drives a "Tesla Model Y".



```
Found Drivers
Phone          Name          Account          Vehicle
6041111111  Robert Chambers  rc1994  Tesla Model Y
6044444444  Robert Chambers  iamkeanureeves777  Tesla Model Y
6046666666  William Robinson  robot55123  Tesla Model Y
6047777777  Kirk Stern  kkkkk9898  Tesla Model Y
6048888888  Star Dabney  starstar95  Tesla Model Y
6049999999  Sheard Wheeler  sd1597  Tesla Model Y
6040000000  Vince Sumner  vince123  Tesla Model Y
```

- **Function Reference**

  1. SQL Query String:
     Where: DriverService.java - line 116
     Method: public Object[] select(DriversModel model)
  2. Query Executed:
     Where: AppDOA.java - line 479
     Method: public Object[] selectDriver(String query)

5. Projection Operation

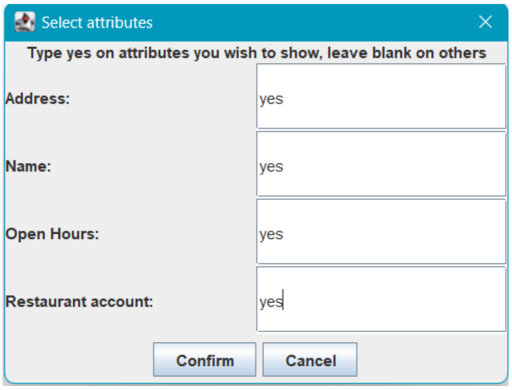
The projection operation works for any table in this application.

- **Example 1: Restaurants Table**

Case 1: We select all attributes to do projection.

SQL statement:

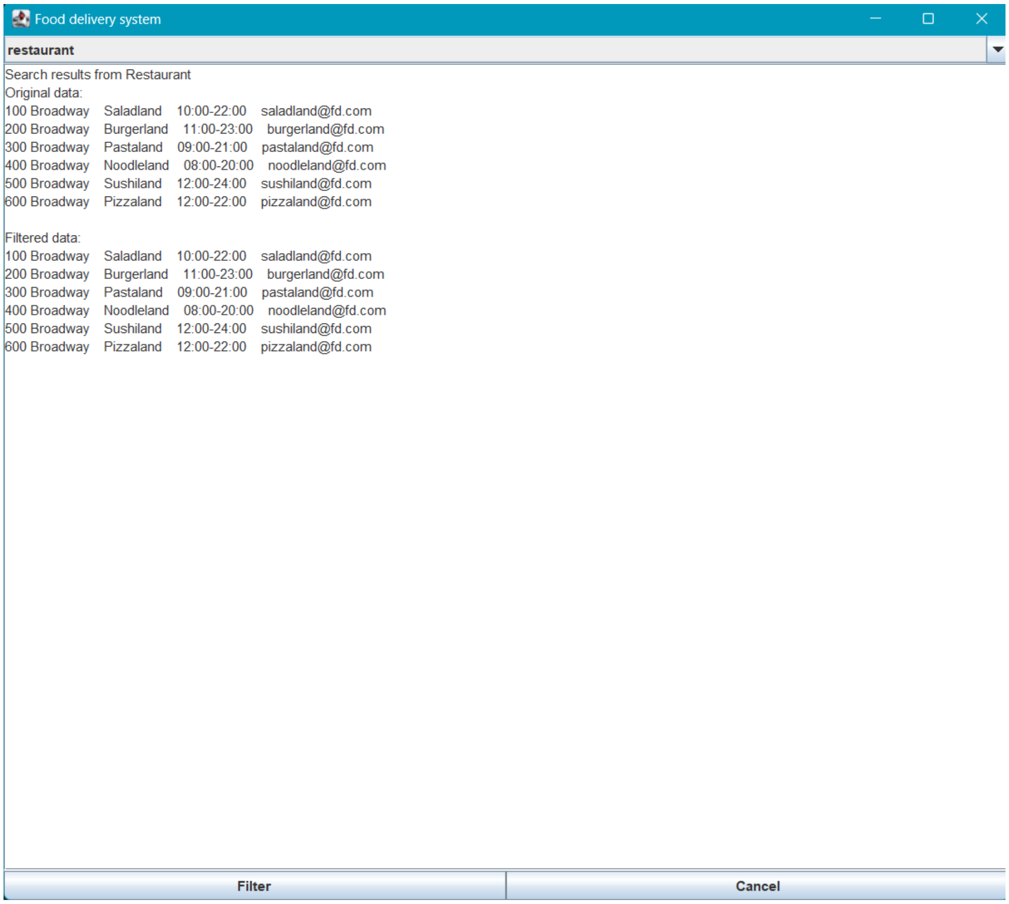
SELECT restaurant_address, restaurant_name, restaurant_openHours, restaurant_account FROM Restaurants



Select attributes

Type yes on attributes you wish to show, leave blank on others

Address:	yes
Name:	yes
Open Hours:	yes
Restaurant account:	yes



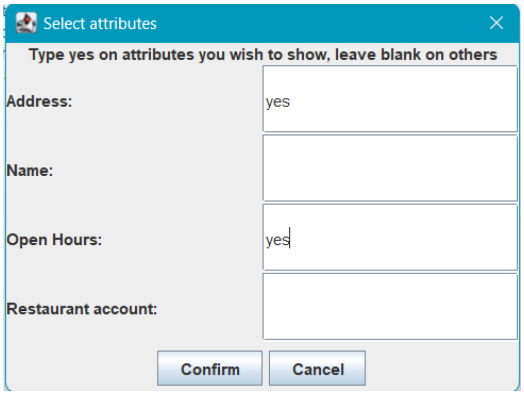
```
Search results from Restaurant
Original data:
100 Broadway   Saladland   10:00-22:00   saladland@fd.com
200 Broadway   Burgerland   11:00-23:00   burgerland@fd.com
300 Broadway   Pastaland   09:00-21:00   pastaland@fd.com
400 Broadway   Noodleland   08:00-20:00   noodleland@fd.com
500 Broadway   Sushiland   12:00-24:00   sushiland@fd.com
600 Broadway   Pizzaland   12:00-22:00   pizzaland@fd.com

Filtered data:
100 Broadway   Saladland   10:00-22:00   saladland@fd.com
200 Broadway   Burgerland   11:00-23:00   burgerland@fd.com
300 Broadway   Pastaland   09:00-21:00   pastaland@fd.com
400 Broadway   Noodleland   08:00-20:00   noodleland@fd.com
500 Broadway   Sushiland   12:00-24:00   sushiland@fd.com
600 Broadway   Pizzaland   12:00-22:00   pizzaland@fd.com
```

Case 2: We select some attributes to do projection.

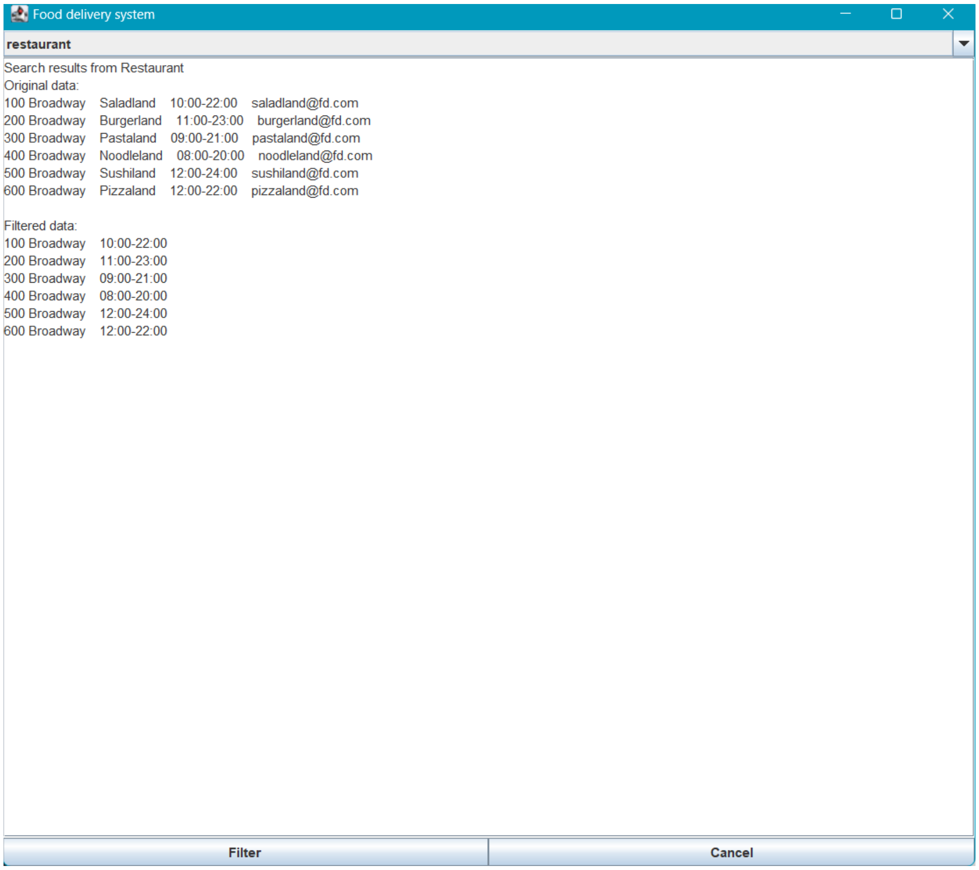
SQL statement:

SELECT restaurant_address, restaurant_openHours FROM Restaurants



Type yes on attributes you wish to show, leave blank on others

Address:	yes
Name:	
Open Hours:	yes
Restaurant account:	



```
Search results from Restaurant
Original data:
100 Broadway   Saladland    10:00-22:00   saladland@fd.com
200 Broadway   Burgerland   11:00-23:00   burgerland@fd.com
300 Broadway   Pastaland    09:00-21:00   pastaland@fd.com
400 Broadway   Noodleland   08:00-20:00   noodleland@fd.com
500 Broadway   Sushiland    12:00-24:00   sushiland@fd.com
600 Broadway   Pizzaland    12:00-22:00   pizzaland@fd.com

Filtered data:
100 Broadway   10:00-22:00
200 Broadway   11:00-23:00
300 Broadway   09:00-21:00
400 Broadway   08:00-20:00
500 Broadway   12:00-24:00
600 Broadway   12:00-22:00
```

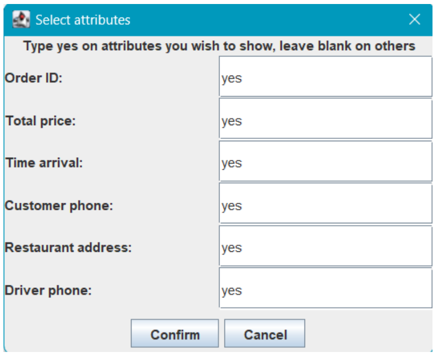
- **Example 2: Orders Table**

Case 1: We select all attributes to do projection.

SQL statement:

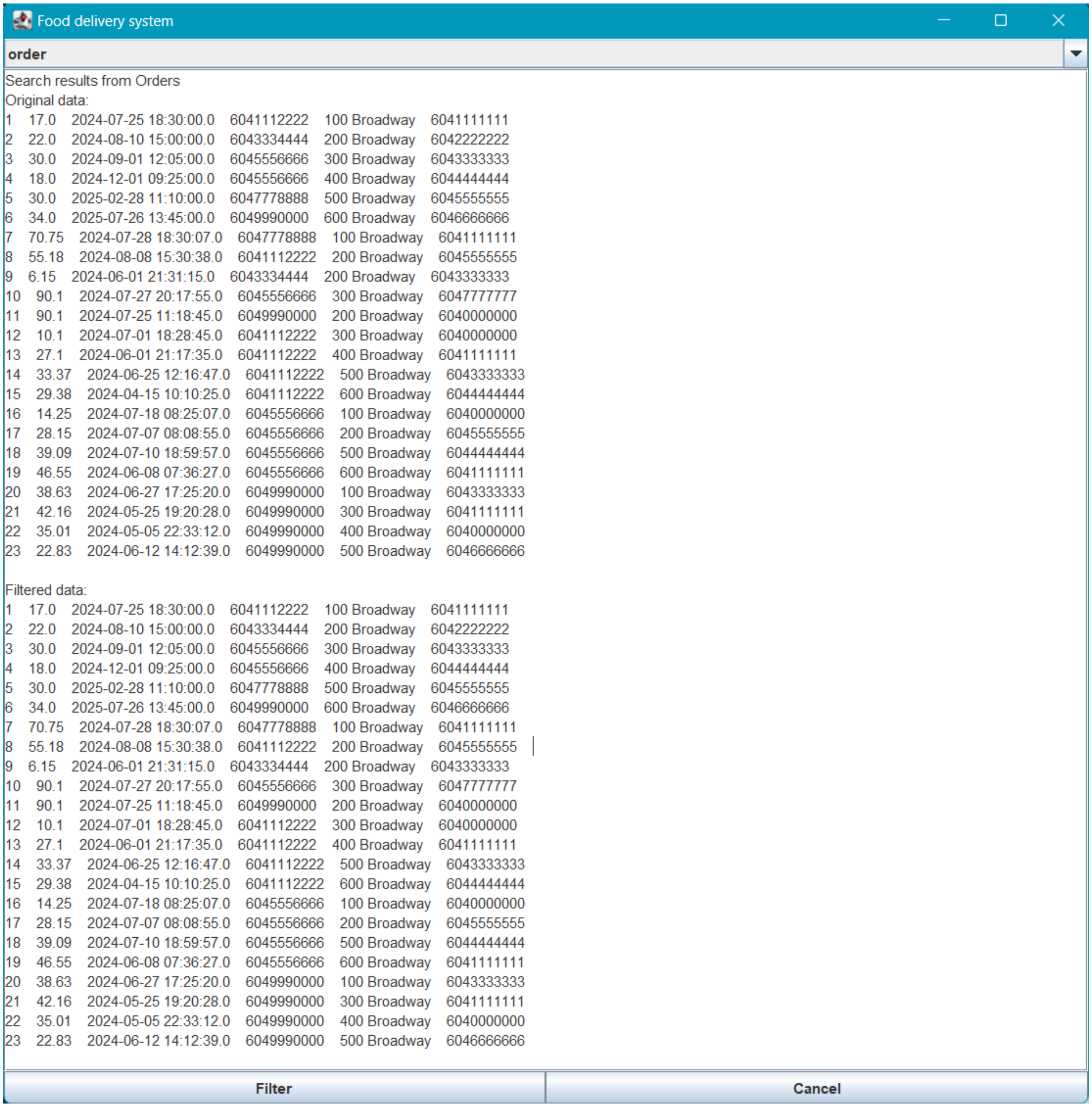
SELECT order_id, order_totalPrice, order_timeArrival, customer_phone, restaurant_address, driver_phone

FROM Orders



Type yes on attributes you wish to show, leave blank on others

Order ID:	yes
Total price:	yes
Time arrival:	yes
Customer phone:	yes
Restaurant address:	yes
Driver phone:	yes



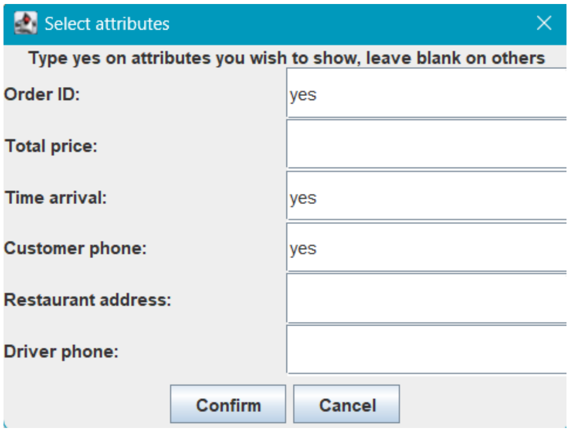
```
Search results from Orders
Original data:
1   17.0   2024-07-25 18:30:00.0   6041112222   100 Broadway   6041111111
2   22.0   2024-08-10 15:00:00.0   6043334444   200 Broadway   6042222222
3   30.0   2024-09-01 12:05:00.0   6045556666   300 Broadway   6043333333
4   18.0   2024-12-01 09:25:00.0   6045556666   400 Broadway   6044444444
5   30.0   2025-02-28 11:10:00.0   6047778888   500 Broadway   6045555555
6   34.0   2025-07-26 13:45:00.0   6049990000   600 Broadway   6046666666
7   70.75   2024-07-28 18:30:07.0   6047778888   100 Broadway   6041111111
8   55.18   2024-08-08 15:30:38.0   6041112222   200 Broadway   6045555555
9   6.15   2024-06-01 21:31:15.0   6043334444   200 Broadway   6043333333
10   90.1   2024-07-27 20:17:55.0   6045556666   300 Broadway   6047777777
11   90.1   2024-07-25 11:18:45.0   6049990000   200 Broadway   6040000000
12   10.1   2024-07-01 18:28:45.0   6041112222   300 Broadway   6040000000
13   27.1   2024-06-01 21:17:35.0   6041112222   400 Broadway   6041111111
14   33.37   2024-06-25 12:16:47.0   6041112222   500 Broadway   6043333333
15   29.38   2024-04-15 10:10:25.0   6041112222   600 Broadway   6044444444
16   14.25   2024-07-18 08:25:07.0   6045556666   100 Broadway   6040000000
17   28.15   2024-07-07 08:08:55.0   6045556666   200 Broadway   6045555555
18   39.09   2024-07-10 18:59:57.0   6045556666   500 Broadway   6044444444
19   46.55   2024-06-08 07:36:27.0   6045556666   600 Broadway   6041111111
20   38.63   2024-06-27 17:25:20.0   6049990000   100 Broadway   6043333333
21   42.16   2024-05-25 19:20:28.0   6049990000   300 Broadway   6041111111
22   35.01   2024-05-05 22:33:12.0   6049990000   400 Broadway   6040000000
23   22.83   2024-06-12 14:12:39.0   6049990000   500 Broadway   6046666666

Filtered data:
1   17.0   2024-07-25 18:30:00.0   6041112222   100 Broadway   6041111111
2   22.0   2024-08-10 15:00:00.0   6043334444   200 Broadway   6042222222
3   30.0   2024-09-01 12:05:00.0   6045556666   300 Broadway   6043333333
4   18.0   2024-12-01 09:25:00.0   6045556666   400 Broadway   6044444444
5   30.0   2025-02-28 11:10:00.0   6047778888   500 Broadway   6045555555
6   34.0   2025-07-26 13:45:00.0   6049990000   600 Broadway   6046666666
7   70.75   2024-07-28 18:30:07.0   6047778888   100 Broadway   6041111111
8   55.18   2024-08-08 15:30:38.0   6041112222   200 Broadway   6045555555
9   6.15   2024-06-01 21:31:15.0   6043334444   200 Broadway   6043333333
10   90.1   2024-07-27 20:17:55.0   6045556666   300 Broadway   6047777777
11   90.1   2024-07-25 11:18:45.0   6049990000   200 Broadway   6040000000
12   10.1   2024-07-01 18:28:45.0   6041112222   300 Broadway   6040000000
13   27.1   2024-06-01 21:17:35.0   6041112222   400 Broadway   6041111111
14   33.37   2024-06-25 12:16:47.0   6041112222   500 Broadway   6043333333
15   29.38   2024-04-15 10:10:25.0   6041112222   600 Broadway   6044444444
16   14.25   2024-07-18 08:25:07.0   6045556666   100 Broadway   6040000000
17   28.15   2024-07-07 08:08:55.0   6045556666   200 Broadway   6045555555
18   39.09   2024-07-10 18:59:57.0   6045556666   500 Broadway   6044444444
19   46.55   2024-06-08 07:36:27.0   6045556666   600 Broadway   6041111111
20   38.63   2024-06-27 17:25:20.0   6049990000   100 Broadway   6043333333
21   42.16   2024-05-25 19:20:28.0   6049990000   300 Broadway   6041111111
22   35.01   2024-05-05 22:33:12.0   6049990000   400 Broadway   6040000000
23   22.83   2024-06-12 14:12:39.0   6049990000   500 Broadway   6046666666
```

Case 2: We select some attributes to do projection.

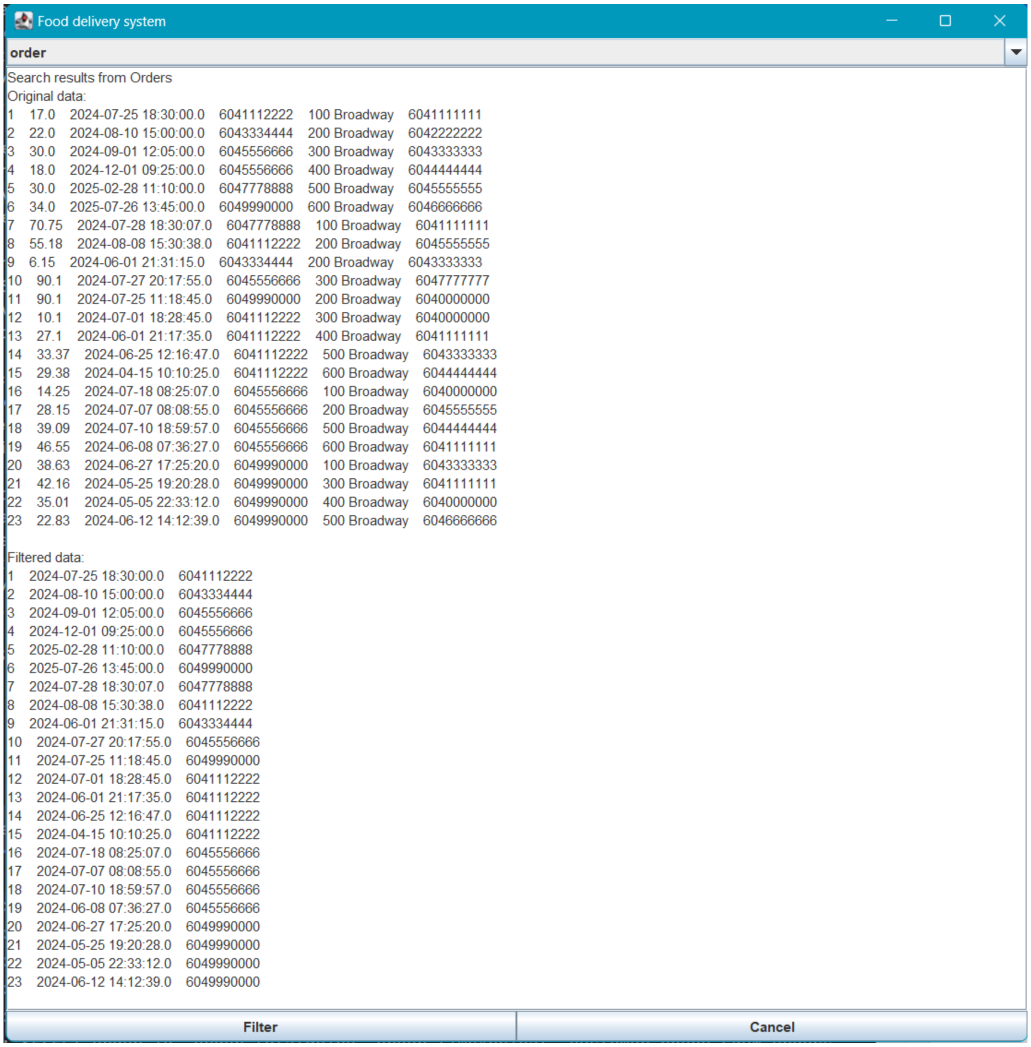
SQL statement:

SELECT order_id, order_totalPrice, order_timeArrival, customer_phone FROM Orders



Type yes on attributes you wish to show, leave blank on others

Order ID:	yes
Total price:	
Time arrival:	yes
Customer phone:	yes
Restaurant address:	
Driver phone:	



```
Search results from Orders
Original data:
1   17.0   2024-07-25 18:30:00.0   6041112222   100 Broadway   6041111111
2   22.0   2024-08-10 15:00:00.0   6043334444   200 Broadway   6042222222
3   30.0   2024-09-01 12:05:00.0   6045556666   300 Broadway   6043333333
4   18.0   2024-12-01 09:25:00.0   6045556666   400 Broadway   6044444444
5   30.0   2025-02-28 11:10:00.0   6047778888   500 Broadway   6045555555
6   34.0   2025-07-26 13:45:00.0   6049990000   600 Broadway   6046666666
7   70.75   2024-07-28 18:30:07.0   6047778888   100 Broadway   6041111111
8   55.18   2024-08-08 15:30:38.0   6041112222   200 Broadway   6045555555
9   6.15   2024-06-01 21:31:15.0   6043334444   200 Broadway   6043333333
10   90.1   2024-07-27 20:17:55.0   6045556666   300 Broadway   6047777777
11   90.1   2024-07-25 11:18:45.0   6049990000   200 Broadway   6040000000
12   10.1   2024-07-01 18:28:45.0   6041112222   300 Broadway   6040000000
13   27.1   2024-06-01 21:17:35.0   6041112222   400 Broadway   6041111111
14   33.37   2024-06-25 12:16:47.0   6041112222   500 Broadway   6043333333
15   29.38   2024-04-15 10:10:25.0   6041112222   600 Broadway   6044444444
16   14.25   2024-07-18 08:25:07.0   6045556666   100 Broadway   6040000000
17   28.15   2024-07-07 08:08:55.0   6045556666   200 Broadway   6045555555
18   39.09   2024-07-10 18:59:57.0   6045556666   500 Broadway   6044444444
19   46.55   2024-06-08 07:36:27.0   6045556666   600 Broadway   6041111111
20   38.63   2024-06-27 17:25:20.0   6049990000   100 Broadway   6043333333
21   42.16   2024-05-25 19:20:28.0   6049990000   300 Broadway   6041111111
22   35.01   2024-05-05 22:33:12.0   6049990000   400 Broadway   6040000000
23   22.83   2024-06-12 14:12:39.0   6049990000   500 Broadway   6046666666

Filtered data:
1   2024-07-25 18:30:00.0   6041112222
2   2024-08-10 15:00:00.0   6043334444
3   2024-09-01 12:05:00.0   6045556666
4   2024-12-01 09:25:00.0   6045556666
5   2025-02-28 11:10:00.0   6047778888
6   2025-07-26 13:45:00.0   6049990000
7   2024-07-28 18:30:07.0   6047778888
8   2024-08-08 15:30:38.0   6041112222
9   2024-06-01 21:31:15.0   6043334444
10   2024-07-27 20:17:55.0   6045556666
11   2024-07-25 11:18:45.0   6049990000
12   2024-07-01 18:28:45.0   6041112222
13   2024-06-01 21:17:35.0   6041112222
14   2024-06-25 12:16:47.0   6041112222
15   2024-04-15 10:10:25.0   6041112222
16   2024-07-18 08:25:07.0   6045556666
17   2024-07-07 08:08:55.0   6045556666
18   2024-07-10 18:59:57.0   6045556666
19   2024-06-08 07:36:27.0   6045556666
20   2024-06-27 17:25:20.0   6049990000
21   2024-05-25 19:20:28.0   6049990000
22   2024-05-05 22:33:12.0   6049990000
23   2024-06-12 14:12:39.0   6049990000
```

- **Function Reference**

  1. SQL Query String:

     Where: [Service Name].java

     e.g. CustomerAccountService.java

     Method: public Object[] project([Service Type Model] model)

     e.g. public Object[] project(CustomersModel model)

  2. Query Executed:

     Where: AppDOA.java

     Method: public Object[] project[Service Name]([Service Type Model] model, String query)

     e.g. public Object[] projectCustomer(CustomersModel model, String query)

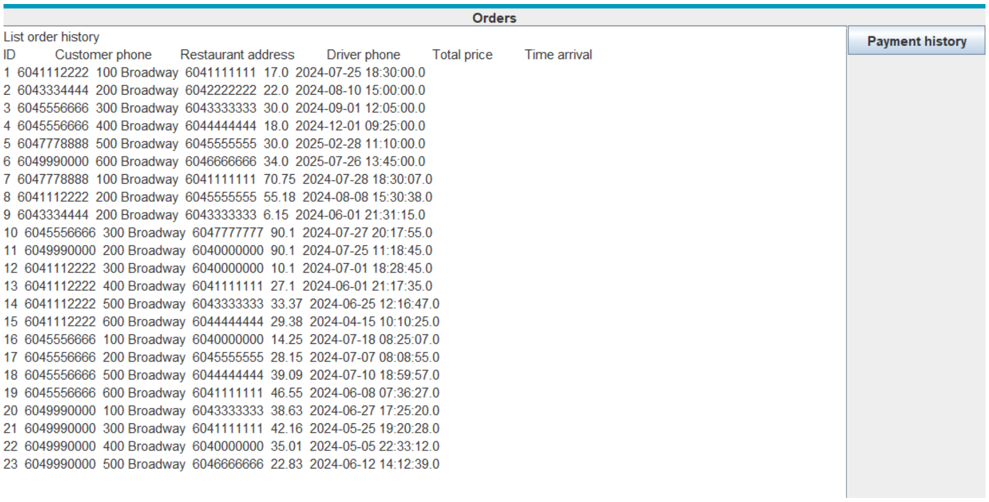
6. Join Operation

- **SQL statement**

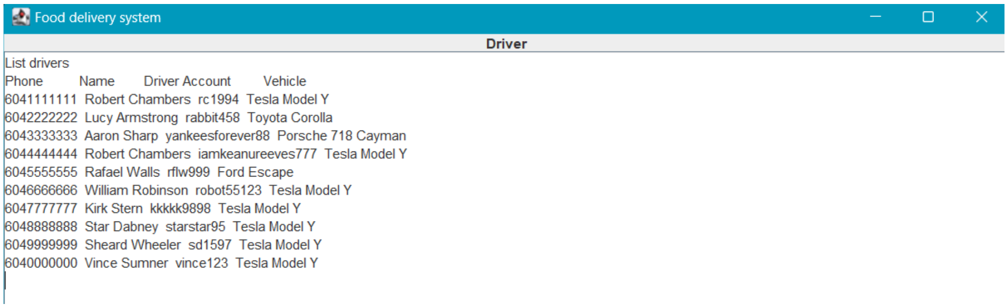
SELECT DISTINCT D.driver_phone, D.driver_name, D.driver_account, D.driver_vehicle
FROM Orders O, Drivers D
WHERE O.driver_phone = D.driver_phone AND O.order_totalPrice > 50.00";

- **Implementation**

Before: Orders, Drivers Table

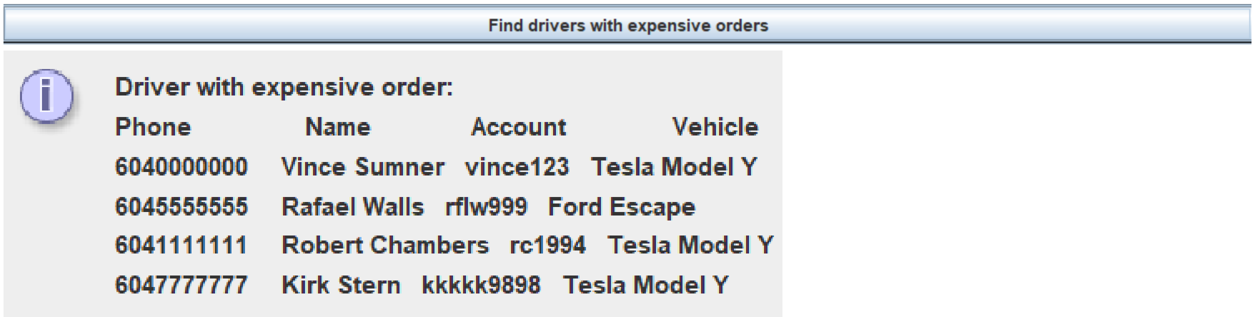


```
List order history
ID        Customer phone      Restaurant address      Driver phone      Total price      Time arrival
1  6041112222  100 Broadway  6041111111  17.0  2024-07-25 18:30:00.0
2  6043334444  200 Broadway  6042222222  22.0  2024-08-10 15:00:00.0
3  6045556666  300 Broadway  6043333333  30.0  2024-09-01 12:05:00.0
4  6045556666  400 Broadway  6044444444  18.0  2024-12-01 09:25:00.0
5  6047778888  500 Broadway  6045555555  30.0  2025-02-28 11:10:00.0
6  6049990000  600 Broadway  6046666666  34.0  2025-07-26 13:45:00.0
7  6047778888  100 Broadway  6041111111  70.75  2024-07-28 18:30:07.0
8  6041112222  200 Broadway  6045555555  55.18  2024-08-08 15:30:38.0
9  6043334444  200 Broadway  6043333333  6.15  2024-06-01 21:31:15.0
10  6045556666  300 Broadway  6047777777  90.1  2024-07-27 20:17:55.0
11  6049990000  200 Broadway  6040000000  90.1  2024-07-25 11:18:45.0
12  6041112222  300 Broadway  6040000000  10.1  2024-07-01 18:28:45.0
13  6041112222  400 Broadway  6041111111  27.1  2024-06-01 21:17:35.0
14  6041112222  500 Broadway  6043333333  33.37  2024-06-25 12:16:47.0
15  6041112222  600 Broadway  6044444444  29.38  2024-04-15 10:10:25.0
16  6045556666  100 Broadway  6040000000  14.25  2024-07-18 08:25:07.0
17  6045556666  200 Broadway  6045555555  28.15  2024-07-07 08:08:55.0
18  6045556666  500 Broadway  6044444444  39.09  2024-07-10 18:59:57.0
19  6045556666  600 Broadway  6041111111  46.55  2024-06-08 07:36:27.0
20  6049990000  100 Broadway  6043333333  38.63  2024-06-27 17:25:20.0
21  6049990000  300 Broadway  6041111111  42.16  2024-05-25 19:20:28.0
22  6049990000  400 Broadway  6040000000  35.01  2024-05-05 22:33:12.0
23  6049990000  500 Broadway  6046666666  22.83  2024-06-12 14:12:39.0
```



```
List drivers
Phone       Name      Driver Account       Vehicle
6041111111  Robert Chambers  rc1994  Tesla Model Y
6042222222  Lucy Armstrong  rabbit458  Toyota Corolla
6043333333  Aaron Sharp  yankeesforever88  Porsche 718 Cayman
6044444444  Robert Chambers  iamkeanureeves777  Tesla Model Y
6045555555  Rafael Walls  rflw999  Ford Escape
6046666666  William Robinson  robot55123  Tesla Model Y
6047777777  Kirk Stern  kkkkk9898  Tesla Model Y
6048888888  Star Dabney  starstar95  Tesla Model Y
6049999999  Sheard Wheeler  sd1597  Tesla Model Y
6040000000  Vince Sumner  vince123  Tesla Model Y
```

After: The Query Result



```
Driver with expensive order:
Phone          Name           Account          Vehicle
6040000000   Vince Sumner   vince123   Tesla Model Y
6045555555   Rafael Walls   rflw999   Ford Escape
6041111111   Robert Chambers   rc1994   Tesla Model Y
6047777777   Kirk Stern   kkkkk9898   Tesla Model Y
```

- **Function Reference**

  1. SQL Query String:

     Where: OrderService.java - line 96

     Method: public DriversModel[] join()

  2. Query Executed:

     Where: AppDOA.java - line 1293

     Method: public DriversModel[] joinOrder(String query)

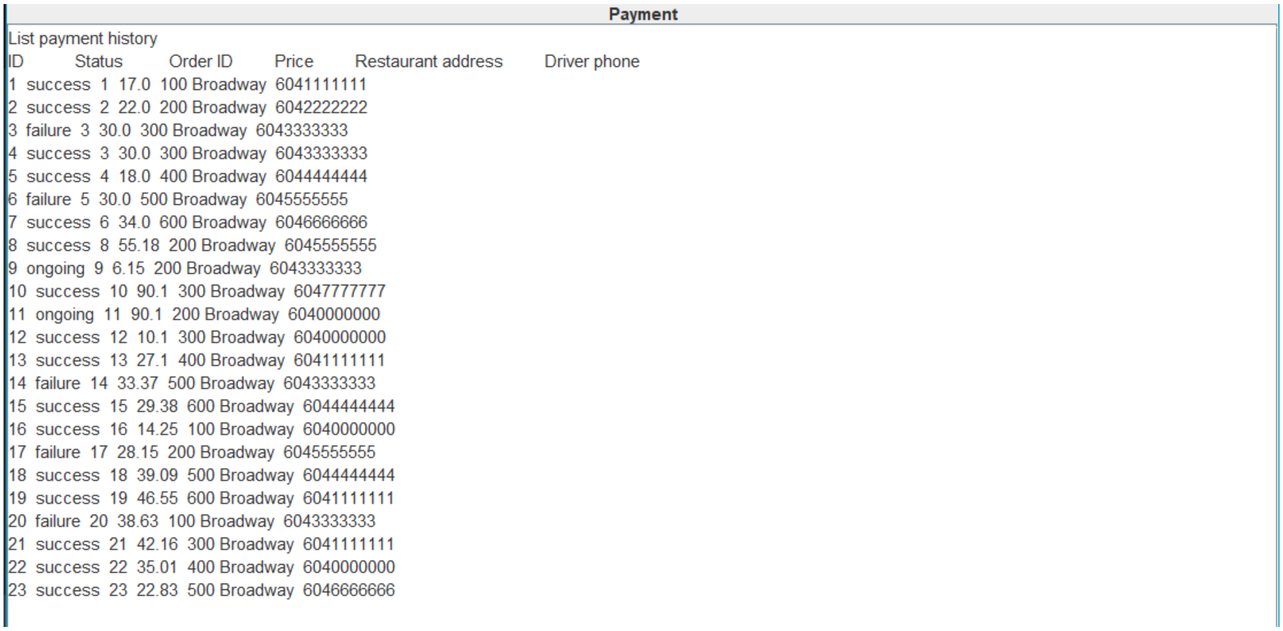
7. Aggregation with GROUP BY Operation

- **SQL statement**

```
SELECT payment_status, COUNT(payment_id) AS number_of_successful_payments
FROM PaymentBasicInfo
GROUP BY payment_status;
```

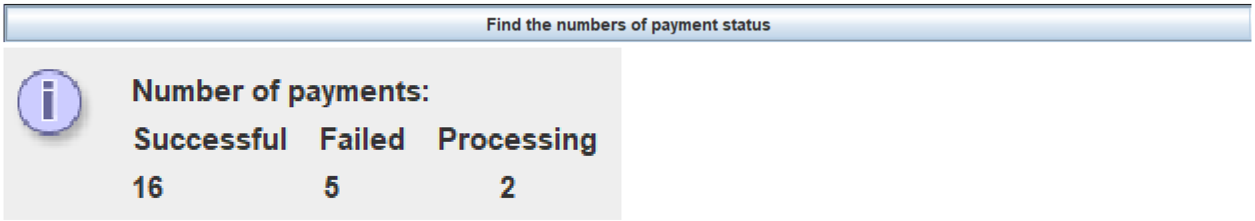
- **Implementation**

Before: Payment Information (Including PaymentBasicInfo Table)



```
List payment history
ID        Status       Order ID      Price       Restaurant address        Driver phone
1  success  1  17.0  100 Broadway  6041111111
2  success  2  22.0  200 Broadway  6042222222
3  failure  3  30.0  300 Broadway  6043333333
4  success  3  30.0  300 Broadway  6043333333
5  success  4  18.0  400 Broadway  6044444444
6  failure  5  30.0  500 Broadway  6045555555
7  success  6  34.0  600 Broadway  6046666666
8  success  8  55.18  200 Broadway  6045555555
9  ongoing  9  6.15  200 Broadway  6043333333
10  success  10  90.1  300 Broadway  6047777777
11  ongoing  11  90.1  200 Broadway  6040000000
12  success  12  10.1  300 Broadway  6040000000
13  success  13  27.1  400 Broadway  6041111111
14  failure  14  33.37  500 Broadway  6043333333
15  success  15  29.38  600 Broadway  6044444444
16  success  16  14.25  100 Broadway  6040000000
17  failure  17  28.15  200 Broadway  6045555555
18  success  18  39.09  500 Broadway  6044444444
19  success  19  46.55  600 Broadway  6041111111
20  failure  20  38.63  100 Broadway  6043333333
21  success  21  42.16  300 Broadway  6041111111
22  success  22  35.01  400 Broadway  6040000000
23  success  23  22.83  500 Broadway  6046666666
```

After: The Query Result



```
Number of payments:
Successful    Failed    Processing
16            5         2
```

- **Function Reference**

1. SQL Query String:
   Where: PaymentBasicService.java - line 78
   Method: public Object[] aggregationGroupBy()
2. Query Executed:
   Where: AppDOA.java - line 1395
   Method: public Object[] aggregationGroupByPaymentBasicInfo(String query)

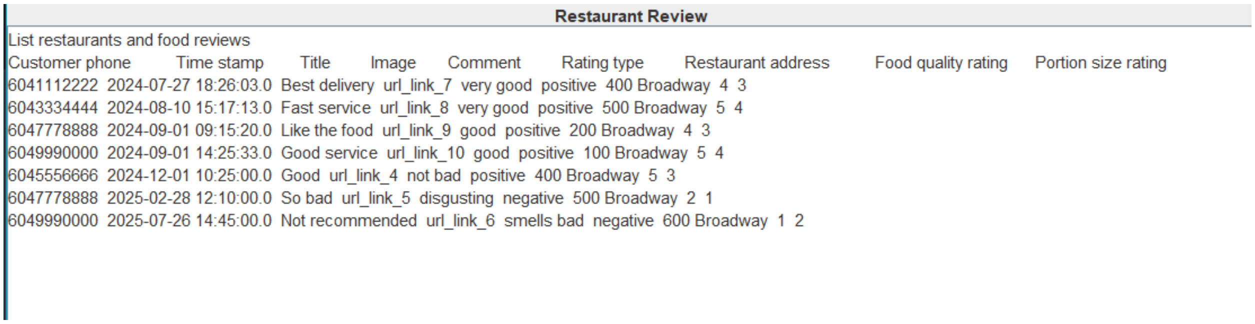
8. Aggregation with HAVING Operation

- **SQL statement**

SELECT restaurant_address, MAX(foodReviews_foodQualityRating) AS highest_food_quality_rating
FROM FoodReviews
GROUP BY restaurant_address
HAVING MAX(foodReviews_foodQualityRating) > 3;

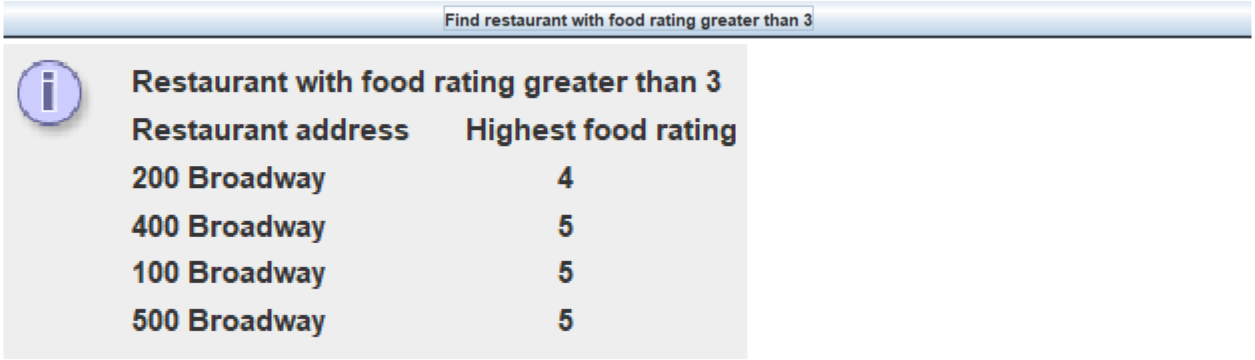
- **Implementation**

Before: FoodReviews Table



```
List restaurants and food reviews
Customer phone     Time stamp     Title      Image     Comment      Rating type     Restaurant address      Food quality rating     Portion size rating
6041112222  2024-07-27 18:26:03.0  Best delivery  url_link_7  very good  positive  400 Broadway  4  3
6043334444  2024-08-10 15:17:13.0  Fast service  url_link_8  very good  positive  500 Broadway  5  4
6047778888  2024-09-01 09:15:20.0  Like the food  url_link_9  good  positive  200 Broadway  4  3
6049990000  2024-09-01 14:25:33.0  Good service  url_link_10  good  positive  100 Broadway  5  4
6045556666  2024-12-01 10:25:00.0  Good  url_link_4  not bad  positive  400 Broadway  5  3
6047778888  2025-02-28 12:10:00.0  So bad  url_link_5  disgusting  negative  500 Broadway  2  1
6049990000  2025-07-26 14:45:00.0  Not recommended  url_link_6  smells bad  negative  600 Broadway  1  2
```

After: The Query Result



Restaurant with food rating greater than 3

Restaurant address	Highest food rating
200 Broadway	4
400 Broadway	5
100 Broadway	5
500 Broadway	5

- **Function Reference**

1. SQL Query String:
   Where: FoodReviewService.java - line 97
   Method: public Object[] aggregationHaving()
2. Query Executed:
   Where: AppDOA.java - line 1697
   Method: public Object[] aggregationHavingFoodReviews(String query)

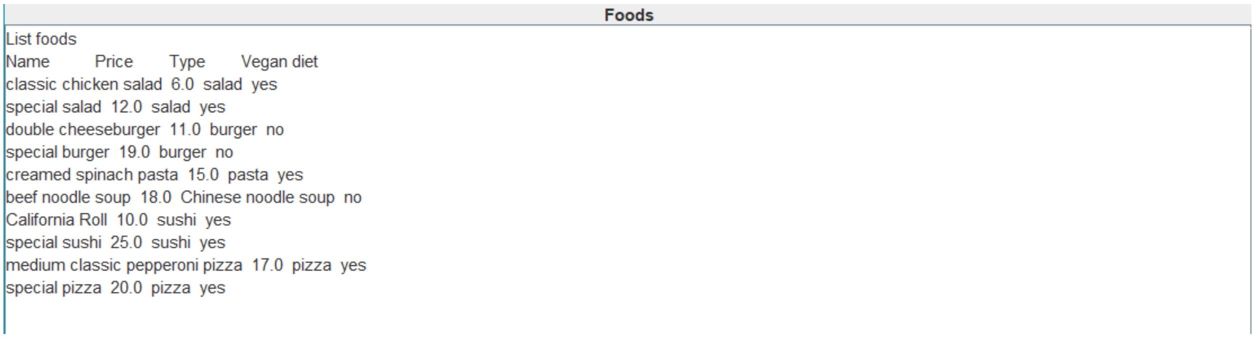
9. Nested Aggregation with GROUP BY

- **SQL statement**

```
SELECT FB.food_type, MIN(FB.food_price) AS lowest_food_price
FROM FoodBasicInfo FB
WHERE FB.food_name IN (SELECT DISTINCT O.food_name
                       FROM Ordered O
                       WHERE O.orderedQuantity > 5)
GROUP BY FB.food_type;
```

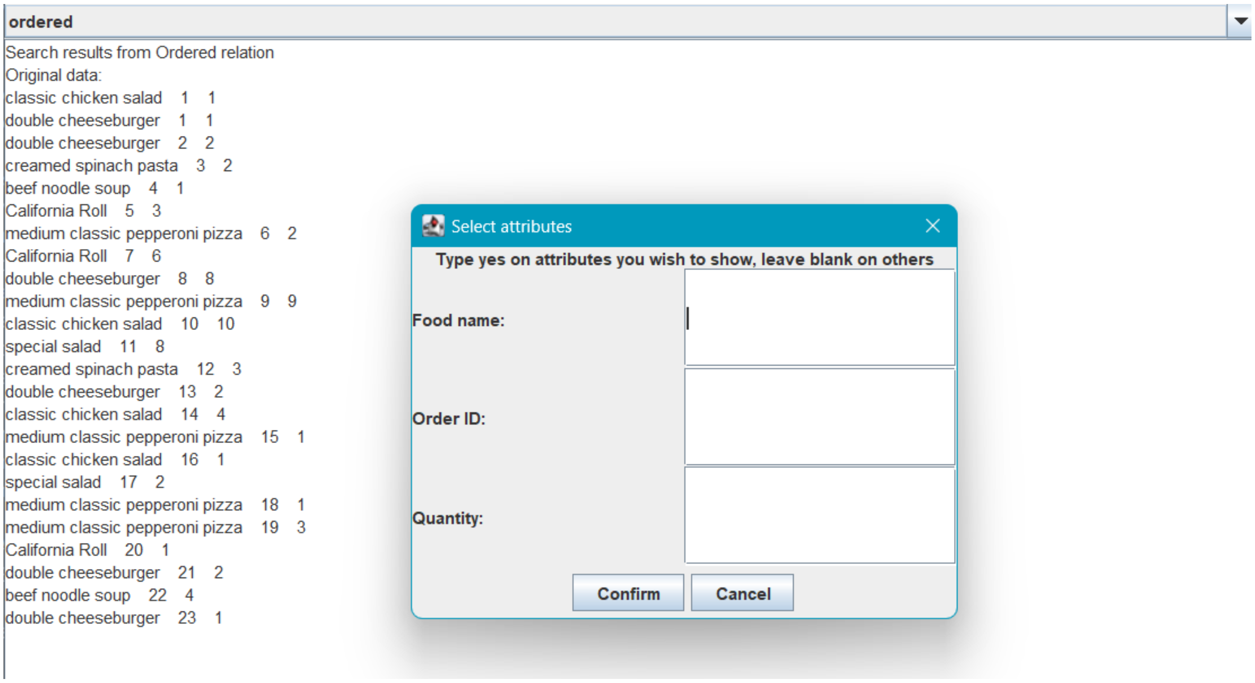
- **Implementation**

Before: Food Information (including FoodBasicInfo), Ordered Table



```
List foods
Name        Price     Type       Vegan diet
classic chicken salad  6.0  salad  yes
special salad  12.0  salad  yes
double cheeseburger  11.0  burger  no
special burger  19.0  burger  no
creamed spinach pasta  15.0  pasta  yes
beef noodle soup  18.0  Chinese noodle soup  no
California Roll  10.0  sushi  yes
special sushi  25.0  sushi  yes
medium classic pepperoni pizza  17.0  pizza  yes
special pizza  20.0  pizza  yes
```

Orderd Table: Food name, OrderID, Quantity.



```
Search results from Ordered relation
Original data:
classic chicken salad   1   1
double cheeseburger   1   1
double cheeseburger   2   2
creamed spinach pasta   3   2
beef noodle soup   4   1
California Roll   5   3
medium classic pepperoni pizza   6   2
California Roll   7   6
double cheeseburger   8   8
medium classic pepperoni pizza   9   9
classic chicken salad   10   10
special salad   11   8
creamed spinach pasta   12   3
double cheeseburger   13   2
classic chicken salad   14   4
medium classic pepperoni pizza   15   1
classic chicken salad   16   1
special salad   17   2
medium classic pepperoni pizza   18   1
medium classic pepperoni pizza   19   3
California Roll   20   1
double cheeseburger   21   2
beef noodle soup   22   4
double cheeseburger   23   1
```

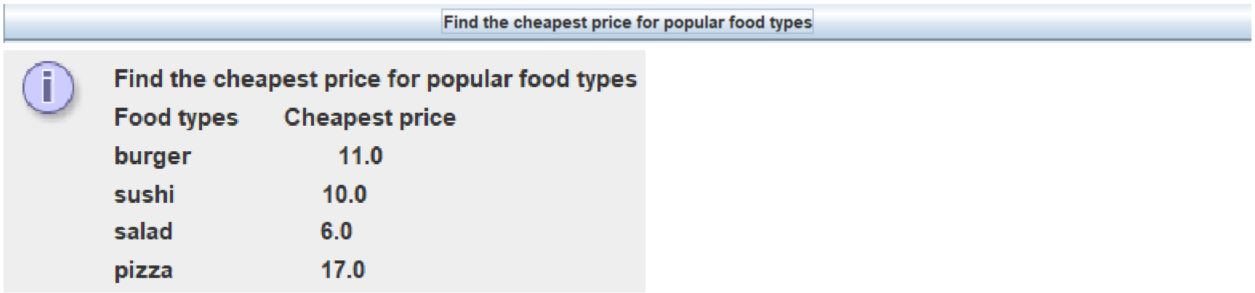
Type yes on attributes you wish to show, leave blank on others

Food name:

Order ID:

Quantity:

After: The Query Result



```
Find the cheapest price for popular food types

Find the cheapest price for popular food types
Food types      Cheapest price
burger              11.0
sushi              10.0
salad              6.0
pizza              17.0
```

- **Function Reference**

  1. SQL Query String:
     Where: FoodBasicService.java - line 110
     Method: public Object[] nestedAggregation()
  2. Query Executed:
     Where: AppDOA.java - line 811
     Method: Object[] nestedAggregationFoodBasicInfo(String query)

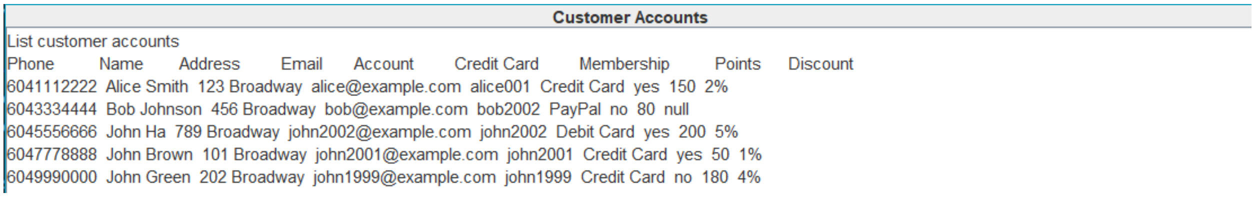
10. Division Operation

- **SQL statement**

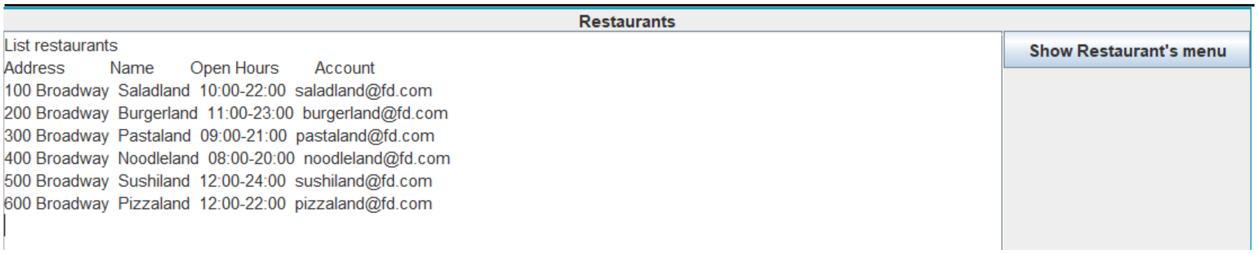
```
SELECT C.customer_phone, C.customer_name
FROM Customers C
WHERE NOT EXISTS (SELECT R.restaurant_address
                  FROM Restaurants R " +
                  WHERE NOT EXISTS (SELECT O.customer_phone
                                    FROM Orders O
                                    WHERE R.restaurant_address = O.restaurant_address
                                    AND O.customer_phone = C.customer_phone));
```

- **Implementation**

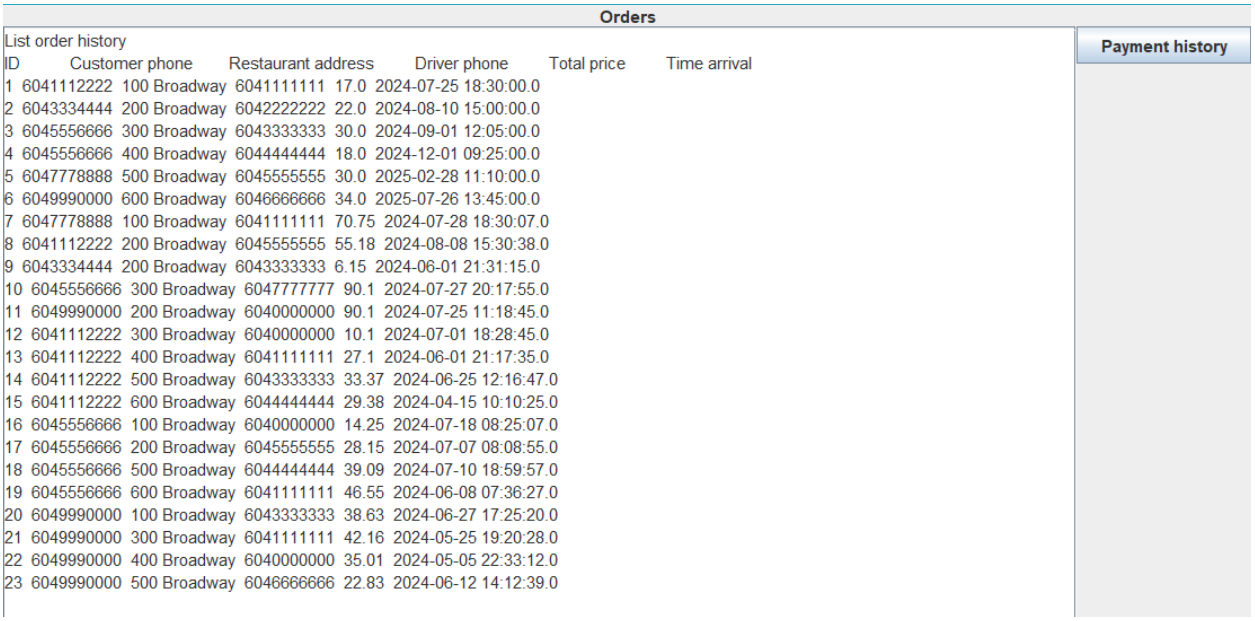
Before: Customers, Restaurants, Orders Table



```
Customer Accounts
List customer accounts
Phone      Name      Address      Email     Account      Credit Card      Membership      Points     Discount
6041112222  Alice Smith  123 Broadway  alice@example.com  alice001  Credit Card  yes  150  2%
6043334444  Bob Johnson  456 Broadway  bob@example.com  bob2002  PayPal  no  80  null
6045556666  John Ha  789 Broadway  john2002@example.com  john2002  Debit Card  yes  200  5%
6047778888  John Brown  101 Broadway  john2001@example.com  john2001  Credit Card  yes  50  1%
6049990000  John Green  202 Broadway  john1999@example.com  john1999  Credit Card  no  180  4%
```

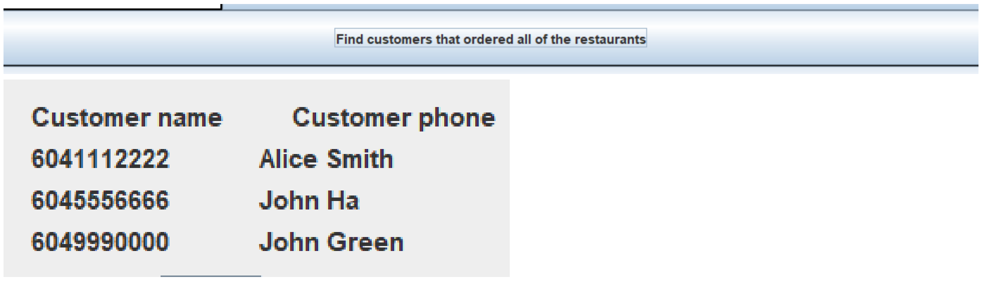


```
List restaurants
Address        Name       Open Hours       Account
100 Broadway  Saladland  10:00-22:00  saladland@fd.com
200 Broadway  Burgerland  11:00-23:00  burgerland@fd.com
300 Broadway  Pastaland  09:00-21:00  pastaland@fd.com
400 Broadway  Noodleland  08:00-20:00  noodleland@fd.com
500 Broadway  Sushiland  12:00-24:00  sushiland@fd.com
600 Broadway  Pizzaland  12:00-22:00  pizzaland@fd.com
```



```
List order history
ID       Customer phone      Restaurant address      Driver phone      Total price      Time arrival
1  6041112222  100 Broadway  6041111111  17.0  2024-07-25 18:30:00.0
2  6043334444  200 Broadway  6042222222  22.0  2024-08-10 15:00:00.0
3  6045556666  300 Broadway  6043333333  30.0  2024-09-01 12:05:00.0
4  6045556666  400 Broadway  6044444444  18.0  2024-12-01 09:25:00.0
5  6047778888  500 Broadway  6045555555  30.0  2025-02-28 11:10:00.0
6  6049990000  600 Broadway  6046666666  34.0  2025-07-26 13:45:00.0
7  6047778888  100 Broadway  6041111111  70.75  2024-07-28 18:30:07.0
8  6041112222  200 Broadway  6045555555  55.18  2024-08-08 15:30:38.0
9  6043334444  200 Broadway  6043333333  6.15  2024-06-01 21:31:15.0
10  6045556666  300 Broadway  6047777777  90.1  2024-07-27 20:17:55.0
11  6049990000  200 Broadway  6040000000  90.1  2024-07-25 11:18:45.0
12  6041112222  300 Broadway  6040000000  10.1  2024-07-01 18:28:45.0
13  6041112222  400 Broadway  6041111111  27.1  2024-06-01 21:17:35.0
14  6041112222  500 Broadway  6043333333  33.37  2024-06-25 12:16:47.0
15  6041112222  600 Broadway  6044444444  29.38  2024-04-15 10:10:25.0
16  6045556666  100 Broadway  6040000000  14.25  2024-07-18 08:25:07.0
17  6045556666  200 Broadway  6045555555  28.15  2024-07-07 08:08:55.0
18  6045556666  500 Broadway  6044444444  39.09  2024-07-10 18:59:57.0
19  6045556666  600 Broadway  6041111111  46.55  2024-06-08 07:36:27.0
20  6049990000  100 Broadway  6043333333  38.63  2024-06-27 17:25:20.0
21  6049990000  300 Broadway  6041111111  42.16  2024-05-25 19:20:28.0
22  6049990000  400 Broadway  6040000000  35.01  2024-05-05 22:33:12.0
23  6049990000  500 Broadway  6046666666  22.83  2024-06-12 14:12:39.0
```

After: The Query Result



```
Customer name       Customer phone
6041112222         Alice Smith
6045556666         John Ha
6049990000         John Green
```

- **Function Reference**

  1. SQL Query String:
     Where: CustomerAccountService.java - line 135
     Method: public Object[] division()
  2. Query Executed:
     Where: AppDOA.java - line 294
     Method: public Object[] divisionCustomer(String query)