

TABLE OF CONTENTS

1.	OVERVIEW.....	5
1.1.	About Food Delivery service.....	5
1.2.	About GrabFood.....	5
2.	BUSINESS RULES.....	6
2.1.	As a User.....	6
2.2.	As a Customer.....	6
2.3.	As a Driver.....	7
2.4.	As a Restaurant.....	7
3.	ENTITY RELATIONSHIP DIAGRAM (ERD)	7
4.	LIST OF TABLES	8
4.1.	Table AUDIT.....	8
4.2.	Table USERS.....	9
4.3.	Table CUSTOMERS	9
4.4.	Table DRIVERS.....	10
4.5.	Table LOCATIONS.....	11
4.6.	Table RESTAURANTS.....	12
4.7.	Table FOODS.....	13
4.8.	Table ORDERS	14
4.9.	Table STATUS.....	18
4.10.	Table ORDER_DETAILS.....	18
4.11.	Table VOUCHERS.....	19
4.12.	Table PAYMENTS	20
4.13.	Table RATINGS	21
5.	QUERY	22
5.1.	SELECT.....	22
5.2.	VIEW	28
6.	TRIGGERS	31
6.1.	TRIGGER ADD_SHIP_COST.....	31
6.2.	TRIGGER ADD_RES_COST	31
6.3.	TRIGGER ADD_PRICE_OD	31
6.4.	TRIGGER ADD_VOUCHER	31

6.5.	TRIGGER UPDATE_LOCATION	32
6.6.	TRIGGER STOCK_LEFT	32
6.7.	TRIGGER UPDATE_MAX_USE.....	32
6.8.	TRIGGER TOTAL_PRICE	32
6.9.	TRIGGER UPDATE CANCEL	32
6.10.	TRIGGER UPDATE_DRIVER_RES	32
6.11.	TRIGGER ADD_STAR	32
7.	PROCEDURES.....	32
7.1.	PROCEDURE UPDATE_STATUS.....	33
7.2.	PROCEDURE POP_MESSAGE	33
7.3.	PROCEDURE LAT_LO	33
7.4.	PROCEDURE GET_SHIP_COST	33
7.5.	PROCEDURE AGE_GROUP_FOOD	33
7.6.	PROCEDURE GROUP_AGE_F	33
7.7.	PROCEDURE AGE_GROUP_SPENDING	33
7.8.	PROCEDURE GROUP_AGE_S.....	33

1. OVERVIEW

1.1. About Food Delivery service

According to Euromonitor International, food delivery via app is expected to reach 38 million USD, growing 11% on average in next 5 years. Kantar TNS estimates that the revenue of the online food delivery market could reach 449 million USD by 2023. These days, the F&B industry in general and the restaurant industry no longer focus on "fighting" too much based on product quality, they began to shift direction and gradually compete fiercely through the quality of customer experience they provide. Concretely, in this case, it is the experience of using Food Delivery service for diners.

Using Food Delivery service is more and more popular compared to traditional business. According to statistics of Q&Me, 75% of the total people surveyed in Vietnam use the restaurant's Food Delivery service. The more positive the delivery experience, the higher the percentage of shoppers who will choose the restaurant again. According to Delivering Consumer Choice by MetaPack, up to 96% of consumers return home if they feel that delivery experience is positive.

Currently, people's use of Food Delivery is increasingly driven by the convenience and rise of many platforms such as GrabFood, Go-Food, LoShip, Baemin, etc. Nowadays, busy lifestyles become more popular. As a common point of many people, Food Delivery has obviously become a basic service that every restaurant should have.

1.2. About GrabFood

a) Introduction

GrabFood is a service that connects local food businesses to people. Grab has always brought people to things that matter to them, and they are now committed to bringing things that matter closer to people. When customers place an order with GrabFood, they receive it, send it off to the restaurant and assign a GrabFood delivery partner to pick up the order to bring it to them. By the way, Grab's advantage is that it already has a force of GrabBike drivers, then the service should be expanded. GrabFood is a more convenient segment in the Grab ecosystem, which both

helps increase income for drivers and helps users become more attached to the Grab application besides hitchhiking.

Thus, with the view of that point, we determine to choose topic “Food Delivery – GrabFood” and build up demo database of this system.

b) How does GrabFood work?

With the opening of more service the Grab application will have an additional Grab Food section. Food and beverage stores associated with Grab will be displayed on the application with the registered menu. Customers can type in the search box in the Grab app to find the food they want and choose to purchase. In addition to the fee for the dish paid by the Grab driver in advance, the user will bear additional shipping costs.

c) Basic function of GrabFood

In this project, our database would simulate 2 basic functions of GrabFood.

- *Order food:* To order food on GrabFood, customers need to follow these steps:
 - Access the Grab application and select the Food service.
 - Search and choose a restaurant, otherwise customers can directly choose the name of the dish they want to eat.
 - Order
 - View the food basket, if do not want to change, just switch to the payment section.
 - Enter Promo code (if any)
 - When customers have a specific request for the dish, they can add it to the note for the driver.
 - When receiving the order, the driver will go to the restaurant to pick up the food according to the order and deliver it.
- *Promotion:* GrabFood encourage customers to make more orders by voucher code, coupon, support the discount campaign of restaurants, etc.

2. BUSINESS RULES

2.1. As a User

- Users make account on Grab mobile app.
- Users can decide whether to be a driver (Grab's partner) or a customer.

2.2. As a Customer

- Customers can book an order via the app and choose whatever they want to eat. But they can order foods in only one restaurant at a time.
- They will decide which payment methods they are going to use. Except for COD, every other payment method usually come with a discount voucher which can lower the total price.
- If their order is in stage 1 (which is ORDERED), they can withdraw it and get 100% refunded.
- Sometimes, lucky customers might get free vouchers up to 70%.
- After receiving the food, customers can rate both the food and the driver.
- If customers want to order some foods for someone, they only need to type in an appointed location, pay for the order and they are good to go.

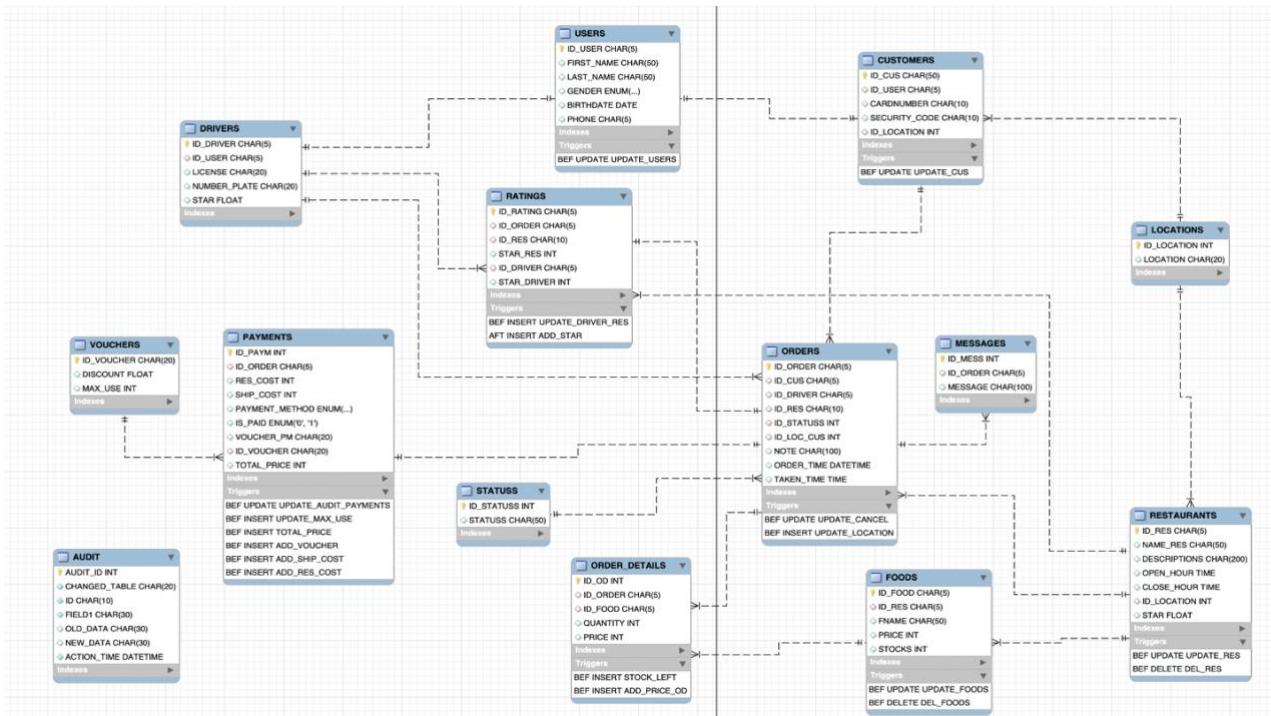
2.3. As a Driver

- Drivers will take the order and go pick up the food from the restaurant. They will be paid depends on the distance from the customer's location to the restaurant's address.
- After each order, drivers will receive stars from customers' ratings. The more stars the driver gets, the more reliable he becomes.

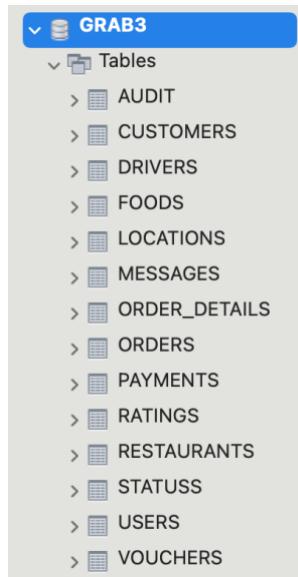
2.4. As a Restaurant

Restaurant will take the order and cook the food. But they have limited stock for each dish, so they are unable to cook unless customers order less than the remaining stocks.

3. ENTITY RELATIONSHIP DIAGRAM (ERD)



4. LIST OF TABLES



Our database was created by 14 tables: AUDIT, CUSTOMERS, DRIVERS, FOODS, LOCATIONS, MESSAGES, ORDER_DETAILS, ORDERS, PAYMENTS, RATINGS, RESTAURANTS, STATUS, USERS, VOUCHERS. The remain of this chapter would describe the data fields in each table in detail. The script of creating database and tables that consists of data are in GRAB_DATABASE_FINAL.sql file.

4.1. Table AUDIT

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the schema is set to 'test mysql'. The left sidebar shows the 'GRAB3' schema with various tables like AUDIT, CUSTOMERS, etc. The main area displays the SQL code for creating the AUDIT table:

```

CREATE TABLE AUDIT (
    AUDIT_ID INT NOT NULL AUTO_INCREMENT,
    CHANGED_TABLE CHAR(20) NOT NULL,
    ID CHAR(10) NOT NULL,
    FIELD1 CHAR(30) NOT NULL,
    OLD_DATA CHAR(30) DEFAULT NULL,
    NEW_DATA CHAR(30) DEFAULT NULL,
    ACTION_TIME DATETIME NOT NULL,
)
PRIMARY KEY(AUDIT_ID);

SELECT * FROM GRAB3.AUDIT;

```

Below the code, a 'Result Grid' shows the data for the AUDIT table. The columns are AUDIT_ID, CHANGED_TABLE, ID, FIELD1, OLD_DATA, NEW_DATA, and ACTION_TIME. The data consists of 10 rows of food stock updates:

AUDIT_ID	CHANGED_TABLE	ID	FIELD1	OLD_DATA	NEW_DATA	ACTION_TIME
1	FOOD	FC01	UPDATE FOOD STOCK	4277	4222	2021-06-06 20:52:26
2	FOOD	FF01	UPDATE FOOD STOCK	4972	4970	2021-06-06 20:52:26
3	FOOD	BG02	UPDATE FOOD STOCK	1352	1351	2021-06-06 20:52:26
4	FOOD	CF10	UPDATE FOOD STOCK	1052	1060	2021-06-06 20:52:26
5	FOOD	SW10	UPDATE FOOD STOCK	2996	2994	2021-06-06 20:52:26
6	FOOD	SC09	UPDATE FOOD STOCK	1833	1834	2021-06-06 20:52:26
7	FOOD	BS09	UPDATE FOOD STOCK	1459	1455	2021-06-06 20:52:26
8	FOOD	SL09	UPDATE FOOD STOCK	2910	2908	2021-06-06 20:52:26
9	FOOD	BG02	UPDATE FOOD STOCK	1351	1345	2021-06-06 20:52:26
10	FOOD	FF02	UPDATE FOOD STOCK	3543	3535	2021-06-06 20:52:26

At the bottom, a query history shows the execution of 'SELECT * FROM GRAB3.AUDIT LIMIT 0, 1000' with a duration of 0.00067 sec / 0.0003...

Table AUDIT will store every change in other tables in the database:

- AUDIT_ID: Primary Key
- CHANGED_TABLE: name of the table that is changed
- ID: ID of the object that received the change
- FIELD1: accomplished action
- OLD_DATA: data before changing
- NEW_DATA: data after changing
- ACTION_TIME: time when applying changes

4.2. Table USERS

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the schema is set to 'test mysql'. The left sidebar shows the 'Schemas' tree, with 'GRAB3' selected. Under 'GRAB3', there are several tables: AUDIT, CUSTOMERS, DRIVERS, FOODS, LOCATIONS, MESSAGES, ORDER_DETAILS, ORDERS, PAYMENTS, RATINGS, RESTAURANTS, STATUS, USERS, and VOUCHERS. Below the schemas tree, there are sections for 'Views', 'Stored Procedures', and 'Functions'. The 'Object Info' tab is selected, showing the current schema as 'GRAB3'. The main workspace displays two queries:

```

1  * CREATE TABLE USERS(
2      ID_USER CHAR(5) PRIMARY KEY,
3      FIRST_NAME CHAR(50),
4      LAST_NAME CHAR(50),
5      GENDER ENUM('MALE', 'FEMALE', 'OTHER'),
6      BIRTHDATE DATE,
7      PHONE CHAR(5)
8  );
9
10 *  SELECT * FROM GRAB3.USERS;

```

Below the queries is a 'Result Grid' showing the data from the 'USERS' table. The columns are ID_USER, FIRST_NAME, LAST_NAME, GENDER, BIRTHDATE, and PHONE. The data includes 10 rows of user information.

ID_USER	FIRST_NAME	LAST_NAME	GENDER	BIRTHDATE	PHONE
1001	Tuan	Phung	MALE	1971-04-12	94576
1002	Tuyet Anh	Dao	FEMALE	1998-01-23	17898
1003	Huyen	Tran	FEMALE	1972-01-18	36406
1004	Le Co	Nguyen	FEMALE	1983-09-28	75417
1005	Tu Anh	Nguyen	FEMALE	1988-03-04	69995
1006	Tue	Luu	MALE	1978-10-03	89336
1007	Hung	Quoc	MALE	1978-05-15	98293
1008	Duong	Uy	MALE	1978-06-25	85256
1009	The Linh	Pham	FEMALE	1984-12-04	91410
1010	Phung	Phan	MALE	1973-10-17	78894

At the bottom of the interface, the status bar shows 'Query Completed'.

- ID_USER: ID of each user (Primary Key)
- FIRST_NAME: first name of the user
- LAST_NAME: last name of the user
- GENDER: gender of the user
- BIRTHDATE: birthdate of the user
- PHONE: telephone number

Concretely, in USERS, we divide them into two main groups: CUSTOMERS and DRIVERS. Since not only they have some private information that couldn't be shared to anyone else but also, they have different purposes, one is served, and the others serves.

4.3. Table CUSTOMERS

```

CREATE TABLE CUSTOMERS (
    ID_CUS CHAR(50) PRIMARY KEY,
    ID_USER CHAR(5),
    CARDNUMBER CHAR(10),
    SECURITY_CODE CHAR(10),
    ID_LOCATION INT,
);

SELECT * FROM GRAB3.CUSTOMERS;

```

ID_CUS	ID_USER	CARDNUMBER	SECURITY_CODE	ID_LOCATION
CUS01	1001	86284	22944	5
CUS02	1002	95680	14551	3
CUS03	1003	99810	33121	8
CUS04	1004	11312	22981	1
CUS05	1005	41973	93869	12
CUS06	1006	21719	46593	6
CUS07	1007	29467	30302	5
CUS08	1008	53537	54556	7
CUS09	1009	18770	66921	12
CUS10	1010	24589	77129	7

Action Output Response Duration / Fetch Time
 3 15:27:09 SELECT * FROM GRAB3.USERS LIMIT 0, 1000 50 row(s) returned 0.00061 sec / 0.0000...

- ID_CUS: ID of each customer (Primary Key)
- ID_USER: ID of each user (Foreign Key from USERS)
- CARDNUMBER: credit card number
- SECURITY_CODE: identity card number
- ID_LOCATION: location of the customer (Foreign Key from LOCATIONS)

4.4. Table DRIVERS

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is GRAB3.
- Tables:** The table being viewed is DRIVERS.
- Table Definition:**

```

CREATE TABLE DRIVERS (
    ID_DRIVER CHAR(5) PRIMARY KEY,
    ID_USER CHAR(5),
    LICENSE CHAR(20),
    NUMBER_PLATE CHAR(20),
    STAR FLOAT,
)
FOREIGN KEY (ID_USER) REFERENCES USERS(ID_USER) ON UPDATE RESTRICT ON DELETE CASCADE

```
- Result Grid:** Shows the data for the DRIVERS table.

ID_DRIVER	ID_USER	LICENSE	NUMBER_PLATE	STAR
DVR01	1096	ABC01010101	30A5981	3.16667
DVR02	1037	FLG20940595	29C2095	3
DVR03	1038	GNV4095040	36B04854	3.66667
DVR04	1039	EKR39403830	21F03849	3.33333
DVR05	1040	FKS1475839	17A93741	2
DVR06	1041	TTH24799374	6789378	3
DVR07	1042	H4H02020843	30A5989	2.2
DVR08	1043	AMC02020845	50C9784	3.33333
DVR09	1044	BMW20308403	48929748	4
DVR10	1045	FKQ208302225	11Q39572	3
- Log:**

Action Output	Time	Action	Response	Duration / Fetch Time
4	15:30:31	SELECT * FROM GRAB3.DRIVERS LIMIT 0, 1000	15 row(s) returned	0.00066 sec / 0.000...

- ID_DRIVER: ID of each driver (Primary Key)
- ID_USER: ID of each (Foreign Key from USERS)
- NUMBER_PLATE: the number plate of the motorbike that driver used to deliver.
- LICENSE: verify driver
- STAR: evaluate delivery quality of driver. Concretely, it will be added after every succeed order through a trigger called ADD_STAR. It will calculate the average star in the RATINGS table that has ID_DRIVER.

4.5. Table LOCATIONS

The screenshot shows the MySQL Workbench interface with the schema **GRAB3** selected. In the central pane, the **LOCATIONS** table is being created with the following SQL code:

```

1 • CREATE TABLE LOCATIONS(
2     ID_LOCATION INT AUTO_INCREMENT PRIMARY KEY,
3     LOCATION CHAR(20)
4 );
5
6
7 •   SELECT * FROM GRAB3.LOCATIONS;

```

The result grid displays the following data:

ID_LOCATION	LOCATION
1	Ha Dong
2	Nam Tu Liem
3	Bac Tu Liem
4	Thanh Xuan
5	Hoan Kiem
6	Hai Ba Trung
7	Long Bien
8	Tay Ho
9	Cau Giay
10

Action Output details:

- Time: 15:32:56
- Action: SELECT * FROM GRAB3.LOCATIONS LIMIT 0, 1000
- Response: 12 row(s) returned
- Duration / Fetch Time: 0.0015 sec / 0.00001...

Query Completed.

- **ID_LOCATION**: ID of each location (Primary Key)
- **LOCATION**: name of the location

4.6. Table RESTAURANTS

The screenshot shows the MySQL Workbench interface with the schema **GRAB3** selected. In the central pane, the **RESTAURANTS** table is being created with the following SQL code:

```

1 • CREATE TABLE RESTAURANTS(
2     ID_RES CHAR(5) PRIMARY KEY,
3     NAME_RES CHAR(50),
4     DESCRIPTIONS CHAR(200),
5     OPEN_HOUR TIME,
6     CLOSE_HOUR TIME,
7     ID_LOCATION INT,
8     STAR FLOAT,
9
10    FOREIGN KEY (ID_LOCATION) REFERENCES LOCATIONS(ID_LOCATION) ON UPDATE RESTRICT ON DELETE CASCADE
11 );
12
13
14 •   SELECT * FROM GRAB3.RESTAURANTS;

```

The result grid displays the following data:

ID_RES	NAME_RES	DESCRIPTIONS	OPEN_HOUR	CLOSE_HOUR	ID_LOCATION	STAR
RES001	NGON	sells classic pizzas	08:00:01	22:00:00	1	3
RES002	Pizza AP	sells healthy pizzas	08:00:02	22:30:00	3	3.25
RES004	Pho Bat Dan	sells traditional Pho	08:00:03	22:30:00	1	2
RES005	Taco Non La	sells vietnamese style tacos	08:00:04	22:30:00	2	1.25
RES006	Savoror	sells Panini, Italian sandwich	08:00:05	22:30:00	2	2.6
RES007	Don Chicken	sells Korean style fried chicken	08:00:06	22:30:00	3	2
RES008	Jelly Beans	sells Tao Pho and snacks	08:00:07	22:30:00	1	3
RES009	Moo Beefsteak	sells beefsteaks	08:00:08	22:30:00	2	3
RES010	Labonte	sells coffee and brunch	08:00:09	22:30:00	4	2
RES011	RES011	RES011	RES011	RES011

Action Output details:

- Time: 15:38:33
- Action: SELECT * FROM GRAB3.RESTAURANTS LIMIT 0, 1000
- Response: 10 row(s) returned
- Duration / Fetch Time: 0.00050 sec / 0.0000...

Query Completed.

- **ID_RES**: ID of each restaurant (Primary Key)
- **DESCRIPTION**: describes overall about the restaurant

- OPEN_HOUR: the time when restaurant opens
- CLOSE_HOUR: the time when the restaurant closes
- ID_LOCATION: stores the restaurant's district number in order to calculate the shipping cost by the procedure GET_SHIP_COST (Foreign Key from LOCATIONS)
- STAR: rating of customer about the restaurant (by the number of star). Average star received from customers which will be updated by trigger ADD_STAR_RES

4.7. Table FOODS

The screenshot shows the MySQL Workbench interface with the 'test mysql' database selected. In the left sidebar, under the 'GRAB3' schema, the 'Tables' section lists various tables like AUDIT, CUSTOMERS, DRIVERS, FOODS, LOCATIONS, MESSAGES, ORDER_DETAILS, ORDERS, PAYMENTS, RATINGS, RESTAURANTS, STATUS, USERS, and VOUCHERS. The 'GRAB3.FOODS' table is currently selected.

In the main pane, the SQL editor contains the following code:

```

CREATE TABLE FOODS(
    ID_FOOD CHAR(5) PRIMARY KEY,
    ID_RES CHAR(5),
    FNAME CHAR(50),
    PRICE INT,
    STOCKS INT,
    FOREIGN KEY (ID_RES) REFERENCES RESTAURANTS(ID_RES) ON UPDATE RESTRICT ON DELETE CASCADE
);

SELECT * FROM GRAB3.FOODS;
  
```

The 'Result Grid' tab shows the data for the FOODS table:

ID_FOOD	ID_RES	FNAME	PRICE	STOCKS
BP003	RES002	baked potatoe	5000	3957
BS005	RES002	beefsteak	107000	4521
BL005	RES002	burger	78000	4396
CB009	RES009	creme brulee	42000	8624
CF10	RES10	coffee	136000	1041
EG004	RES04	egg	5000	9862
FC001	RES01	fried chicken	100000	4214
FC007	RES07	fried chicken	77000	955
FD004	RES04	fried dough	5000	2947
FD008	RES08	fried dough	125000	4910

At the bottom, the status bar indicates 'Query Completed'.

- ID_FOOD: ID of each food, each ID_FOOD belongs to only one restaurant (Primary Key)
- ID_RES: ID of restaurant (Foreign Key from RESTAURANTS)
- FNAME: the name of the food.
- PRICE: food's price per unit.
- STOCKS: the food's quantity left. After each order, food's stocks will be deducted by the amount in the customer's order. If the amount ordered is larger than the stocks, system will not let them order the food and pop out a message says "Only {stocks} {fname} left"

4.8. Table ORDERS

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the schema is set to 'test mysql'. Below it, the 'Schemas' tab is selected. On the left, the 'GRAB3' schema is expanded, showing various tables like AUDIT, CUSTOMERS, DRIVERS, etc. The central pane displays the SQL code for creating the 'ORDERS' table:

```

1 • CREATE TABLE ORDERS(
2     ID_ORDER CHAR(5) PRIMARY KEY,
3     ID_CUS CHAR(5),
4     ID_DRIVER CHAR(5),
5     ID_RES CHAR(10),
6     ID_STATUS INT,
7     ID_LOC_CUS INT,
8     NOTE CHAR(100),
9     ORDER_TIME DATETIME,
10    TAKEN_TIME TIME,
11
12    FOREIGN KEY (ID_RES) REFERENCES RESTAURANTS(ID_RES) ON UPDATE RESTRICT ON DELETE CASCADE,
13    FOREIGN KEY (ID_CUS) REFERENCES CUSTOMERS(ID_CUS) ON UPDATE RESTRICT ON DELETE CASCADE,
14    FOREIGN KEY (ID_DRIVER) REFERENCES DRIVERS(ID_DRIVER) ON UPDATE RESTRICT ON DELETE CASCADE,
15    FOREIGN KEY (ID_STATUS) REFERENCES STATUSUSS(ID_STATUS) ON UPDATE RESTRICT ON DELETE CASCADE,
16    FOREIGN KEY (ID_LOC_CUS) REFERENCES LOCATIONS(ID_LOCATION) ON UPDATE RESTRICT ON DELETE CASCADE
17
18 );
19
20
21 • SELECT * FROM GRAB3.ORDERS;

```

Below the code, the 'Result Grid' shows the data for the 'ORDERS' table:

ID_ORDER	ID_CUS	ID_DRIVER	ID_RES	ID_STATUS	ID_LOC_CUS	NOTE	ORDER_TIME	TAKEN_TIME
OD01	CUS05	DVR01	RES01	1	11		2021-01-16 20:27:06	
OD02	CUS06	DVR08	RES02	1	12		2021-01-16 15:05:53	
OD03	CUS01	DVR15	RES10	1	5		2021-02-14 20:23:11	
OD04	PVN01	PVN01	RES04	1	4		2021-01-16 20:27:06	

The bottom status bar indicates 'Query Completed'.

This table store data of *one* order:

- ID_ORDER: ID of each order (Primary Key)
- ID_CUS: ID of the customer who made this order (Foreign Key from CUSTOMERS)
- ID_DRIVER: ID of the driver who delivery this order (Foreign Key from DRIVERS)
- ID_RES: ID of restaurant that provided the food for this order (Foreign Key from RESTAURANTS)
- ID_STATUSUSS: the status of the order which would tell the customers the condition of their orders (Foreign Key from STATUSUSS). It is also used to determine whether an order can be canceled or not. Based on that, we could decide the amount of money must be returned to the customer. Each order would have 4 stages which will be described in detail later in the table STATUSUSS. Besides, ID_STATUSUSS will be updated through a procedure called UPDATE_STATUS.
- An order might be able to cancel if ID_STATUS is 1 (ORDERED) and the ID_STATUSUSS will be set to 5 (CANCELED), other ID_STATUSUSS couldn't

be withdrawn. The UPDATE_STATUS will check it for the customers. Here is what a customer must do if he wants to cancel an order. Let's take OD02 as an example.

Session 1 (Top):

```

1 •  SELECT * FROM GRAB3.ORDERS;
2
3 •  CALL UPDATE_STATUS(5, '00082');

```

Result Grid:

ID_ORDER	ID_CUS	ID_DRIVER	ID_RES	ID_STATUS	ID_L...	NOTE	ORDER_TIME	TAKEN_TIME
OD02	CUS05	DVR08	RES02	1	12	NULL	2021-01-16 15:05:53	NULL
OD04	CUS04	DVR11	RES09	1	1	NULL	2021-02-19 01:31:44	NULL
OD05	CUS07	DVR02	RES02	1	5	NULL	2021-04-08 15:14:44	NULL
OD06	CUS10	DVR04	RES09	1	7	NULL	2021-05-13 07:45:10	NULL
OD07	CUS13	DVR09	RES03	1	12	NULL	2021-02-02 15:17:52	NULL
OD08	CUS16	DVR03	RES07	1	9	NULL	2021-02-18 17:29:39	NULL
OD09	CUS19	DVR12	RES01	1	8	NULL	2021-02-04 08:12:50	NULL
OD10	CUS20	DVR05	RES01	1	5	NULL	2021-03-20 01:50:20	NULL
OD11	CUS25	DVR01	RES03	1	7	NULL	2021-03-23 14:11:45	NULL
OD12	CUS28	DVR10	RES06	1	10	NULL	2021-03-20 01:58:49	NULL
OD13	CUS31	DVR06	RES04	1	12	NULL	2021-05-14 05:00:03	NULL
OD14	CUS34	DVR13	RES02	1	2	NULL	2021-02-15 08:59:00	NULL
OD15	CUS33	DVR14	RES07	1	12	NULL	2021-02-12 00:13:20	NULL
OD16	CUS32	DVR01	RES10	1	6	NULL	2021-03-11 02:36:17	NULL
OD17	CUS01	DVR08	RES05	1	5	NULL	2021-04-06 18:47:22	NULL
OD18	CUS20	DVR04	RES04	1	12	NULL	2021-04-28 22:47:18	NULL
OD19	CUS14	DVR01	RES01	1	3	NULL	2021-02-04 08:50:20	NULL
OD20	CUS33	DVR15	RES04	1	11	NULL	2021-03-23 07:00:02	NULL
OD21	CUS15	DVR11	RES08	1	11	NULL	2021-02-06 23:04:35	NULL
OD22	CUS03	DVR02	RES10	1	8	NULL	2021-05-11 14:09:55	NULL
OD23	none	none	RES09	1	19	NULL	2021-05-20 09:59:24	NULL

Session 2 (Bottom):

```

1 •  SELECT * FROM GRAB3.ORDERS;
2
3 •  CALL UPDATE_STATUS(5, '00082');

```

Result Grid:

ID_ORDER	ID_CUS	ID_DRIVER	ID_RES	ID_STATUS	ID_L...	NOTE	ORDER_TIME	TAKEN_TIME
OD02	CUS05	DVR08	RES02	5	12	NULL	2021-01-16 15:05:53	NULL
OD03	CUS07	DVR13	RES10	1	3	NULL	2021-02-15 02:20:11	NULL
OD04	CUS04	DVR11	RES09	1	1	NULL	2021-02-19 01:31:44	NULL
OD05	CUS07	DVR02	RES02	1	5	NULL	2021-04-08 15:14:44	NULL
OD06	CUS10	DVR04	RES09	1	7	NULL	2021-05-13 07:45:10	NULL
OD07	CUS13	DVR09	RES03	1	12	NULL	2021-02-02 15:17:52	NULL
OD08	CUS16	DVR03	RES07	1	9	NULL	2021-02-18 17:29:39	NULL
OD09	CUS19	DVR12	RES01	1	8	NULL	2021-02-04 08:12:50	NULL
OD10	CUS22	DVR05	RES04	1	5	NULL	2021-04-01 07:03:48	NULL
OD11	CUS25	DVR07	RES03	1	7	NULL	2021-02-22 14:11:45	NULL
OD12	CUS28	DVR10	RES06	1	10	NULL	2021-03-20 01:58:49	NULL
OD13	CUS31	DVR06	RES04	1	12	NULL	2021-05-14 05:00:03	NULL
OD14	CUS34	DVR13	RES02	1	2	NULL	2021-02-15 08:59:00	NULL
OD15	CUS33	DVR14	RES07	1	12	NULL	2021-02-12 00:13:20	NULL
OD16	CUS32	DVR01	RES10	1	6	NULL	2021-03-11 02:36:17	NULL
OD17	CUS01	DVR08	RES05	1	5	NULL	2021-04-06 18:47:22	NULL
OD18	CUS20	DVR04	RES04	1	12	NULL	2021-04-28 22:47:18	NULL
OD19	CUS14	DVR01	RES01	1	3	NULL	2021-02-04 08:50:20	NULL
OD20	CUS33	DVR15	RES04	1	11	NULL	2021-03-23 07:00:02	NULL
OD21	CUS15	DVR11	RES08	1	11	NULL	2021-02-06 23:04:35	NULL
OD22	CUS03	DVR02	RES10	1	8	NULL	2021-05-11 14:09:55	NULL
OD23	none	none	RES09	1	19	NULL	2021-05-20 09:59:24	NULL

When the ID_STATUS changes into 5, his order is successfully canceled.

- Let's take another example, the “OD04” order has ID_STATUS = 2.

Session 1 (Top):

```

1 • SELECT * FROM GRAB3.ORDERS;
2
3 • CALL UPDATE_STATUS(5, '0004');

```

Session 2 (Bottom):

```

1 • SELECT * FROM GRAB3.ORDERS;
2
3 • CALL UPDATE_STATUS(5, '0004');

```

ORDERS Table Data:

ID_ORDER	ID_CUS	ID_DRIVER	ID_RES	ID_STATUS	ID_LO...	NOTE	ORDER_TIME	TAKEN_TIME
OD01	CUS05	DVR01	RES01	1	11		2021-01-16 20:27:06	2021-01-16 20:27:06
OD02	CUS05	DVR08	RES02	5	12		2021-01-16 15:05:53	2021-01-16 15:05:53
OD04	CUS04	DVR11	RES09	2	1		2021-02-19 01:31:44	2021-02-19 01:31:44
OD06	CUS10	DVR04	RES08	1	7		2021-05-13 07:45:10	2021-05-13 07:45:10
OD07	CUS13	DVR09	RES03	1	12		2021-02-02 15:17:52	2021-02-02 15:17:52
OD08	CUS16	DVR03	RES07	1	9		2021-02-18 17:29:39	2021-02-18 17:29:39
OD09	CUS19	DVR12	RES01	1	8		2021-02-04 08:12:50	2021-02-04 08:12:50
OD10	CUS22	DVR05	RES04	1	5		2021-04-01 07:03:48	2021-04-01 07:03:48
OD11	CUS25	DVR07	RES03	1	7		2021-02-22 14:11:45	2021-02-22 14:11:45
OD12	CUS28	DVR10	RES06	1	10		2021-03-20 01:58:49	2021-03-20 01:58:49
OD13	CUS31	DVR06	RES04	1	12		2021-05-14 05:00:03	2021-05-14 05:00:03
OD14	CUS34	DVR13	RES02	1	2		2021-02-15 08:59:00	2021-02-15 08:59:00
OD15	CUS35	DVR14	RES01	1	12		2021-03-12 00:13:20	2021-03-12 00:13:20
OD16	CUS32	DVR09	RES10	1	6		2021-05-11 05:26:17	2021-05-11 05:26:17
OD17	CUS01	DVR08	RES06	1	5		2021-04-06 18:47:22	2021-04-06 18:47:22
OD18	CUS20	DVR04	RES04	1	12		2021-04-28 22:47:16	2021-04-28 22:47:16
OD19	CUS14	DVR01	RES01	1	3		2021-03-23 19:50:30	2021-03-23 19:50:30
OD20	CUS33	DVR15	RES04	1	11		2021-02-11 07:00:02	2021-02-11 07:00:02
OD21	CUS15	DVR11	RES08	1	11		2021-02-06 23:04:35	2021-02-06 23:04:35
OD22	CUS03	DVR02	RES10	1	8		2021-05-11 14:09:55	2021-05-11 14:09:55
OD23	CUS06	hubone	RES09	1	19		2021-05-05 07:22:54	2021-05-05 07:22:54

Action Output:

Time	Action
15 16:09:58	SELECT * FROM GRAB3.ORDERS LIMIT 0, 1000

Response: 52 row(s) returned
Duration / Fetch Time: 0.00096 sec / 0.000...

Query Completed

Action Output:

Time	Action
15 16:09:58	SELECT * FROM GRAB3.ORDERS LIMIT 0, 1000

Response: 52 row(s) returned
Duration / Fetch Time: 0.00096 sec / 0.000...

Query Completed

After running the UPDATE_STATUS procedure, ID_STATUS of OD04 still has not changed.

```

1 •  SELECT * FROM GRAB3.ORDERS;
2
3 •  CALL UPDATE_STATUS(5, '0004');
4
5 •  CALL POP_MESSAGE();

```

MESSAGE

> You cannot cancel this order!

Action Output

Time	Action
483 16:13:33	CALL POP_MESSAGE()

Response

1row(s) returned

Duration / Fetch Time

0.00081 sec / 0.0000...

Query Completed

- ID_LOCATION is the address that the order is delivered to. If the customers want to order some foods for someone, they just need to insert in a location. Else, system will automatically set the customers' address as default. (Foreign key from LOCATIONS)
- ORDER_TIME: represents the time when the driver receives the order
- TAKEN_TIME: represents the time when the customer receives the order.
- TAKEN_TIME will be updated whenever ID_STATUS is 4.

```

1 •  SELECT * FROM GRAB3.ORDERS;
2
3 •  CALL UPDATE_STATUS(4, '00B4');
4

```

ID_ORDER	ID_CUS	ID_DRIVER	ID_RES	ID_STATUS	ID_L...	NOTE	ORDER_TIME	TAKEN_TIME
0001	CUS05	DVR01	RES01	1	11		2021-01-16 20:27:06	
0002	CUS05	DVR08	RES02	1	12		2021-01-16 15:05:53	
0004	CUS04	DVR11	RES09	4	1		2021-02-19 01:31:44	2021-06-07 16:23:35
0006	CUS10	DVR04	RES08	1	7		2021-05-13 07:45:10	
0007	CUS13	DVR03	RES01	1	12		2021-05-18 17:29:39	
0008	CUS14	DVR03	RES07	1	9		2021-05-18 17:29:39	
0009	CUS19	DVR12	RES01	1	8		2021-02-04 08:12:50	
0010	CUS22	DVR05	RES04	1	5		2021-04-01 07:03:48	
0011	CUS22	DVR07	RES03	1	7		2021-02-22 14:11:45	
0012	CUS28	DVR10	RES06	1	10		2021-03-20 01:58:49	
0013	CUS31	DVR06	RES04	1	12		2021-05-14 05:00:03	
0014	CUS34	DVR13	RES02	1	2		2021-02-15 08:59:00	
0015	CUS35	DVR14	RES07	1	12		2021-02-12 00:13:20	
0016	CUS35	DVR14	RES10	1	6		2021-04-06 16:47:22	
0017	CUS51	DVR08	RES05	1	5		2021-04-06 16:47:22	
0018	CUS80	DVR04	RES04	1	12		2021-04-28 22:47:16	
0019	CUS14	DVR01	RES01	1	3		2021-03-23 19:50:30	
0020	CUS33	DVR15	RES04	1	11		2021-02-11 07:00:02	
0021	CUS15	DVR11	RES08	1	11		2021-02-08 23:04:35	
0022	CUS03	DVR02	RES10	1	8		2021-05-11 14:09:55	
0023	CUS04	DVR04	RES09	1	12		2021-05-20 09:22:35	

Object Info Session
No object selected

Action Output Response Duration / Fetch Time
Time Action 1414 16:23:39 SELECT * FROM GRAB3.ORDERS LIMIT 0, 1000 52 row(s) returned 0.0018 sec / 0.00004...

Query Completed

4.9. Table STATUSS

```

1 •  CREATE TABLE STATUSS(
2     ID_STATUS INT PRIMARY KEY,
3     STATUS CHAR(50)
4 );
5
6
7 •  SELECT * FROM GRAB3.STATUSS;

```

ID_STATUS	STATUS
1	ORDERING
2	PREPARING
3	DELIVERING
4	TAKEN
5	CANCELED

Object Info Session
No object selected

Action Output Response Duration / Fetch Time
Time Action 1415 16:32:44 SELECT * FROM GRAB3.STATUSS LIMIT 0, 1000 5 row(s) returned 0.0012 sec / 0.00001...

Query Completed

- ID_STATUS: ID of each status (Primary Key)
- STATUS: store the status of each order

4.10. Table ORDER_DETAILS

The screenshot shows the MySQL Workbench interface with the 'test mysql' database selected. The 'Schemas' tree on the left shows the 'GRAB3' schema containing tables like AUDIT, CUSTOMERS, DRIVERS, FOODS, LOCATIONS, MESSAGES, ORDER_DETAILS, ORDERS, PAYMENTS, RATINGS, RESTAURANTS, STATUS, USERS, and VOUCHERS. The 'Tables' node is expanded. The central pane displays the SQL code for creating the ORDER_DETAILS table:

```

1  CREATE TABLE ORDER_DETAILS(
2      ID_OD INT AUTO_INCREMENT PRIMARY KEY,
3      ID_ORDER CHAR(5),
4      ID_FOOD CHAR(5),
5      QUANTITY INT,
6      PRICE INT,
7
8      FOREIGN KEY (ID_ORDER) REFERENCES ORDERS(ID_ORDER) ON UPDATE RESTRICT ON DELETE CASCADE,
9      FOREIGN KEY (ID_FOOD) REFERENCES FOODS(ID_FOOD) ON UPDATE RESTRICT ON DELETE CASCADE
10  );
11
12
13 *  SELECT * FROM GRAB3.ORDER_DETAILS;

```

The 'Result Grid' pane below shows the data for the ORDER_DETAILS table:

ID_OD	ID_ORDER	ID_FOOD	QUANTITY	PRICE
1	OD001	FO01	5	500000
2	OD001	FF01	2	240000
3	OD002	BG02	1	940000
4	OD003	CF10	2	272000
5	OD003	SW10	2	252000
6	OD004	SO09	4	688000
7	OD004	BS09	4	428000
8	OD004	SL09	2	226000
9	OD005	BG02	6	564000
10	OD005	FF02	8	1488000
11	OD006	FD06	5	635000
12	OD008	TP08	3	231000
13	OD007	PZ03	1	101000
14	OD008	FC07	3	231000
...

The bottom status bar indicates the query was completed at 14:16 on 16/3/04, selecting all rows from the ORDER_DETAILS table with a limit of 0, 1000, and returning 82 rows in 0.00054 sec / 0.000... ms.

This table stores every food that a customer ordered orderly:

- ID_OD: ID of each detail order line (Primary Key)
- ID_ORDER: ID of the order (Foreign Key from ORDERS)
- ID_FOOD: ID of the food (Foreign Key from FOODS)
- QUANTITY: quantity of each food
- PRICE: price of the food per unit times quantity

4.11. Table VOUCHERS

The screenshot shows the MySQL Workbench interface with the 'VOUCHERS' table selected. The left sidebar shows the 'GRAB3' schema with various tables like AUDIT, CUSTOMERS, DRIVERS, FOODS, LOCATIONS, MESSAGES, ORDER_DETAILS, ORDERS, PAYMENTS, RATINGS, RESTAURANTS, STATUS, USERS, and VOUCHERS. The main area displays the table structure:

```

CREATE TABLE VOUCHERS (
    ID_VOUCHER CHAR(20) PRIMARY KEY,
    DISCOUNT FLOAT,
    MAX_USE INT
);

```

Below this, a query is run:

```

SELECT * FROM GRAB3.VOUCHERS;

```

The result grid shows the following data:

ID_VOUCHER	DISCOUNT	MAX_USE
AIRPI10	0.1	992
BANK10	0.1	988
GRAB20	0.2	200
GRAB50	0.5	100
MOMO10	0.1	989
OOV	0	0
QUOCKHANH	0.59	100
SINH-NHAT	0.7	50
TET	0.7	100
ZALO10	0.1	990
MAX	MAX	MAX

At the bottom, the query history shows:

```

14... 16:54:29 SELECT * FROM GRAB3.VOUCHERS LIMIT 0, 1000

```

And the status bar indicates:

Response: 10 row(s) returned Duration / Fetch Time: 0.0011 sec / 0.00003...

- **ID_VOUCHER:** ID of each voucher (Primary Key)
- **DISCOUNT:** a deduction percentage of the price towards each order
- **MAX_USE:** maximum time of the voucher be used. It will automatically be deducted after being inserted in PAYMENTS.

4.12. Table PAYMENTS

The screenshot shows the MySQL Workbench interface with the 'test mysql' database selected. In the top navigation bar, 'Administration' is active, followed by 'Schemas'. The 'GRAB3' schema is selected. In the left sidebar, under 'Tables', the 'PAYMENTS' table is highlighted. The main pane displays the table's structure:

```

CREATE TABLE PAYMENTS (
    ID_PAYM INT AUTO_INCREMENT PRIMARY KEY,
    ID_ORDER CHAR(5),
    RES_COST INT,
    SHIP_COST INT,
    PAYMENT_METHOD ENUM('MOMO', 'AIRPAY', 'ZALO', 'BANKING', 'COD'),
    IS_PAID ENUM('0', '1'),
    VOUCHER_PM CHAR(20),
    ID_VOUCHER CHAR(20) DEFAULT NULL,
    TOTAL_PRICE INT,
    FOREIGN KEY (ID_ORDER) REFERENCES ORDERS(ID_ORDER) ON UPDATE RESTRICT ON DELETE CASCADE,
    FOREIGN KEY (ID_VOUCHER) REFERENCES VOUCHERS(ID_VOUCHER) ON UPDATE RESTRICT ON DELETE CASCADE
);

```

Below the structure, a query is run:

```

SELECT * FROM GRAB3.PAYMENTS;

```

The results grid shows the following data:

ID_PAYM	ID_ORDER	RES_COST	SHIP_COST	PAYMENT_METHOD	IS_PAID	VOUCHER_PM	ID_VOUCHER	TOTAL_PRICE
42	OD42	204000	43836	COD	0	ZALO10	ZALO	247836
43	OD43	456000	77915	ZALO	1	ZALO10	ZALO	533915
44	OD44	780000	57385	AIRPAY	1	AIRP10	ZALO	837385
45	OD45	612000	51094	ZALO	1	ZALO10	ZALO	663094
46	OD46	790000	77915	MOMO	1	MOMO10	ZALO	839915
47	OD47	1860000	57285	BANKING	1	BANK10	ZALO	1837385
48	OD48	779000	85020	COD	0	ZALO	ZALO	864020
49	OD49	153000	33732	MOMO	1	MOMO10	ZALO	186732
50	OD50	312000	101970	AIRPAY	1	AIRP10	ZALO	413970
52	OD52	1560000	101970	MOMO	1	MOMO10	OOV	1661970

At the bottom, the status bar shows 'Query Completed'.

- **ID_PAYM:** ID of each payment transaction (Primary Key)
- **ID_ORDER:** ID of each order (Foreign Key from ORDERS)
- **RES_COST:** the cost for the food ordered by customer calculated by multiply food price with quantity in each order
- **SHIP_COST:** the cost for the delivery from the restaurant to the appointed location. It will be calculated by the GET_SHIP_COST procedure.
- **PAYMENT_METHOD:** customers can choose which payment method they would like to use. Except for COD, other payment method comes with a voucher that will automatically added by ADD_VOUCHER trigger. It will directly reduce the total price.
- **IS_PAID:** is added automatically based on the payment method above. It also let the driver knows whether the order has been paid or not.
- **VOUCHER_PM:** contains vouchers which come with the payment methods.
- **ID_VOUCHER:** ID of voucher that might be released by GrabFood (Foreign Key from VOUCHERS)
- If the MAX_USE of any voucher is 0 (Example: ID_PAYM = 52, MAX_USE of OOV = 0)

```

test mysql

Administration Schemas GRAB_FINAL GRAB_TRIGGER_FINAL GRAB_PROCEDURE_FINAL GRAB_DATABASE_FINAL GRAB_VIEWS_FINAL PAYMENTS
Limit to 1000 rows
Schemas
Filter objects
GRAB3
Tables
AUDIT
CUSTOMERS
DRIVERS
FOODS
LOCATIONS
MESSAGES
ORDER_DETAILS
ORDERS
PAYMENTS
RATINGS
RESTAURANTS
STATUS
USERS
VOUCHERS
Views
Stored Procedures
Functions
QLBH
Object Info Session
No object selected

```

```

5 SHIP_COST INT,
6 PAYMENT_METHOD ENUM('MOMO', 'AIRPAY', 'ZALO', 'BANKING', 'COD'),
7 TS_PAID ENUM('0', '1'),
8 VOUCHER_PM CHAR(20),
9 ID_VOUCHER CHAR(20) DEFAULT NULL,
10 TOTAL_PRICE INT,
11
12 FOREIGN KEY (ID_ORDER) REFERENCES ORDERS(ID_ORDER) ON UPDATE RESTRICT ON DELETE CASCADE,
13 FOREIGN KEY (ID_VOUCHER) REFERENCES VOUCHERS(ID_VOUCHER) ON UPDATE RESTRICT ON DELETE CASCADE
14 ;
15
16 • INSERT INTO PAYMENTS(ID_ORDER, PAYMENT_METHOD, ID_VOUCHER) VALUES ('0052', 'MOMO', '00V');
17
18 • SELECT * FROM GRAB3.PAYMENTS;
19
20 • CALL POP_MESSAGE();
21

```

Result Grid Filter Rows: Search Export:

MESSAGE
> OOV OUT OF VOUCHER

Result 4

Action Output Response Duration / Fetch Time

Time	Action	Response	Duration / Fetch Time
1423 16:51:31	CALL POP_MESSAGE()	1 row(s) returned	0.00085 sec / 0.000...

Query Completed

- **TOTAL_PRICE**: total price customers must pay after discounting which will be added by **TOTAL_PRICE** Trigger.

4.13. Table RATINGS

```

test mysql

Administration Schemas GRAB_FINAL GRAB_TRIGGER_FINAL GRAB_PROCEDURE_FINAL GRAB_DATABASE_FINAL GRAB_VIEWS_FINAL RATINGS
Limit to 1000 rows
Schemas
Filter objects
GRAB3
Tables
AUDIT
CUSTOMERS
DRIVERS
FOODS
LOCATIONS
MESSAGES
ORDER_DETAILS
ORDERS
PAYMENTS
RATINGS
RESTAURANTS
STATUS
USERS
VOUCHERS
Views
Stored Procedures
Functions
QLBH
QLBH69
sys
Object Info Session
No object selected

```

```

1 • CREATE TABLE RATINGS(
2   ID_RATING CHAR(5) PRIMARY KEY,
3   ID_ORDER CHAR(5),
4   ID_RES CHAR(10),
5   STAR_RES INT,
6   ID_DRIVER CHAR(5),
7   STAR_DRIVER INT,
8
9   FOREIGN KEY (ID_RES) REFERENCES RESTAURANTS(ID_RES) ON UPDATE RESTRICT ON DELETE CASCADE,
10  FOREIGN KEY (ID_ORDER) REFERENCES ORDERS(ID_ORDER) ON UPDATE RESTRICT ON DELETE CASCADE,
11  FOREIGN KEY (ID_DRIVER) REFERENCES DRIVERS(ID_DRIVER) ON UPDATE RESTRICT ON DELETE CASCADE
12
13
14 • SELECT * FROM GRAB3.RATINGS;

```

Result Grid Filter Rows: Search Export/Import:

ID_RATING	ID_ORDER	ID_RES	STAR_RES	ID_DRIVER	STAR_DRIVER
RAT01	OD001	RES001	1	DVR01	1
RAT02	OD002	RES002	3	DVR08	4
RAT03	OD003	RES003	3	DVR15	5
RAT04	OD004	RES009	1	DVR11	1
RAT05	OD005	RES002	1	DVR02	2
RAT06	OD006	RES008	3	DVR04	4
RAT07	OD007	RES003	4	DVR09	4
RAT08	OD008	RES009	3	DVR02	4
RAT09	OD009	RES001	1	DVR12	3
RAT10	OD010	RES004	1	DVR05	3
RAT11	OD011	RES003	3	DVR07	2
RAT12	OD012	RES002	1	DVR10	1
RAT13	OD013	RES004	4	DVR06	2
RAT14	OD014	RES002	3	DVR13	1
RAT15	OD015	RES007	1	DVR14	3
RAT16	OD016	RES009	4	DVR04	4
RAT17	OD017	RES005	2	nvne0k	3

RATINGS 1

Action Output Response Duration / Fetch Time

Time	Action	Response	Duration / Fetch Time
1427 17:01:05	SELECT * FROM GRAB3.RATINGS LIMIT 0, 1000	52 row(s) returned	0.00087 sec / 0.000...

Query Completed

- **ID_RATING**: ID of each rating (Primary Key)
- **ID_ORDER**: ID of each order (Foreign Key from ORDERS)

- ID_RES: ID of restaurant will automatically be updated by trigger UPDATE_DRIVER_RES (Foreign Key from RESTAURANTS)
- STAR_RES: stores the number of stars rated by customers for the restaurant
- ID_DRIVER: ID of each driver (Foreign Key from DRIVERS)
- STAR_DRIVER: stores the number of stars rated by customers for the driver.

5. QUERY

In order to explore insights from the database, we implement some demo queries. The script and results are in GRAB_VIEWS_FINAL.sql file.

5.1. SELECT

The 10 queries below provide some basic information about our database. Results after implemented could be used by a variety of audiences such as restaurants or market researcher company.

- Top 5 restaurants have the highest rating. Since customers tend to order from restaurants with high ratings, GrabFood can organize a campaign in order to promote restaurants that are highly rated and help the others with poor ratings.

```

test mysql
GRAB_FINAL GRAB_TRIGGER_FINAL GRAB_PROCEDURE_FINAL GRAB_DATABASE_FINAL GRAB_VIEWS_FINAL ORDER_DETAILS
Administration Schemas
Schemas
Filter objects
GRAB
Tables
AUDIT
CUSTOMERS
DRIVERS
FOODS
LOCATIONS
MESSAGES
ORDER_DETAILS
ORDERS
PAYMENTS
RATINGS
RESTAURANTS
STATUS
USERS
VOUCHERS
Views
Stored Procedures
Functions
QLBH
QLBH69
sys

Object Info Session
No object selected

1 -- 1 TOP 5 RES HAVE HIGHEST RATES
2 • SELECT RESTAURANTS.ID_RES, RESTAURANTS.STAR AS STAR FROM RESTAURANTS
   ORDER BY STAR DESC
4 LIMIT 5;
5
6
Result Grid Filter Rows: Search Edit: Export/Import: Fetch rows: 
ID_RES STAR
RES001 3.25
RES002 3
RES003 3
RES004 3
RES006 2.6
7
8
9
10
11
12
13
14... 18:00:22 SELECT RESTAURANTS.ID_RES, RESTAURANTS.STAR AS STAR FROM RESTAURANTS ORDER BY STAR DESC LIMIT 5
5 row(s) returned
0.00049 sec / 0.000...
Query Completed

```

- Restaurants still open at night. Some of GrabFood's customers mostly work at night so creating a list of restaurants that still open late will help them ordering some foods.

```

test mysql
GRAB_FINAL GRAB_TRIGGER_FINAL GRAB_PROCEDURE_FINAL GRAB_DATABASE_FINAL GRAB_VIEWS_FINAL ORDER_DETAILS
Administration Schemas
Schemas
Filter objects
GRAB
Tables
AUDIT
CUSTOMERS
DRIVERS
FOODS
LOCATIONS
MESSAGES
ORDER_DETAILS
ORDERS
PAYMENTS
RATINGS
RESTAURANTS
STATUS
USERS
VOUCHERS
Views
Stored Procedures
Functions
QLBH
QLBH69
sys

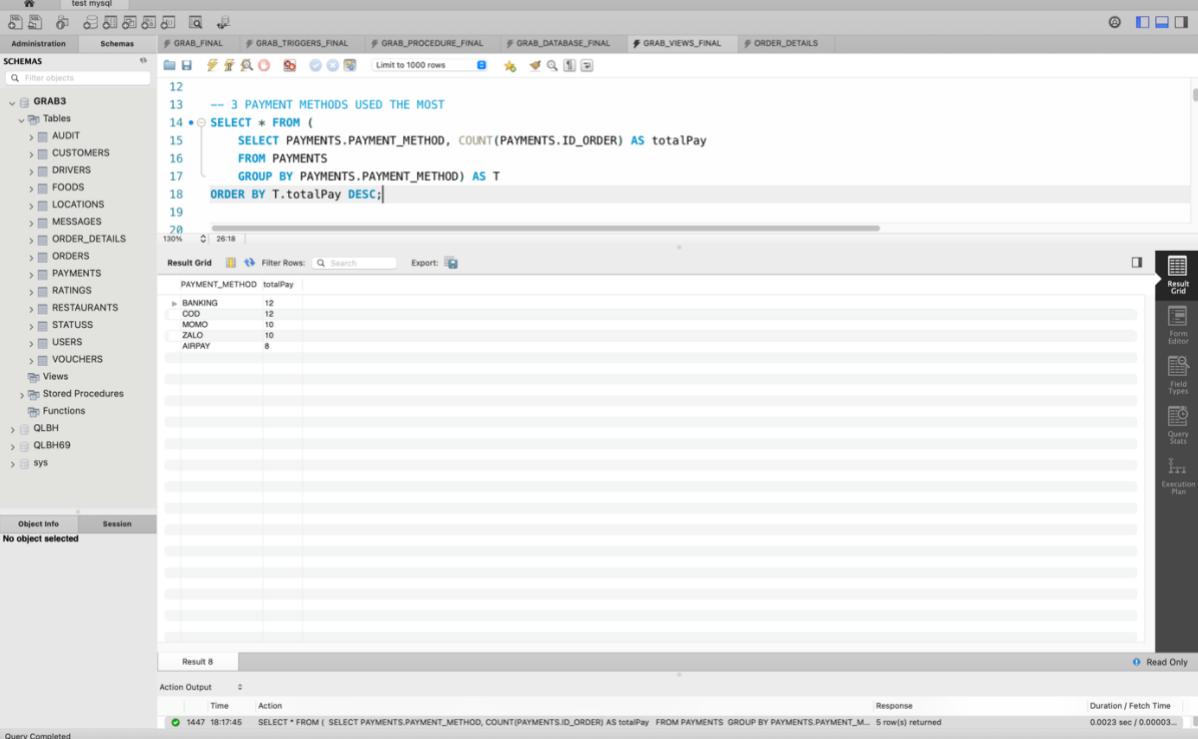
Object Info Session
No object selected

5
6
7 -- 2 FIND 'CAC QUAN AN DEM'
8 • SELECT * FROM RESTAURANTS
9 WHERE CLOSE_HOUR > '22:00:00' OR CLOSE_HOUR < '04:00:00'
10 OR OPEN_HOUR > '22:00:00' OR OPEN_HOUR < '04:00:00';
11
12
13
14... 18:13:10 SELECT * FROM RESTAURANTS WHERE CLOSE_HOUR > '22:00:00' OR CLOSE_HOUR < '04:00:00' OR OPEN_HOUR > '22:00:00' OR OPEN_HOUR < '04:00:00'...
6 row(s) returned
0.0018 sec / 0.000...
Query Completed

```

- Payment methods are used mostly. This will help GrabFood visualizing the growing trend in payment methods (PM) so that they can associate with

the PM's owner to release more vouchers, marketing campaigns in order to attract more customers.



The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```

12
13 -- 3 PAYMENT METHODS USED THE MOST
14 • SELECT * FROM (
15   SELECT PAYMENTS.PAYMENT_METHOD, COUNT(PAYMENTS.ID_ORDER) AS totalPay
16   FROM PAYMENTS
17   GROUP BY PAYMENTS.PAYMENT_METHOD) AS T
18 ORDER BY T.totalPay DESC;
19
20
Result Grid | Filter Rows: Search Export: 
PAYOUTMENT_METHOD totalPay
> BANKING 12
> COD 12
> MOMO 10
> ZALO 10
> AIRPAY 8

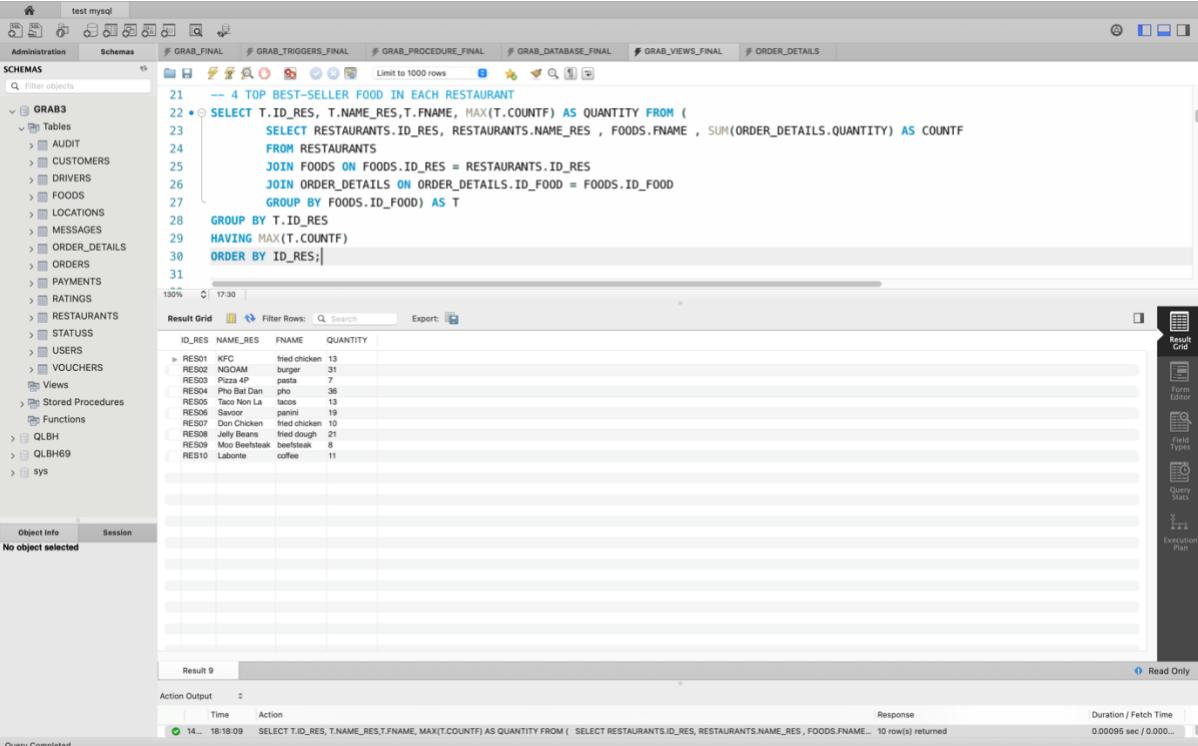
```

Execution Plan:

- 1447 18:17:45 SELECT * FROM (SELECT PAYMENTS.PAYMENT_METHOD, COUNT(PAYMENTS.ID_ORDER) AS totalPay FROM PAYMENTS GROUP BY PAYMENTS.PAYMENT_M...

Query Completed

- Top best-seller food in each restaurant. New customers tend to order the best-seller in the restaurant that they are about to order. So this table is going to give them some recommendations.



The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```

21 -- 4 TOP BEST-SELLER FOOD IN EACH RESTAURANT
22 • SELECT T.ID_RES, T.NAME_RES, T.FNAME, MAX(T.COUNTF) AS QUANTITY FROM (
23   SELECT RESTAURANTS.ID_RES, RESTAURANTS.NAME_RES, FOODS.FNAME, SUM(ORDER_DETAILS.QUANTITY) AS COUNTF
24   FROM RESTAURANTS
25   JOIN FOODS ON FOODS.ID_RES = RESTAURANTS.ID_RES
26   JOIN ORDER_DETAILS ON ORDER_DETAILS.ID_FOOD = FOODS.ID_FOOD
27   GROUP BY FOODS.ID_FOOD) AS T
28 GROUP BY T.ID_RES
29 HAVING MAX(T.COUNTF)
30 ORDER BY ID_RES;
31
Result Grid | Filter Rows: Search Export: 
ID_RES NAME_RES FNAME QUANTITY
> RES01 KFC fried chicken 13
RES02 NOGAM burger 31
RES03 Pizza 4P pasta 7
RES04 Pho Bat Dan pho 36
RES05 Non La tacobell 13
RES06 Savory pani 19
RES07 Don Chicken fried chicken 10
RES08 Jelly Beans fried dough 21
RES09 Moo Beefsteak beefsteak 8
RES10 Laborne coffee 11

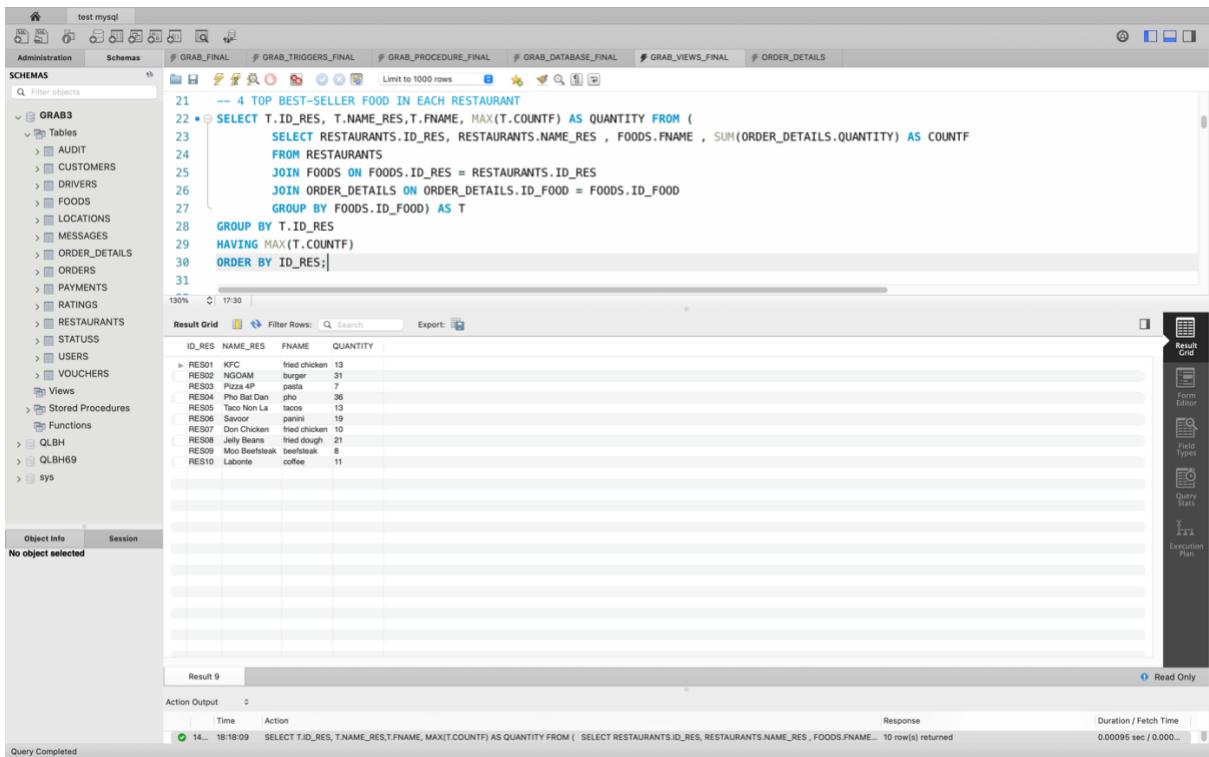
```

Execution Plan:

- 14... 18:18:09 SELECT T.ID_RES, T.NAME_RES, T.FNAME, MAX(T.COUNTF) AS QUANTITY FROM (SELECT RESTAURANTS.ID_RES, RESTAURANTS.NAME_RES, FOODS.FNAME...

Query Completed

- Top foods ordered by customers under 25 years old. This will not only help GrabFood visualize the trend in a specific market share but also help the restaurants to know which food that increase their revenue so that they can either improve the dish's quality or increase the stocks.



The screenshot shows the MySQL Workbench interface with a query editor and results grid. The query is:

```

21 -- 4 TOP BEST-SELLER FOOD IN EACH RESTAURANT
22 • SELECT T.ID_RES, T.NAME_RES, T.FNAME, MAX(T.COUNTF) AS QUANTITY FROM (
23     SELECT RESTAURANTS.ID_RES, RESTAURANTS.NAME_RES, FOODS.FNAME, SUM(ORDER_DETAILS.QUANTITY) AS COUNTF
24     FROM RESTAURANTS
25     JOIN FOODS ON FOODS.ID_RES = RESTAURANTS.ID_RES
26     JOIN ORDER_DETAILS ON ORDER_DETAILS.ID_FOOD = FOODS.ID_FOOD
27     GROUP BY FOODS.ID_FOOD) AS T
28 GROUP BY T.ID_RES
29 HAVING MAX(T.COUNTF)
30 ORDER BY ID_RES;
31

```

The results grid shows the top 4 best-selling foods per restaurant:

ID_RES	NAME_RES	FNAME	QUANTITY
RES01	KFC	fried chicken	13
RES02	NGOAM	burger	31
RES03	Pho 4P	pasta	1
RES04	Pho Bat Dan	pho	36
RES05	Taco Non La	taco	13
RES06	Savory	pasta	19
RES07	Spicy Chicken	fried chicken	8
RES08	Jelly Beans	fried dough	21
RES09	Moo Beefsteak	beefsteak	8
RES10	Labonte	coffee	11

Below the results, the execution plan is shown:

Action Output	Time	Action	Response	Duration / Fetch Time
14... 18:18:09		SELECT T.ID_RES, T.NAME_RES, T.FNAME, MAX(T.COUNTF) AS QUANTITY FROM (SELECT RESTAURANTS.ID_RES, RESTAURANTS.NAME_RES, FOODS.FNAME...	10 row(s) returned	0.00095 sec / 0.000...

Query Completed

- Top customers have highest spending in specific month in order to discover whether there is a special occasion in that month which made customers spend more than usual.

test mysql

Administration Schemas GRAB_FINAL GRAB_TRIGGER_FINAL GRAB_PROCEDURE_FINAL GRAB_DATABASE_FINAL GRAB_VIEWS_FINAL ORDER_DETAILS

Schemas Filter objects

GRAB

- Tables
 - AUDIT
 - CUSTOMERS
 - DRIVERS
 - FOODS
 - LOCATIONS
 - MESSAGES
 - ORDER_DETAILS
 - ORDERS
 - PAYMENTS
 - RATINGS
 - RESTAURANTS
 - STATUS
 - USERS
 - VOUCHERS
- Views
- Stored Procedures
- Functions
- QLBH
- QLBH69
- sys

No object selected

```

44
45 -- 6 TOP CUSTOMERS HAVE HIGHEST SPENDING IN SPECIFIC MONTH
46 • SELECT USERS.ID_USER, CONCAT(FIRST_NAME, ' ', LAST_NAME) AS 'CUSTOMER NAME', SUM(PAYMENTS.TOTAL_PRICE) AS 'SPENDING'
47   FROM USERS JOIN CUSTOMERS ON USERS.ID_USER = CUSTOMERS.ID_USER
48   JOIN ORDERS ON ORDERS.ID_CUS = CUSTOMERS.ID_CUS
49   JOIN PAYMENTS ON PAYMENTS.ID_ORDER = ORDERS.ID_ORDER
50 WHERE MONTH(ORDER_TIME) = 5 AND YEAR(ORDER_TIME) = 2021
51 GROUP BY USERS.ID_USER
52 ORDER BY SPENDING DESC
53 LIMIT 5;
54
55

```

Result Grid Filter Rows: Search Export: Fetch rows: Result 10 Read Only

ID_USER	CUSTOMER NAME	SPENDING
1021	Vu Nguyen	2774770
1010	Phung Phan	954300
1007	Hung Quoc	839915
1003	Huyen Tran	760285
1026	Huyen Anh Nguyen	681836

Action Output Time Action Response Duration / Fetch Time

14... 18:44:47 SELECT USERS.ID_USER, CONCAT(FIRST_NAME, ' ', LAST_NAME) AS 'CUSTOMER NAME', SUM(PAYMENTS.TOTAL_PRICE) AS 'SPENDING' FROM USERS JOIN C... 5 row(s) returned 0.0041 sec / 0.00001...

Query Completed

- Total GrabFood's revenue in each month

test mysql

Administration Schemas GRAB_FINAL GRAB_TRIGGER_FINAL GRAB_PROCEDURE_FINAL GRAB_DATABASE_FINAL GRAB_VIEWS_FINAL ORDER_DETAILS

Schemas Filter objects

GRAB

- Tables
 - AUDIT
 - CUSTOMERS
 - DRIVERS
 - FOODS
 - LOCATIONS
 - MESSAGES
 - ORDER_DETAILS
 - ORDERS
 - PAYMENTS
 - RATINGS
 - RESTAURANTS
 - STATUS
 - USERS
 - VOUCHERS
- Views
- Stored Procedures
- Functions
- QLBH
- QLBH69
- sys

No object selected

```

56 -- 7 TOTAL REVENUE OF EACH MONTH
57 • SELECT MONTH(ORDER_TIME) AS MONTH, SUM(TOTAL_PRICE) AS TOTAL_SPENDING
58   FROM ORDERS JOIN PAYMENTS ON ORDERS.ID_ORDER = PAYMENTS.ID_ORDER
59   GROUP BY MONTH(ORDER_TIME)
60   ORDER BY MONTH;
61

```

Result Grid Filter Rows: Search Export: Result 11 Read Only

MONTH	TOTAL_SPENDING
1	4838115
2	9681462
3	6532974
4	5597496
5	7294625

Action Output Time Action Response Duration / Fetch Time

14... 18:47:35 SELECT MONTH(ORDER_TIME) AS MONTH, SUM(TOTAL_PRICE) AS TOTAL_SPENDING FROM ORDERS JOIN PAYMENTS ON ORDERS.ID_ORDER = PAYMENTS.I... 5 row(s) returned 0.0010 sec / 0.00002...

Query Completed

- Top drivers have highest revenue in May. These are the people who bring the most profit for GrabFood so that Grab might reward these drivers.

The screenshot shows the MySQL Workbench interface with a query editor window. The schema is set to 'test mysql'. The query is:

```

62 -- 8 TOP DRIVERS HAVE HIGHEST REVENUE IN SPECIFIC MONTH
63 • SELECT ID_DRIVER, SUM(PAYMENTS.SHIP_COST) AS TOTAL_REVENUE
64 FROM ORDERS JOIN PAYMENTS ON ORDERS.ID_ORDER = PAYMENTS.ID_ORDER
65 WHERE MONTH(ORDER_TIME) = 5
66 GROUP BY ID_DRIVER
67 ORDER BY TOTAL_REVENUE DESC;
68

```

The result grid displays the following data:

ID_DRIVER	TOTAL_REVENUE
DVR04	105680
DVR11	119153
DVR14	110905
DVR06	85020
DVR02	83265
DVR10	79785
DVR07	57385
DVR09	5297

Below the grid, the status bar shows 'Query Completed'.

- Top 5 drivers have the highest ratings. This table points out drivers might not earn the most money, but they do provide excellent services which satisfy the customers.

The screenshot shows the MySQL Workbench interface with a query editor window. The schema is set to 'test mysql'. The query is:

```

69 -- 9 DRIVERS MANNER
70 • SELECT ORDERS.ID_DRIVER, CONCAT(FIRST_NAME, ' ', LAST_NAME) AS NAME, STAR, COUNT(ORDERS.ID_ORDER) AS SOCHUYEN
71 FROM DRIVERS JOIN USERS ON DRIVERS.ID_USER = USERS.ID_USER
72 JOIN ORDERS ON ORDERS.ID_DRIVER = DRIVERS.ID_DRIVER
73 GROUP BY ORDERS.ID_DRIVER
74 ORDER BY STAR DESC;
75

```

The result grid displays the following data:

ID_DRIVER	NAME	STAR	SOCHUYEN
DVR09	Khuyen Nguyen	4	3
DVR15	Nguyen Hoang	3.83333	6
DVR03	Thien Doan	3.66667	3
DVR12	Trung Lam	3.5	2
DVR04	Yen Nguo	3.33333	3
DVR08	Thu Nguyen	3.33333	3
DVR01	Thanh Lam	3.16667	6
DVR05	Thien Pham	3	1
DVR06	Tien Tran	3	3
DVR10	Hung Kieu	3	2
DVR14	Hai Tran	3	3
DVR11	Kien Do	2.75	1
DVR13	Thuy Nguyen	2.33333	3
DVR07	Kha Nguo	2.2	5
DVR05	Giang Pham	2	2

Below the grid, the status bar shows 'Query Completed'.

- Customers who have the birthday in this month and spent more than 700.000 VNĐ to help restaurants to offer suitable promotions to customers who have the birthday in this month and have spent a certain amount of money for food delivery service.

The screenshot shows the MySQL Workbench interface with a query editor and results grid. The query retrieves customer information based on their birthday and total spending:

```

77 -- 10 CUSTOMERS WHO HAVE BIRTHDAY THIS MONTH AND HAVE SPENT > 700000
78 • SELECT USERS.ID_USER, USERS.LAST_NAME AS 'CUSTOMER NAME', USERS.BIRTHDATE AS BIRTHDATE, SUM(PAYMENTS.TOTAL_PRICE) AS TOTAL_SPENDING
79   FROM USERS JOIN CUSTOMERS ON USERS.ID_USER = CUSTOMERS.ID_USER
80     JOIN ORDERS ON ORDERS.ID_CUS = CUSTOMERS.ID_CUS
81       JOIN PAYMENTS ON PAYMENTS.ID_ORDER = ORDERS.ID_ORDER
82 WHERE MONTH(USERS.BIRTHDATE) = MONTH(NOW())
83 GROUP BY USERS.ID_USER
84 HAVING TOTAL_SPENDING > 700000;
85

```

The results grid shows one row:

ID_USER	CUSTOMER NAME	BIRTHDATE	TOTAL_SPENDING
1008	Vo	1978-06-25	1754986

Below the results, the "Execution Plan" pane shows the query's execution details:

Action Output	Time	Action	Response	Duration / Fetch Time
14... 19:58:46		SELECT USERS.ID_USER, USERS.LAST_NAME AS 'CUSTOMER NAME', USERS.BIRTHDATE AS BIRTHDATE, SUM(PAYMENTS.TOTAL_PRICE) AS TOTAL_SPENDING...	1 row(s) returned	0.0014 sec / 0.00001...

Query Completed

5.2. VIEW

- Customer information:** View CUSTOMER_INFORMATION displays some basic information of customers, such as name, gender, phone, location. Drivers could have this information about customer to deliver food easier. Some private information such as credit card number or security code of customers must have been secured.

test mysql

Administration Schemas GRAB_FINAL GRAB_TRIGGER_FINAL GRAB_PROCEDURE_FINAL GRAB_DATABASE_FINAL GRAB_VIEWS_FINAL ORDER_DETAILS

Filter objects Limit to 1000 rows

```

88
89
90 -- 1 VIEW CUSTOMER_INFORMATION
91 • CREATE VIEW CUSTOMER_INFORMATION AS
92 SELECT C.ID_CUS, U.LAST_NAME, U.FIRST_NAME, U.GENDER, U.PHONE, LOCATION FROM USERS U
93 JOIN CUSTOMERS C ON C.ID_USER = U.ID_USER JOIN LOCATIONS L ON L.ID_LOCATION = C.ID_LOCATION;
94
95 • SELECT * FROM CUSTOMER_INFORMATION;
96

```

Result Grid Filter Rows Search Export:

ID_CUS	LAST_NAME	FIRST_NAME	GENDER	PHONE	LOCATION
CUS04	Nguyen	Le Co	FEMALE	75417	Ha Dong
CUS18	Khuat	Van	FEMALE	32947	Ha Dong
CUS24	Pham	Kim Anh	FEMALE	27964	Ha Dong
CUS26	Nguyen	Huyen Anh	FEMALE	54456	Ha Dong
CUS27	Nguyen	Hung	MALE	25940	Nam Tu Liem
CUS28	Nguyen	Thien	MALE	14940	Bac Tu Liem
CUS02	Dao	Tuyet Anh	FEMALE	17894	Bac Tu Liem
CUS12	Tran	Duong	FEMALE	46312	Bac Tu Liem
CUS14	Do	My	OTHER	60937	Bac Tu Liem
CUS15	Tran	Vu	MALE	30267	Bac Tu Liem
CUS29	Nguyen	Dat	MALE	23295	Thanh Xuan
CUS01	Phung	Tuan	MALE	94576	Hoan Kiem
CUS07	Quoc	Hung	MALE	59953	Hoan Kiem
CUS08	Nguyen	Quyen	MALE	14941	Hoan Kiem
CUS06	Luu	Tue	MALE	69336	Hoang Mai
CUS17	Ly	Tuong	MALE	50986	Hoang Mai
CUS32	Nguyen	Binh	MALE	40774	Hoang Mai
CUS09	Nguyen	Dung	MALE	36990	Hai Ba Trung
CUS19	Pham	Phuong	MALE	78894	Hai Ba Trung
CUS25	Tran	Quyen	MALE	36559	Hai Ba Trung
CUS03	Tran	Huyen	FEMALE	36409	Long Bien
CUS16	Vu	Thuy	FEMALE	71150	Long Bien
CUS18	Nguyen	Ho	MALE	36443	Long Bien
CUS30	Do	Huong	FEMALE	63639	Tay Ho
CUS28	Tran	Nga	FEMALE	39921	Cau Giay
CUS15	Le	Hung	MALE	17412	Dong Da
CUS05	Tran	Le	MALE	14942	Dong Da
CUS05	Nguyen	Tu Anh	FEMALE	66995	Ba Dinh
CUS09	Huynh	The Linh	FEMALE	91410	Ba Dinh
CUS11	Dang	Vu	MALE	90782	Ba Dinh
CUS13	Do	Huy	MALE	23306	Ba Dinh

CUSTOMER_INFORMATION 15

Action Output Time Action Response Duration / Fetch Time

14... 19:59:53 SELECT * FROM CUSTOMER_INFORMATION LIMIT 0, 1000 35 row(s) returned 0.0054 sec / 0.0000...

Query Completed

- Bill:** View BILL print out the bill of accomplished order (after customers ordering the food and the chosen restaurant has finished cooking). Informations which were printed on the bill consist of restaurant's name, customer's name, driver's name, food, location, payments in details.

test mysql

Administration Schemas GRAB_FINAL GRAB_TRIGGER_FINAL GRAB_PROCEDURE_FINAL GRAB_DATABASE_FINAL GRAB_VIEWS_FINAL ORDER_DETAILS

Filter objects Limit to 1000 rows

```

98 -- 2 VIEW BILL
99 • CREATE VIEW BILL AS
100 SELECT O.ID_ORDER,
101     R.NAME_RES AS RESTAURANT,
102     GROUP_CONCAT(F.FNAME SEPARATOR ', ') AS MENU,
103     CONCAT(Z.DRINAME1, ' ', Z.DRINAME2) AS DRIVER,
104     CONCAT(Z.CUSNAME1, ' ', Z.CUSNAME2) AS CUSTOMER,
105     L.LOCATION AS LOCATION,
106     P.IS_PAID,
107     P.TOTAL_PRICE AS TOTAL
108 FROM ORDERS O
109 JOIN ORDER_DETAILS OD ON OD.ID_ORDER = O.ID_ORDER
110 JOIN FOODS F ON F.ID_FOOD = OD.ID_FOOD
111 JOIN RESTAURANTS R ON R.ID_RES = O.ID_RES
112 JOIN LOCATIONS L ON L.ID_LOCATION = O.ID_LOC_CUS
113 JOIN PAYMENTS P ON P.ID_ORDER = O.ID_ORDER
114 JOIN (SELECT *
115     FROM (SELECT ID_ORDER AS OD1, ID_CUS AS CUSID, FIRST_NAME AS CUSNAME1, LAST_NAME AS CUSNAME2
116             FROM ORDERS O
117             JOIN CUSTOMERS C ON O.ID_CUS = C.ID_CUS
118             JOIN USERS U ON U.ID_USER = C.ID_USER ) AS A
119             JOIN (SELECT ID_ORDER AS OD2, ID_DRIVER AS DRIID, FIRST_NAME AS DRINAME1 ,
120                   LAST_NAME AS DRINAME2
121                   FROM ORDERS O
122                   JOIN DRIVERS C ON O.ID_DRIVER = C.ID_DRIVER
123                   JOIN USERS U ON U.ID_USER = C.ID_USER ) AS B
124             ON A.OD1 = B.OD2) AS Z ON O.ID_ORDER = Z.OD2
125             GROUP BY O.ID_ORDER;
126

```

Result Grid Filter Rows Search Export:

ID_ORDER	RESTAURANT	MENU	DRIVER	CUSTOMER	LOCATION	IS_PAID	TOTAL
OD01	KFC	French fries, Med chicken	Thanh Lam	Tu Anh Nguyen	Dong Da	1	809145
OD02	NGOAM	burger	Thi Nguyen	Tu Anh Nguyen	Ba Dinh	1	179020
OD03	Labonita	sandwich, coffee	Nguyen Hoang	Tuan Phung	Hoan Kiem	1	568564

BILL 16

Read Only

Query Completed

- **Restaurants that have highest revenue in each district:** View DHR displays the list of restaurants that have the highest revenue from districts by descending order so that they can promote the marketing strategy for that place, and also to catch the trend of the market.

The screenshot shows the MySQL Workbench interface with the 'test mysql' database selected. In the left sidebar, under the 'Schemas' section, the 'GRAB3' schema is expanded, showing various tables like AUDIT, CUSTOMERS, DRIVERS, FOODS, LOCATIONS, MESSAGES, ORDER_DETAILS, ORDERS, PAYMENTS, RATINGS, RESTAURANTS, STATUSS, USERS, and VOUCHERS. Below these are Views, Stored Procedures, and Functions, with two specific ones highlighted: 'QLBH' and 'QLBH69'. The main pane displays the SQL code for creating the 'DHR' view:

```

129
130 -- 3 VIEW RESTAURANTS HAVE HIGHEST REVENUE EACH DISTRICT
131 • CREATE VIEW DHR AS
132   SELECT T.ID_LOCATION, T.LOCATION, T.NAME_RES AS RESTAURANT, MAX(T.REVENUE) AS 'TOTAL REVENUE' FROM
133   (SELECT LOCATIONS.ID_LOCATION, LOCATIONS.LOCATION, NAME_RES, SUM(PAYMENTS.RES_COST) AS REVENUE
134   FROM RESTAURANTS JOIN ORDERS ON RESTAURANTS.ID_RES = ORDERS.ID_RES
135   JOIN PAYMENTS ON PAYMENTS.ID_ORDER = ORDERS.ID_ORDER
136   JOIN LOCATIONS ON RESTAURANTS.ID_LOCATION = LOCATIONS.ID_LOCATION
137   GROUP BY RESTAURANTS.ID_RES) AS T
138   GROUP BY T.ID_LOCATION
139   ORDER BY ID_LOCATION;
140
141 130% 19:141

```

The 'Result Grid' tab is active, showing the results of the query:

ID_LOCATION	LOCATION	RESTAURANT	TOTAL REVENUE
1	Ha Dong	NOCOAM	785000
2	Nam Tu Liem	Taco Non La	2550000
3	Bac Tu Liem	KFC	2740000
4	Thanh Xuan	Labonte	3089000

At the bottom, the status bar indicates 'Query Completed'.

- **Districts have the most orders:** View MOST_ORDERS displays the list of orders of from districts (through restaurants in each district) by descending order. Then it shows the district with the most orders so that they can send more drivers to that place to meet the customer's need.

```

143
144 -- 4 VIEW DISTRICTS HAVE THE MOST ORDERS
145 • CREATE VIEW MOST_ORDERS AS
146   SELECT LOCATIONS.LOCATION, COUNT(ORDERS.ID_ORDER) AS TOTAL_ORDER
147   FROM LOCATIONS JOIN RESTAURANTS ON LOCATIONS.ID_LOCATION = RESTAURANTS.ID_LOCATION
148   JOIN ORDERS ON ORDERS.ID_RES = RESTAURANTS.ID_RES
149   GROUP BY LOCATIONS.ID_LOCATION
150   ORDER BY TOTAL_ORDER DESC
151   LIMIT 3;
152
153 • SELECT * FROM MOST_ORDERS;
154

```

LOCATION	TOTAL_ORDER
Ha Dong	20
Nam Tu Liem	13
Bac Tu Liem	13

Action Output Time Action Response Duration / Fetch Time
14... 20:06:35 SELECT * FROM MOST_ORDERS LIMIT 0, 1000 3 row(s) returned 0.0028 sec / 0.00001...

- Restaurants which have highest star in each district:** It is helpful for customers who come to one district for the first time and want to find out which one is the best restaurants there.

```

154
155 -- 5 VIEW RESTAURANT WHICH HAVE HIGHEST STAR IN EACH DISTRICT
156 • CREATE VIEW HIGH_RES_DISTRICT AS
157   SELECT NAME_RES, MAX(STAR) AS STAR, LOCATION
158   FROM RESTAURANTS JOIN LOCATIONS ON RESTAURANTS.ID_LOCATION = LOCATIONS.ID_LOCATION
159   GROUP BY LOCATIONS.ID_LOCATION;
160
161 • SELECT * FROM HIGH_RES_DISTRICT;
162
163
164

```

NAME_RES	STAR	LOCATION
KFC	3.25	Bac Tu Liem
NGOAM	3	Ha Dong
Taco Non La	3	Nam Tu Liem
Lobone	2	Thanh Xuân

Action Output Time Action Response Duration / Fetch Time
14... 20:11:06 SELECT * FROM HIGH_RES_DISTRICT LIMIT 0, 1000 4 row(s) returned 0.00099 sec / 0.000...

6. TRIGGERS

Trigger is a procedure that is executed on the server side when an event such as Insert, Delete, or Update occurs. Trigger is a special type of stored procedure (with no parameters) that is executed automatically when there is a data modification event. In our project, Triggers are stored and managed in the database, which is used in case we want to update and control changes to the data in the table. The script and results are in GRAB_TRIGGER_FINAL.sql file. In details, we created 11 Triggers:

6.1. TRIGGER ADD_SHIP_COST

Trigger ADD_SHIP_COST will add the shipping cost into PAYMENTS. It will take the distance from ID_LOC_CUS in ORDERS to ID_LOCATION in RESTAURANTS. Meanwhile, the shipping cost will be calculated by Procedure GET_SHIP_COST.

6.2. TRIGGER ADD_RES_COST

It calculates the total food price in each order. Simultaneously, it will be added into PAYMENTS.

6.3. TRIGGER ADD_PRICE_OD

After customer inserts food's name and quantity, Trigger ADD_PRICE_OD will calculate the price for that food and add into ORDER_DETAILS.

6.4. TRIGGER ADD_VOUCHER

Trigger ADD_VOUCHER will add a voucher into VOUCHER_PM bases on the payment method customer has chosen. It also set IS_PAID to 0 if payment method is COD. With others, it will set to 1.

6.5. TRIGGER UPDATE_LOCATION

When customers insert into ORDERS, if they do not insert another address, UPDATE_LOCATION will set the ID_LOC_CUS to customers who made that order as default.

6.6. TRIGGER STOCK_LEFT

After every ORDER_DETAILS, it will update STOCKS by deducting the amount customers ordered. If the amount ordered is larger than the stocks, the system will not receive the food and displays a message “Only {stocks} {fname} left”.

6.7. TRIGGER UPDATE_MAX_USE

It is used to update MAX_USE in VOUCHERS by deducting by 1. If MAX_USE is less than or equal to 0, it will display a message “{name of voucher} out of voucher”. Also, if MAX_USE is equal to 0, the system will automatically update the discount (%) of that voucher to 0.

6.8. TRIGGER TOTAL_PRICE

This Trigger takes the sum of discounted food price after applying vouchers and shipping cost, then add into PAYMENTS.

6.9. TRIGGER UPDATE CANCEL

The objective of this Trigger is to check whether the order can be canceled or not. If the order could be canceled, the system will automatically set the RES_COST & SHIP_COST in table PAYMENTS to 0, which means that the restaurant and the driver do not gain any money from this order. Else if the order cannot be canceled, it will insert into table MESSAGES line “You cannot cancel this order!”.

6.10. TRIGGER UPDATE_DRIVER_RES

It will update ID_RES and ID_DRIVER into RATINGS after customers insert ID_ORDER.

6.11. TRIGGER ADD_STAR

Update STAR in DRIVER and STAR in RESTARANTS by calculating average of all the ratings.

7. PROCEDURES

Stored Procedure is compiled and stored in memory on initialization. That means it will execute faster than sending each SQL statement to MySQL. In this project, to avoid recompiling many times, which is very time consuming compared to precompiled, we use stored procedure to speed up code execution and can reuse code. The script and results are in GRAB_VIEWS_FINAL.sql file.

7.1. PROCEDURE UPDATE_STATUS

DRIVER will update the status of an order. If he or she updates the status of order to 4 (TAKEN), it will set the TAKEN_TIME in ORDERS to the moment.

7.2. PROCEDURE POP_MESSAGE

It will print out the latest messages.

7.3. PROCEDURE LAT_LO

This Procedure take an ID_LOCATION and set a pair of latitude and longitude to the location in order to simplify the calculating process.

7.4. PROCEDURE GET_SHIP_COST

After getting the latitude and longitude of the location, GET_SHIP_COST will calculate the shipping cost customers must pay by taking the distance times 8000(VND/km).

7.5. PROCEDURE AGE_GROUP_FOOD

After inputting the age range, the procedure will print out foods that have been ordered the most by customers that are in the range.

7.6. PROCEDURE GROUP_AGE_F

It will print out all the customers in some age ranges' favorite food.

7.7. PROCEDURE AGE_GROUP_SPENDING

It will print out the total amount of money spent by customers in the given age range.

7.8. PROCEDURE GROUP_AGE_S

It will print out the total amount of money spent by customers in some age ranges.

THE END