UOW
COLLEGE
HONG KONG
香港伍倫貢學院

AST 20401

Database Systems and Design

Group Project

Project Title:

" Online Food Ordering System Database Design"

Group Details:

Ip Tsun Ting (H10006777)	• Order Section Procedures • Order Section Dummy Data • ER Modelling and Refinement
Shadev Kharel (H10006790)	• Restaurant View Procedures • Restaurant and Delivery Section Dummy Data • ER Modelling and Normalization
Chow Cheuk Yin (H10006718)	• User Section Table and Dummy • Business Analysis Queries(1/2)
Ip Chun Lam (H10006776)	• Drivers Section Table Design • Business Analysis Queries(1/2)

Submitted to: Dr. Ka Man Pang

Table of Contents:

1. Background

In this Internet and e-commerce world, the food delivery industry is rapidly evolving. Various delivery platforms like Food Panda, Uber Eats, DoorDash, etc are operating as intermediary agents between globally placed restaurants and end customers. Databases are the single source of truth for platforms like Food Panda as they store structured information regarding food items, restaurant details, menus, orders, and their statuses.

2. Introduction

By analyzing the B2C(Business to Consumer) model of these platforms, we tried to implement a database for effectively managing basic-level operations to fulfill some of their functional requirements. Therefore, to achieve the goals, In the project, we divided the database into three different views: Users, Restaurants, and Delivery. This is because the **Users section** can be fully fledged for storing personal and sensitive login information. There was also a need for an independent **restaurant view** to ensuring better communication between the restaurant and the delivery platforms. . The **delivery view** we have integrated in our project however is very simple and just contains, order_info and the info of the delivery agent. Despite the simplicity, the project aims to optimize operational efficiency and handle a significant number of functions.

3. Entity – Relationship Diagram

As mentioned in section 1, we have classified different entity types based on basic operational requirements. A detailed explanation of ER Model specifications is given below:

Major Entities:

a. Users (**user_ID,** f_name , l_name, address, email, class, Discount_rate ...)

b. Payment Method (**Payment_Method, Account_Number**)

c. Drivers(**Driver_ID,** License_Number , Name, ……)

d. Orders (**order_id, order_items,** accountNumber, order_amount, driver_ID, menu_items order_status …)

e. **Restaurant (** **rest_id,** name)

f. Dish (dish_ID , dish_price , dish_name, veg)

g. Branch (**Branch_ID, rest_id ,** address, phone ….)

h. **Menu** (**Menu_ID**, Menu_type, ….)

Assumptions:

a. A specific restaurant owns a dish**.** *(for eg: McDonald's owns dishes like Mc Slurry, MC Burger, etc.)*

b. Every restaurant can have multiple branches, but the branch_id for each branch is 1, 2, 3… or soon. McDonalds branches 1, 2, 3 & KFC branches 1, 2, 3 are different.

c. Each branch of a restaurant is allowed to have at most three different kinds of menus: breakfast, lunch, or dinner.

d. One order can have dishes from multiple restaurants. (For example: A Food Panda user can order a pizza from KFC, and in the same order, they can add a burger from McDonald's. The driver has to pick up orders from both restaurants and then deliver them to the user.

Basic Relationships:

a. Each(1:1) **Users** *add* one or more(1:n) **Payment Method**.
For example: User 1 'John David' can have Alipay account no. 123 and Octopus account no—456, both as their Payment Method.

b. A **Payment Method(1:1)** is *used to* order either 0 or many**(0:n) Orders**.
For example: John David's Alipay account can order on December 3, 2024, and on December 4, 2024, which are two different orders.

c. A **Driver(1:1)** *takes* either no orders at a time and can take multiple **orders over time(0:n)**.
For example: A driver newly joined might not have taken a single order till now for the company.

d. A **Restaurant(1:1)** *operates* many branches, and it has to have at least 1 **(1:n) Branch.**
For example: McDonald's can have 100 different branches across various cities, but to exist as a restaurant, a restaurant needs to operate at least one branch.

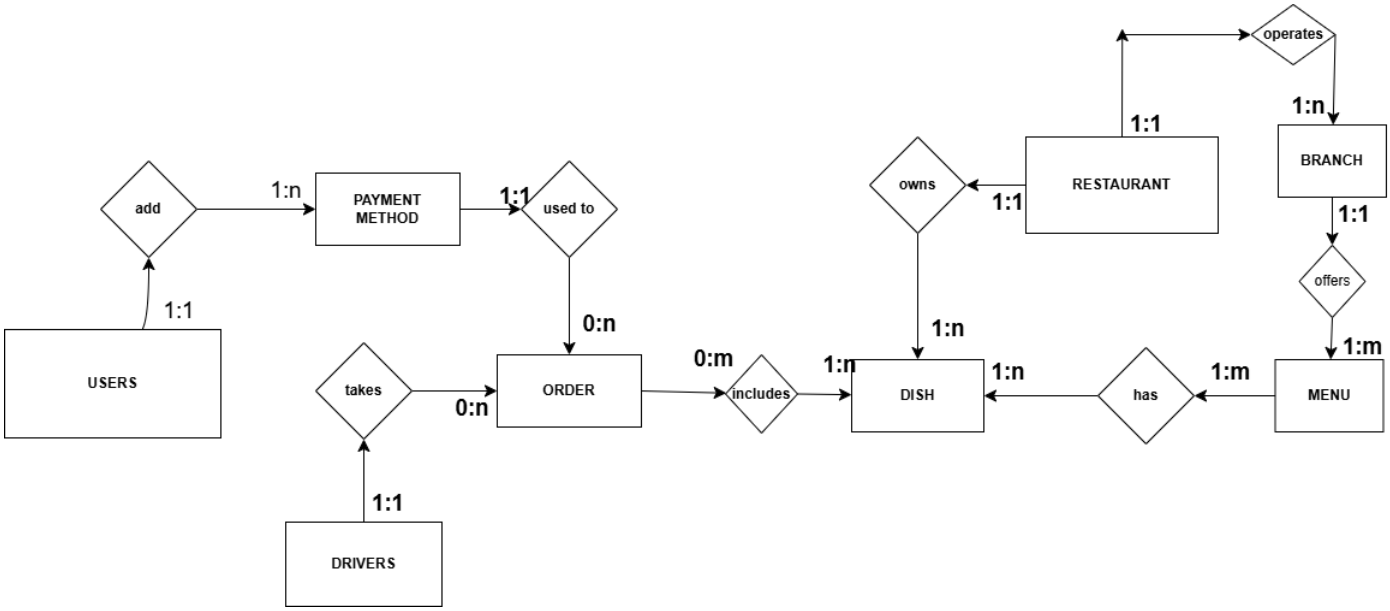
e. A **Restaurant(1:1)** *owns* multiple **(1:n) dishes**.
For example: McDonald's has multiple sets of burgers, multiple drinks, and soon.

f. A **Branch(1:1)** *offers* at least 1 and at most 3 **(1:m) Menu** i.e. Breakfast/ Lunch / Dinner, *For example: McDonald's in some regions of Sai Kung may not offer breakfast in the early morning, but in urban cities like Central offers all 3 menus because of the demand.*

g. A **Menu (1:m)** *has* multiple **Dishes(1:n)**, and the dishes can be on multiple menus, as dishes are owned by restaurants and menus by branch.

h. An **Order(0:m)** *includes* at least one and at most multiple **(1:n)Dishes from the menu**, and a dish can be in multiple orders too.

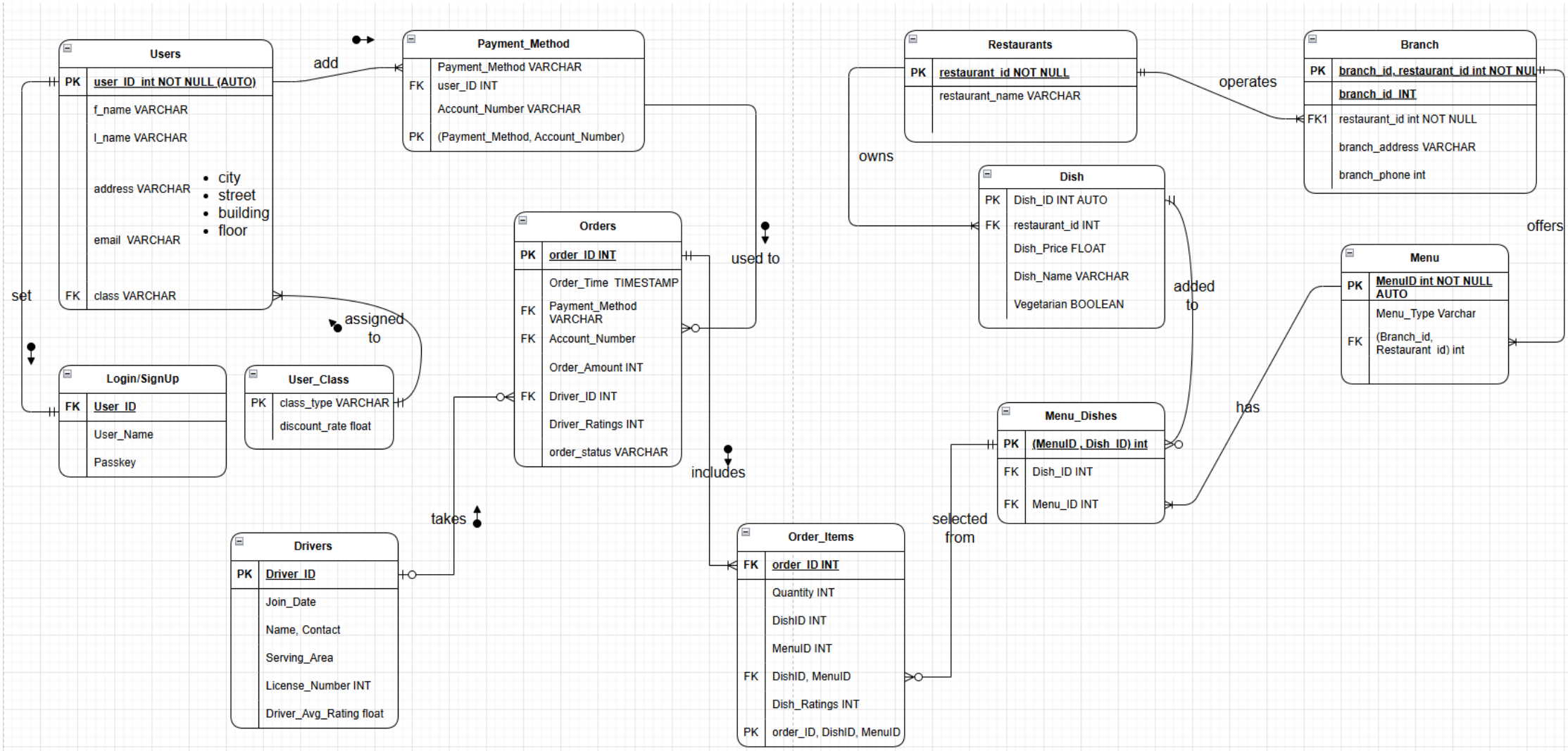
Unnormalized ER Model for our Project.



Problems Faced with this Unnormalized Design:

1. This ER model has multiple many-to-many relationships, such as between Menu and Dish, Order and Dish, etc. To remove potential redundancies, we added linking tables like the Menu_Dish Table between Menu and Dish, which alone handles redundancies.
2. No foreign key references in each entity leads to heavy data repetition across multiple tables.
3. Without normalization, mySQL queries must access repeated data, which degrades the query performances.
4. Also, without normalization, various anomalies like deletion, updation, and insertion can occur. For example, to make an order, a dummy payment method should have to be added to the orders table, causing an insertion anomaly.

Normalized- Entity Relationship Diagram for Food Delivery Management System - Designed in Draw IO



4. Normalizing and Refining ER Model.

As previously mentioned, we tried to refine the original ER model by normalizing it from 1st up to a 3rd normal form. Some notable 3NF improvements in brief include:

1. The addition of the **Menu_Dishes** table, which references foreign keys from Dish_ID and Menu_ID and makes it a composite primary key for itself, prevents data repetition across various tables. For example, in this Dish Table, we frequently add each dish to each row, as a result, the other irrelevant column has to be repeated redundantly, causing anomalies, mostly updation and insertion.

Dish Table(Before)

Dish_ID	Menu_ID	Rest_ID	....others
1	1	5	
1	2	5	
1	3	5	
1	4	5	

Menu_Dish Table

Dish_ID	Menu_ID
1	1
1	2
1	3
1	4

Dish Table(After)

Dish_ID	Rest_ID	others..
1	1	

(fig: illustrating the need for menu_dish table)

2. Rather than having **Users(** **user_ID,** f_name , l_name, address, email, class, discount_rate, ...) table we divided it to **class(class_type, discount_rate)** and **Users(user_ID,** f_name , l_name, address, email, class(FK from class)**) ,** to remove transitive dependency, i.e. class_type → discount_rate.

3. Finally, rather than having order_items in **Orders** (**order_id, order_items,** …..) , we made a different **order_items** relation to handle this case of many to many relationships, as a dish can be in multiple orders, and an order can have many dishes.

4.1. Major Tables and their 3NF Check:

a. **Users: user_ID → f_name , l_name, address, email, class** making it a 2NF & the table doesn't have any transitive dependencies, so it is in 3NF.

b. **Restaurants: restaurant_id → restaurant_name ,** no other functional dependencies, so it's in 3NF.

c. **Branches: restaurant_id, branch_id (**compositely determine**) → branch_address, branch_phone.** Also, restaurant_id cannot determine branch_id. No transitive dependencies. So, it's in 3NF.

branch_id	restaurant_id	branch_address	branch_phone
1	1	123 Nathan Road, Kowloon	52669796
2	1	Tai Wai, 2/F, The Wai	88444488
1	2	456 Queen's Road, Hong Kong	1234568
2	2	789 Tsim Sha Tsui, Kowloon	1234569

d. **Dish: Dish (dish_ID**, dish_price, dish_name, veg), it has no composite key so it is already in 2NF, also no non-key attribute can determine the rest. So, it's in 3NF.

e. **Menu: Menu_ID → Menu_type, Branch_ID, Restaurant_ID.** In this table, we added Restaurant_ID for query optimization. No composite key so 2NF. Either of Menu_type, branchID, or RestaurantID cannot determine each other. So, it's in 3NF.

f. Menu_Dish: A linking table with two foreign key references to Dish and Menu, either of Dish_ID or Menu_ID cannot determine each other. So, no partial and transitive dependencies. Therefore, it's in 3NF.

g. Orders: Order_ID determines each of the non-key attributes uniquely. No transitive dependencies as each attribute are independent of each other. Therefore, it is in 3NF

h. Order_items: (order_id, dishID, menuID) as a composite key. Only one non-key attribute, so no transitive dependency. Therefore, it is in 3NF.

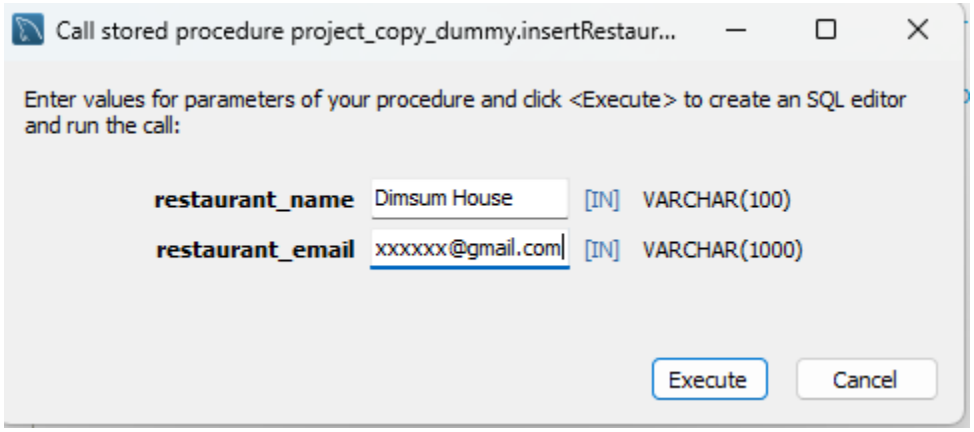
order_ID	quantity	dishID	menu_ID	Dish_Ratings
1	2	1	1	3
1	1	3	1	4
2	3	7	1	4
3	1	6	2	NULL
5	1	1	2	2

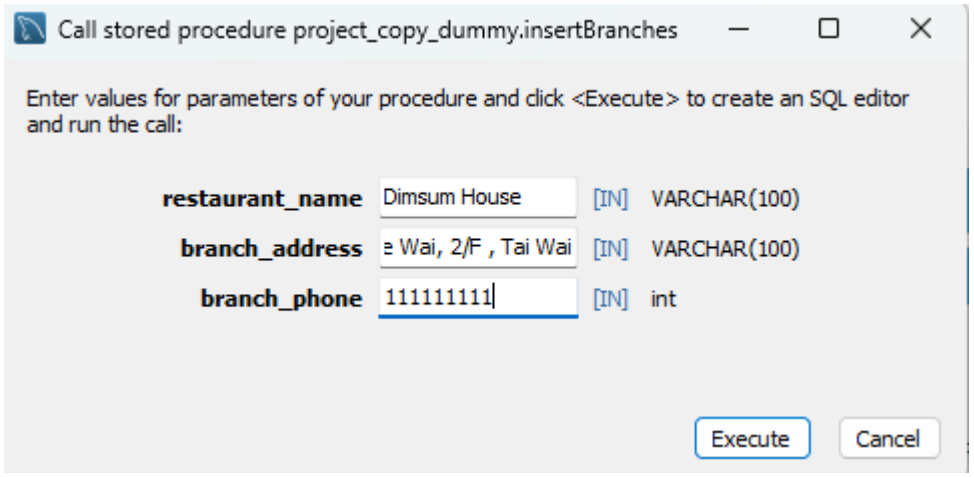
5. Procedures and Functions:

For the project, we have designed 18 different procedures to support significant functional requirements. Some of them are:

5.1. Restaurant View Procedures:

5.1.1. **Insert Procedures**: For inserting new restaurants and branches that offer food, that can be delivered using our platform.





5.1.2. **Filter Procedures:** To extract certain information as needed, Similar to how we go to the Foodpanda website and see restaurants close to us, filter vegetarian restaurants, filter by city, or filter by time.

a.

```
drop Procedure FilterbyCity
Delimiter //
Create Procedure FilterbyCity(
 in city varchar(50)
)
Begin
Select  concat (r.restaurant_name , " on " , b.branch_address) as "Restaurants Close To You"
from restaurants r inner join branch b
on b.restaurant_id  = r.restaurant_id
where branch_address like concat( '%', city, '%') ;
End //

Delimiter ;

-- CALL
/*Test Demonstration(WORKING CONDITION)*/
call FilterbyCity('Tai Wai');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Restaurants Close To You
McDonalds HK on Tai Wai, 2/F, The Wai

b.

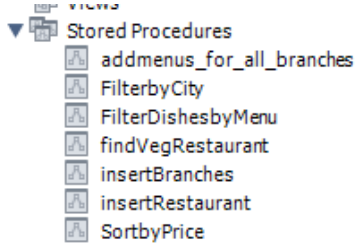
```
Create Procedure FilterDishesbyMenu(
in access_time time ,
in city VARCHAR(50),
in restaurant VARCHAR(50)
)
Begin
Declare meal_type VARCHAR(50) ; Declare Bid int ; Declare Rid int ; Declare Mid int ;
Select restaurant_id into Rid
from restaurants
where lower(restaurant_name) like lower(restaurant) ;

Select branch_id into Bid
from branch
where restaurant_id = Rid AND branch_address like (concat('%', city , '%')) ;

IF(access_time between '06:00:00' AND '11:59:59') THEN SET meal_type = 'BREAKFAST' ;
ELSEIF( access_time between '12:00:00' AND '17:59:59') THEN SET meal_type = 'LUNCH' ;
ELSE SET meal_type = 'DINNER' ;
END IF;

Select Menu_ID into Mid
from Menu
where restaurant_id = Rid  AND branch_id = Bid  AND lower(menu_type) like lower(meal_type) ;

Select Dish_Name, Dish_Price, Vegetarian
from menu_dishes m  inner join dish d
on m.Dish_ID = d.Dish_ID
where m.MenuID = Mid ;

END //
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Dish_Name	Dish_Price	Vegetarian
Big Mac	10.99	0
Fries	8.50	1
McVeggie Burger	20.00	1
Double Cheeseburger	15.50	0
Sundae	7.75	1

This procedure takes the city and restaurant from the user, and according to the user's access time, filters out menu_types like breakfast/lunch/dinner, and then displays the dishes available in that restaurant in that city at that time.

In addition to these notable procedures, we designed some other useful procedures for restaurant view only, which can be accessed in the *restaurantView_procedures.sql* file, they are:

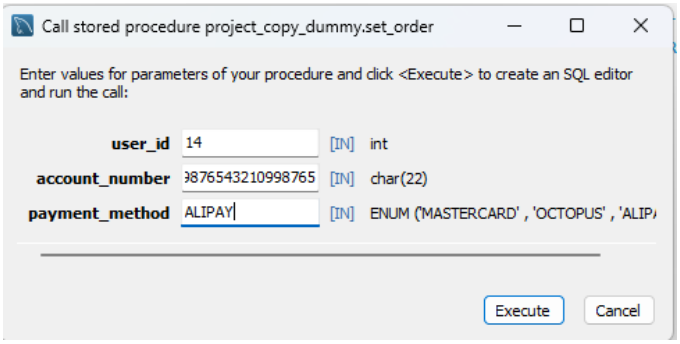


(Note: Workings of some of these are highlighted in the appendix 5)

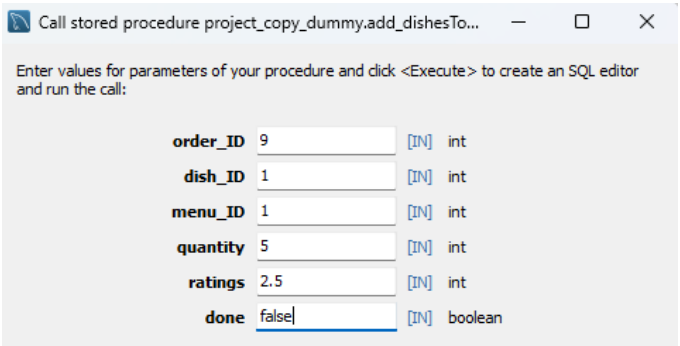
5.2. Order View Procedures:

For users, various order procedures are designed to facilitate the final delivery. Some procedures include:

5.2.1 Create an Order and Add Dishes to that Order



Description: The procedure ***'set_order'*** takes a unique user_id, an account number they wish to pay using, and the mode of payment. Then see if the user has already added the payment method for his account. If it is confirmed the order is set giving it a unique order ID.



Description: Then the ***'add_dishesToOrders'*** procedure is called which helps in adding dishes in a single order. In this case dish of a menu is added to the order, with quantity which helps to calculate the price.(Appendix 1)

5.2.2. getOrderHistory( ) Procedure

```
delimiter //
CREATE  PROCEDURE getOrderHistory()
BEGIN
    select * from orders where done=true;
END //
delimiter ;
```

order_ID	user_ID	Order_Time	account_number	driver_id	Payment_Method	order_amount	done	driver_rating
1	1	2024-12-06 13:08:03	4111111111111111	1	MASTERCARD	100	1	5
2	2	2024-12-06 13:08:03	9876543210123456	2	ALIPAY	200	1	4
3	1	2024-12-06 13:08:03	4333444433334444	3	MASTERCARD	150	1	5
5	6	2024-12-06 18:53:21	5222333344445555	1	MASTERCARD	34.97	1	2
7	6	2024-12-06 19:31:51	987654321987	3	OCTOPUS	11.381	1	2
8	14	2024-12-06 20:44:12	9876543210998765	6	ALIPAY	39.9784	1	5

Description: Selects all orders that have been delivered as mentioned in code ‘done= true”

5.2.3. get_unfinish_order() Procedure

```
delimiter //
CREATE PROCEDURE get_unfinish_order()
BEGIN
    select * from orders where done=false;
END //
delimiter ;
```

order_ID	user_ID	Order_Time	account_number	driver_id	Payment_Method	order_amount	done	driver_rating
9	14	2024-12-06 22:08:57	9876543210998765	6	ALIPAY	52.2025	0	0

Description: Gets all the orders that haven’t yet been marked delivered.

5.3. Delivery View:

This section is dedicated to tracking order status, rating drivers and dishes, and toggling the status of the delivery guy, i.e. whether they are currently accepting any order or not. Some procedures are:

5.3.1. Mark_Order_Done() procedure [Appendix 3]

Description: This procedure helps to set the status or the *done* attribute in the order table to true. This can happen after the driver clicks the delivered option in his application interface.

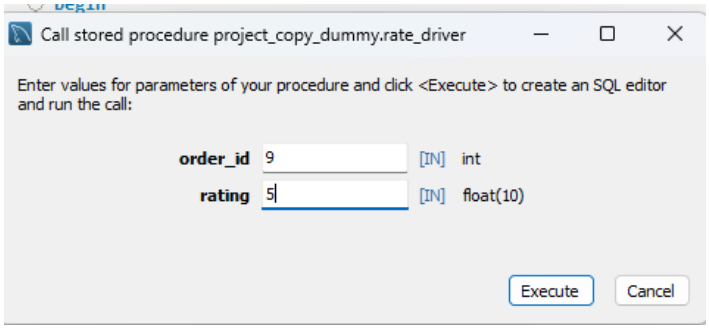
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_ID	user_ID	Order_Time	account_number	driver_id	Payment_Method	order_amount	done	driver_rating
1	1	2024-12-06 13:08:03	4111111111111111	1	MASTERCARD	100	1	5
2	2	2024-12-06 13:08:03	9876543210123456	2	ALIPAY	200	1	4
3	1	2024-12-06 13:08:03	4333444433334444	3	MASTERCARD	150	1	5
5	6	2024-12-06 18:53:21	5222333344445555	1	MASTERCARD	34.97	1	2
7	6	2024-12-06 19:31:51	987654321987	3	OCTOPUS	11.381	1	2
8	14	2024-12-06 20:44:12	9876543210998765	6	ALIPAY	39.9784	1	5
9	14	2024-12-06 22:08:57	9876543210998765	6	ALIPAY	52.2025	1	0

The previous unfinished order in section 4.2.2 with order_id: 9 is now in the order_history table as **done** and is flagged true with the marker_order_done() call.

5.3.2. Rate_Driver ( ) Procedure [Appendix 4]

Description: This allows the user to give ratings /5 to the driver for his service after the delivery is done. This marks the individual delivery rating, but towards the end of this procedure, the next procedure named *update_driver_rating* is called which will update the aggregate ratings of driver.



order_ID	user_ID	Order_Time	account_number	driver_id	Payment_Method	order_amount	done	driver_rating
1	1	2024-12-06 13:08:03	4111111111111111	1	MASTERCARD	100	1	5
2	2	2024-12-06 13:08:03	9876543210123456	2	ALIPAY	200	1	4
3	1	2024-12-06 13:08:03	4333444433334444	3	MASTERCARD	150	1	5
5	6	2024-12-06 18:53:21	5222333344445555	1	MASTERCARD	34.97	1	2
7	6	2024-12-06 19:31:51	987654321987	3	OCTOPUS	11.381	1	2
8	14	2024-12-06 20:44:12	9876543210998765	6	ALIPAY	39.9784	1	5
9	14	2024-12-06 22:08:57	9876543210998765	6	ALIPAY	52.2025	1	5

(*fig showing ratings updates in orders table*)

Driver_ID	driver_rating	Join_Date	fName	lName	contact_number	serving_area	idle	license_no
1	3.5	2023-01-15	Chan	Tak-Wai	12345678	Tsim Sha Tsui	1	987654321
2	0	2022-06-22	Li	Ming	23456789	Causeway Bay	0	123456789
3	3.5	2021-11-30	Wong	Chun-Ho	34567890	Mong Kok	1	234567890
4	0	2023-03-14	Ng	Pui-Fong	45678901	Kowloon Bay	1	345678901
5	0	2022-07-25	Leung	Ka-Yan	56789012	Wan Chai	0	456789012
6	5	2021-08-10	Cheung	Ho-Kwan	67890123	Sha Tin	1	567890123

(fig showing aggregate rating of driver_id 6 changed in drivers table)

6. Business Analysis Queries.

Finally, we also designed some efficient queries to effectively analyse some key business statistics, to help restaurants make business decisions properly in future.

1. Get Highest Performing Restaurants:

```
7
8     -- Select the restaurant with highest numbers of orders
9     select count(order_items.order_id) as 'Highest No. of Orders',  restaurants.Restaurant_name, restaurant_email
10    from
11    restaurants inner join dish
12    on dish.restaurant_id = restaurants.restaurant_id
13    inner join order_items
14    on dish.dish_ID = order_items.dishid
15    group by(restaurants.restaurant_id);
16
17    /*-------------------------------------------------------------------------------------------------------------
18    /*-------------------------------------------------------------------------------------------------------------
19
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Highest No. of Orders	Restaurant_name	restaurant_email
8	McDonalds HK	info@mcdonalds.hk
2	Burger King HK	contact@burgerking.hk

Description: This fetches the top restaurants with the highest no. of order population. Because, our order database was small, we can only see two restaurants being displayed as a response to the query.

```
20    -- Select the restaurant with highest revenue(in dollars)
21    select sum(dish.dish_price * order_items.quantity ) as 'Reveneue' , restaurants.restaurant_name
22    from
23    restaurants inner join dish
24    on dish.restaurant_id  =  restaurants.restaurant_id
25    left join order_items
26    on dish.dish_ID = order_items.dishid
27    group by(restaurants.restaurant_name);
28
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Reveneue	restaurant_name
205.84	McDonalds HK
42.72	Burger King HK
NULL	Dimsum House
NULL	Dominos Pizza HK
NULL	KFC HK
NULL	Pacific Coffee
NULL	Pizza Hut HK
NULL	Starbucks HK
NULL	Subway HK
NULL	Tim Ho Wan
NULL	Tsui Wah Resta...

Description: Functionally similar to the previous one, but this rather suggests top restaurants with the highest revenue.

2. Ratings Reports for Dishes and Drivers:

```
43    -- This it to basically Filter out Worst Rated Dishes so that restaurant might improve their taste
44 •  select ifnull(avg(dish_ratings), 0) as 'Ratings'  , dish_ID, dish_name, restaurant_name  -- non rated dishes are assigned 0 because
45    from order_items
46    left join dish
47    on order_items.dishID = dish.dish_ID
48    inner join restaurants
49    on dish.restaurant_id = restaurants.restaurant_id
50    group by(dish_ID) ,(dish_name) , (restaurant_name)
51    order by(avg(dish_ratings)) asc limit 3 ;
52
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Ratings	dish_ID	dish_name	restaurant_name
0.0000	6	Double Cheeseburger	McDonalds HK
2.0000	9	Whopper	Burger King HK
3.2000	1	Big Mac	McDonalds HK

Description: Filters out worst-rated dishes. This helps in businesses to know more about the taste they are serving to the marketplace.

```
65    -- This is to show lowest rated driver
66 •   select driver_rating, fname, lname
67    from drivers
68    where exists ( select driver_id from orders where drivers.driver_id = orders.driver_id) /
69    order by(driver_rating) asc;
70
71
72
73
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

driver_rating	fname	lname
0	Li	Ming
3.5	Chan	Tak-Wai
3.5	Wong	Chun-Ho
5	Cheung	Ho-Kwan

Description: This query filters out the worst-rated drivers. Because our database stores the default rating value as 0, this query helps filter out drivers with 0 ratings who have not yet delivered any orders(i.e., received any ratings).

Apart from these, we also designed some simple queries to geographically analyse the order trends too.*(in file: business_analysis_project.sql)*

Conclusion:

Therefore, by creating a practical and real-life simulating dummy database from openAI and Mockaroo, we trialed multiple procedures strictly abiding by the database design we obtained in our ER Model. This approach enabled us to optimize our queries while also maintaining data integrity across the relations in the relational database system(RDBMS). This online ordering management system project deemed beneficial for our group as it taught us how to handle integrity constraints like foreign key, and primary key, along with improving the effectiveness of mySQL queries. Finally, we gained solid practical experience in designing a nonredundant and normalized conceptual database schema and implementing it effectively within an RDBMS(MySQL).

Word Count: 1995.

----------END----------

Appendix:

Appendix 1: **Order_Items Table:**

order_ID	quantity	dishID	menu_ID	Dish_Ratings
9	5	1	1	3
8	1	9	7	2
8	3	13	7	5
7	3	1	2	5
5	1	1	2	2

(We can see the (9,5,1,1,3) data inserted in order_items table using the following add_dishestoOrders procedure below:)

```
create procedure add_dishesToOrders( in order_ID int , in dish_ID int ,
in menu_ID int , in quantity int , in ratings int , in done boolean)
Begin
declare dish_price float;  declare curr_amt float; declare discount float ;

if(done = false) -- only do if order is not done yet, or until the user hit done button in  UI
then
insert into order_items(order_ID, quantity, menu_ID, dishID, dish_ratings) values
(order_ID , quantity, menu_ID , dish_ID , ratings) ;

select d.Dish_Price into dish_price from  dish d where d.dish_ID = dish_ID limit 1 ;

Select order_amount INTO curr_amt
FROM orders
WHERE orders.order_ID = order_ID
limit 1;

IF curr_amt IS NULL THEN
SET curr_amt = 0;
END IF;
set curr_amt =  curr_amt + (dish_price * quantity) ; /*Discounting acc to the user_Class*/

select discount_rate into discount
from user_class inner join users
on users.class = user_class.class
where users.user_id = (select user_id from orders where orders.order_ID = order_id ) ;

update orders
set order_amount =  curr_amt - discount/100 * curr_amt  -- /*alternative: COALESCE(order_amount, 0)*/
where orders.order_ID = order_ID ;
END IF ;
END //
```

Appendix 2:

order_ID	user_ID	Order_Time	account_number	driver_id	Payment_Method	order_amount	done	driver_rating
1	1	2024-12-06 13:08:03	4111111111111111	1	MASTERCARD	100	1	5
3	1	2024-12-06 13:08:03	4333444433334444	3	MASTERCARD	150	1	5
5	6	2024-12-06 18:53:21	5222333344445555	1	MASTERCARD	34.97	1	2

```
delimiter //
CREATE PROCEDURE get_sorted_payment_order(
    in payment ENUM ('MASTERCARD' , 'OCTOPUS' , 'ALIPAY')
)
BEGIN
    select * from orders where payment_method=payment;
END //
delimiter ;
```

Appendix 3:

```
delimiter //
create procedure marker_order_done(
in order_ID int
)
Begin
update orders
set done = true
where orders.order_ID = order_ID ;

END //
Delimiter ;
```

Appendix 4:

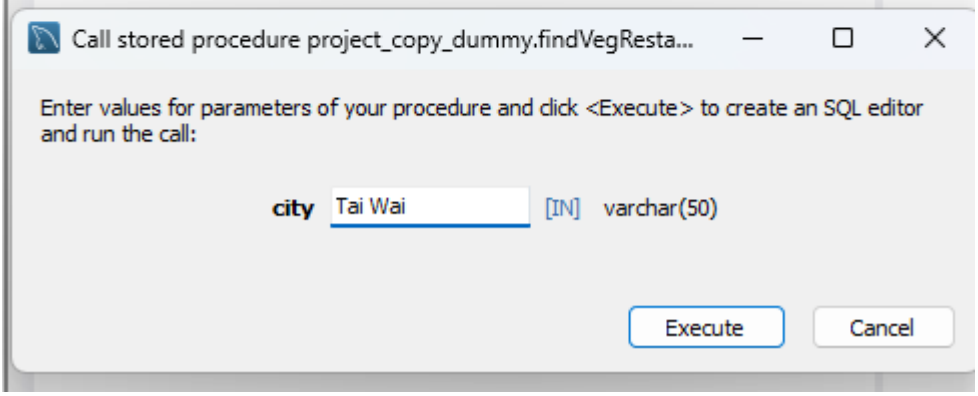
```
delimiter //
create procedure rate_driver(
in order_id int ,
in rating float(10)
)
begin

if ((select done from orders where orders.order_id = order_ID) = true)
then
update orders
set orders.driver_rating  = rating
where orders.order_id = order_ID ;
call update_driver_rating( @order_id);
end if ;

end //
delimiter ;
```

```
delimiter //
CREATE PROCEDURE update_driver_rating(
in orderid int
)
begin
 declare new_rating float ;
 declare d_ID int;

 select driver_ID into d_ID
 from orders
 where orders.order_ID = orderid ;

select avg(driver_rating)  into new_rating
from orders
where  orders.done is true
group by (driver_id) having driver_id =d_ID ;

update drivers
set drivers.driver_rating = new_rating
where drivers.Driver_ID = d_ID;
end //
delimiter ;
```

Appendix 5:

a. Procedure and call: Find Veg Restaurant →

```
154  CREATE PROCEDURE `findVegRestaurant`(
155  in city varchar(50)
156  )
157  Begin
158  Select restaurant_name , dish_name
159  from restaurants inner join dish
160  on restaurants.restaurant_id = dish.restaurant_id
161  inner join branch on branch.restaurant_id = dish.restaurant
162  where Vegetarian = true
163  and branch_address like concat('%' , city , '%') ;
164  End //
165  Delimiter ;
```



Result:

	restaurant_name	dish_name
▶	McDonalds HK	Fries
	McDonalds HK	McVeggie Burger
	McDonalds HK	Apple Pie
	McDonalds HK	Sundae

b. Procedure and call : Sort by Price →

```
123  Delimiter //
124  Create procedure SortbyPrice (
125  in city VARCHAR(50),
126  in restaurant VARCHAR(50)
127  )
128  Begin
129  Select distinct dish_name , dish_price
130  from dish inner join
131  menu_dishes on dish.Dish_ID  = menu_dishes.Dish_ID
132  where dish.restaurant_id  = (select restaurant_id from restaurants where restaurant_name like concat('%', restaurant, '%
133  AND   menu_dishes.MenuID in (
134                               select MenuID from menu
135                               where menu.Branch_ID =
136                                  (select branch_id from branch
137                                   where branch.branch_address like concat('%' , city , '%')
138                                  )
139                              )
140  order by  Dish_Price desc;
141
142  END //
143  Delimiter ;
144
145  -- CALL 5
146  call SortbyPrice('Tai Wai', 'McDonalds');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

dish_name	dish_price
McVeggie Burger	20.00
Double Cheeseburger	15.50
Filet-O-Fish	12.99
Big Mac	10.99
Fries	8.50
Sundae	7.75

6. ER Model draw.io collab link: (Note: Open with drawIO)

https://drive.google.com/file/d/19HdJ_zNo3_vRq3C7tobB7wCr2_ZKOX7n/view?usp=sharing