University of British Columbia, Department of Computer Science

CPSC 304

Cover Page for Project Part __Milestone2____

Date: _____2020.10.23_____

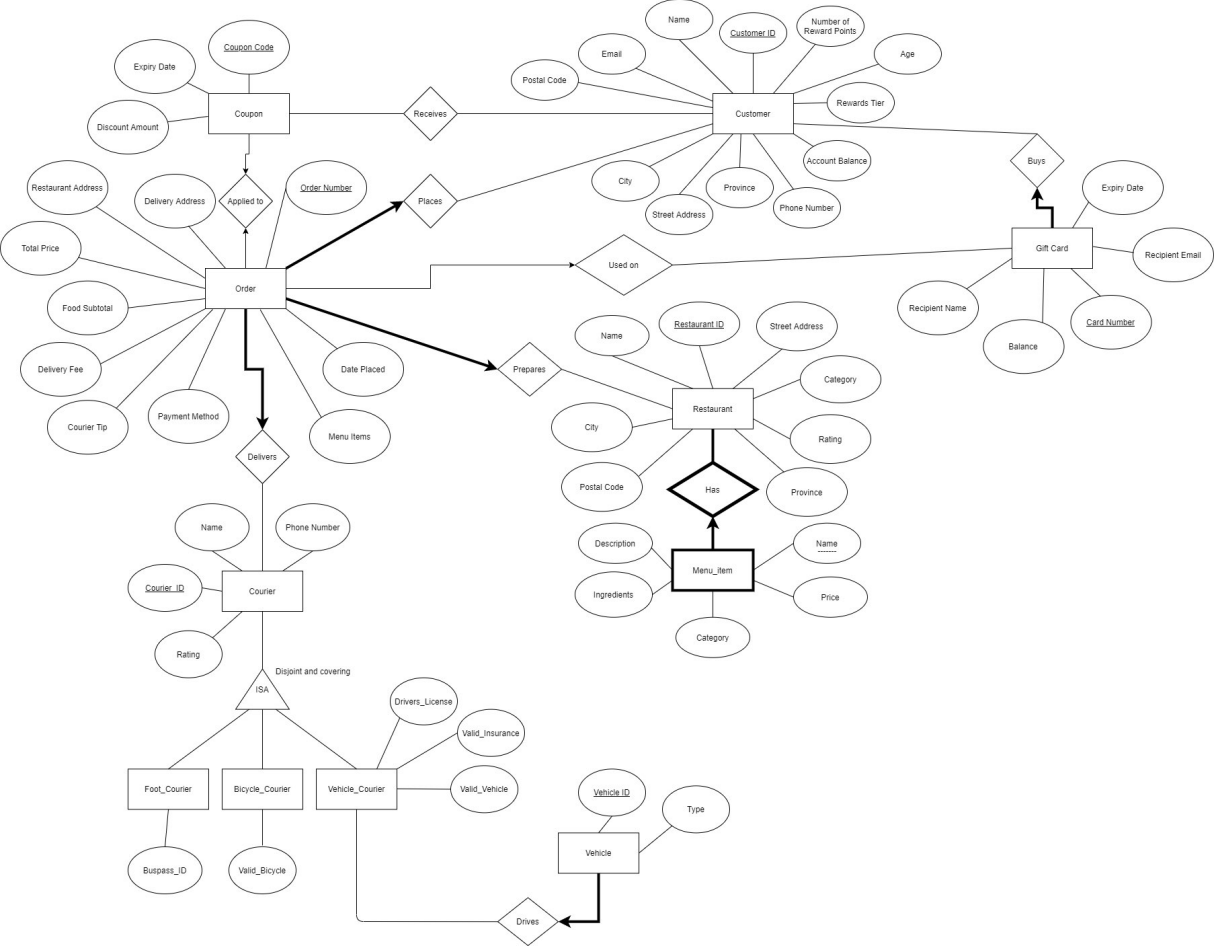
Project Group Number on Canvas: ___86___

Group Members:

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
XinYue Wang	32203168	b1k2b	qianxunxun@outlook.com
Jennifer Chan	39733985	l5h1b	jan_ckw@hotmail.com
Victor Cheng	13090618	c2p6w	victorcheng0425@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


Coupon Code
Expiry Date
Discount Amount
Coupon
Receives
Name
Customer ID
Number of Reward Points
Email
Age
Postal Code
Customer
Rewards Tier
Account Balance
City
Province
Street Address
Phone Number
Buys
Expiry Date
Gift Card
Recipient Email
Recipient Name
Balance
Card Number
Restaurant Address
Delivery Address
Applied to
Order Number
Places
Total Price
Order
Used on
Food Subtotal
Delivery Fee
Courier Tip
Payment Method
Date Placed
Menu Items
Delivers
Prepares
Name
Restaurant ID
Street Address
Category
City
Restaurant
Rating
Postal Code
Has
Province
Description
Name
Ingredients
Menu_item
Price
Category
Name
Phone Number
Courier_ID
Courier
Rating
Disjoint and covering
ISA
Drivers_License
Valid_Insurance
Foot_Courier
Bicycle_Courier
Vehicle_Courier
Valid_Vehicle
Vehicle ID
Type
Vehicle
Buspass_ID
Valid_Bicycle
Drives

Milestone 2 – Group 86

XinYue Wang
Jennifer Chan
Victor Cheng

Relation schemas:

Coupon(*coupon_code*: string, *expiry_date*: string, *discount_amount*: real)

- Primary key: coupon_code.
- Candidate keys: None.
- Foreign keys: None.

Order(*order_number*: integer, *delivery_address*: string, *restaurant_address*: string, *total_price*: real, *food_subtotal*: real, *delivery_fee*: real, *courier_tip*: real, *payment_method*: string, *menu_items*: string, *date_placed*: string, *customer_ID*: integer, *restaurant_ID*: integer, *card_number*: integer, *courier_ID*: integer, *coupon_code*: string)

- Primary key: order_number.
- Candidate keys: None.
- Foreign key customer_ID references CUSTOMER. (Represents the one-to-many relationship Places from ER diagram.)
- Foreign key restaurant_ID references RESTAURANT. (Represents the one-to-many relationship Prepares from ER diagram.)
- Foreign key card_number references GIFT_CARD. (Represents the one-to-many relationship Used_On from ER diagram.)
- Foreign key courier_ID references COURIER. (Represents the one-to-many relationship DELIVERS from ER diagram.)
- Foreign key coupon_code references COUPON. (Represents the one-to-one relationship Applied_To from ER diagram.)

Courier(*courier_ID*: integer, *name*: string, *rating*: real, *phone_number*: integer)

- Primary key: courier_ID.
- Candidate keys: None.
- Foreign keys: None.

Vehicle_Courier(*courier_ID*: integer, *valid_insurance*: string, *drivers_license*: string, *valid_vehicle*: string)

- Primary key: courier_ID.
- Candidate keys: (drivers_license).
- Foreign key courier_ID references COURIER.
- Note: *valid_insurance* is a field that represents whether the courier has insurance for their vehicle; the value is the string “yes” or “no.”

- Note: *valid_vehicle* is a field that represents whether the courier's vehicle meets the industry safety requirements; the value is the string "yes" or "no."

Bicycle_Courier(*courier_ID*: integer, *valid_bicycle*: string)

- Primary key: courier_ID.
- Candidate keys: None.
- Foreign key courier_ID references COURIER.
- Note: *valid_bicycle* is a field that represents whether the courier's bicycle meets the industry safety requirements; the value is the string "yes" or "no."

Foot_Courier(*courier_ID*: integer, *buspass_ID*: integer)

- Primary key: courier_ID.
- Candidate keys: (buspass_ID).
- Foreign key courier_ID references COURIER.
- Note: Foot Couriers may optionally register a *buspass_ID* with the app if they wish to deliver orders by bus.

Vehicle_Drives(*vehicle_ID*: integer, *type*: string, *courier_ID*: integer)

- Primary key: vehicle_ID.
- Candidate keys: None.
- Foreign key courier_ID references VEHICLE_COURIER. (References the one-to-many relationship DRIVES from ER diagram.)

Customer(*customer_ID*: integer, *email*: string, *name*: string, *age*: integer, *reward_points*: integer, *rewards_tier*: string, *street_address*: string, *city*: string, *province*: string, *phone_number*: integer, *account_balance*: real)

- Primary key: customer_ID.
- Candidate keys: (email).
- Foreign keys: None.

Gift_Card_Buys(*card_number*: integer, *recipient_name*: string, *recipient_email*: string, *balance*: real, *customer_ID*: integer, *expiry_date*: string)

- Primary key: card_number.
- Candidate keys: None.
- Foreign key customer_ID references CUSTOMER. (References the one-to-many relationship BUYS from ER diagram.)

Restaurant(*restaurant_ID*: integer, *name*: string, *street_address*: string, *city*: string, *province*: string, *postal_code*: string, *category*: string, *rating*: real)

- Primary key: restaurant_ID.
- Candidate keys: (street_address, postal_code).
- Foreign keys: None.

Menu_Items_Has(*name*: string, *restaurant_ID*: integer, *description*: string, *price*: real, *ingredients*: string, *category*: string)

- Primary key: (name, restaurant_ID).
- Candidate keys: None.
- Foreign key restaurant_ID references RESTAURANT. (References the one-to-many relationship HAS from ER diagram.)

Receives(*customer_ID*: integer, *coupon_code*: integer)

- Primary key: (customer_ID, coupon_code).
- Candidate keys: None.
- Foreign key customer_ID references CUSTOMER.
- Foreign key coupon_code references COUPON.

Functional Dependencies:

Coupon: (BCNF)

coupon_code -> expiry_date, discount_amount

Order: (2NF)

order_number -> delivery_address, total_price, payment_method, menu_items, date_placed, customer_ID, restaurant_ID, card_number, courier_ID, coupon_code, delivery_fee, courier_tips, food_subtotal

delivery_fee, courier_tips, food_subtotal -> total_price

Courier*:* (BCNF)

courier_ID -> name, rating, phone_number

Vehicle_Courier: (BCNF)

courier_ID -> drivers_license, valid_insurance, valid_vehicle

drivers_license -> valid_insurance, valid_vehicle, courier_ID

Bicycle_Courier: (BCNF)

courier_ID -> valid_bicycle

Foot_Courier: (BCNF)

courier_ID -> buspass_ID

buspass_ID -> courier_ID

Vehicle_Drives: (BCNF)

courier_ID -> vehicle_ID, type

vehicle_ID -> type

Customer: (2NF)

customer_ID -> email, name, age, reward_points, rewards_tier, street_address, city, province, phone_number, account_balance, postal_code

postal_code -> city, province

email -> customer_ID, name, age, reward_points, rewards_tier, street_address, city, province, phone_number, account_balance, postal_code

Gift_Card_Buys: (BCNF)

card_number -> recipient_name, recipient_email, balance, expiry_date, customer_ID

Restaurant: (2NF)

restaurant_ID -> name, category, rating, street_address, postal_code, city, province

street_address, postal_code -> name, restaurant_ID, category, rating, city, province

postal_code -> city, province

Menu_Items_Has: (BCNF)

name, restaurant_ID-> description, price, ingredients, category

Receives: (BCNF)

customer_id, coupon_code -> expiry_date

Normalization:

Before Normalization:

Order: (2NF)

order_number -> delivery_address, total_price, payment_method, menu_items, date_placed, customer_ID, restaurant_ID, card_number, courier_ID, coupon_code, delivery_fee, courier_tips, food_subtotal, restaurant_address

delivery_fee, courier_tips, food_subtotal -> total_price

After Normalization:

Decompose to:

Order_fee (delivery_fee, food_subtotal, courier_tips, total_price)

- *Primary Keys:* (delivery_fee, food_subtotal, courier_tips)
- *Foreign Keys: None*
- *Candidate Keys: None*

Order (order_number, delivery_address, restaurant_address, payment_method, menu_items, date_placed, customer_ID, restaurant_ID, card_number, courier_ID, coupon_code, delivery_fee, courier_tips, food_subtotal)

- *Primary Keys: order_number*
- *Foreign Keys: customer_id references Customer; restaurant_id references Restaurant; courier_ID references to Courier; card_number references to Giftcard_Buys; coupon_code references to Coupon.*
- *Candidate Keys: None*

Before Normalization:

Restaurant: (2NF)

restaurant_ID -> name, category, rating, street_address, postal_code, city, province

street_address, postal_code -> name, restaurant_ID

postal_code -> city, province

After Normalization:

Decompose to:

Address(Postal Code, City, Province)

- *Primary Keys: Postal Code*
- *Foreign Keys: None*
- *Candidate Keys: None*

Restaurant(restaurant_ID, name, category, rating, street_address, postal_code)

- *Primary Keys: restaurant_ID*
- *Foreign Keys: postal_code* references Address
- *Candidate Keys: (street_address, postal_code)*

Before Normalization:

Customer: (2NF)

customer_id -> email, name, age, reward_points, rewards_tier, street_address, city, province, phone_number, account_balance, postal_code

postal_code -> city, province

email -> customer_id, name, age, reward_points, rewards_tier, street_address, city, province, phone_number, account_balance, postal_code

After Normalization:

Address(Postal Code, City, Province)

- *Primary Keys: Postal Code*

- *Foreign Keys: None*
- *Candidate Keys: None*

Customer (customer_id, email, name, age, reward_points, rewards_tier, street_address, phone_number, account_balance, postal_code)

- *Primary Key: customer_ID*
- *Candidate Keys: email*
- *Foreign Keys: postal_code references to Address*

SQL DDL:

```
CREATE TABLE Customer(
    customer_id INTEGER,
    email CHAR(40),
    age INTEGER,
    phone_number INTEGER,
    street_address CHAR(100),
    postal_code CHAR(10),
    name CHAR(30),
    reward_points INTEGER,
    rewards_tier CHAR(20),
    account_balance REAL,
    PRIMARY KEY(customer_id),
    FOREIGN KEY (postal_code) REFERENCES Address
)

CREATE TABLE Address (
    postal_code CHAR(10),
    city CHAR(20),
    province CHAR(20),
    PRIMARY KEY (postal_code)
)

CREATE TABLE Giftcard_Buys(
    customer_id INTEGER,
    card_number INTEGER,
    recipient_email CHAR(40),
    recipient_name CHAR(30),
    expiry_date CHAR(20),
    balance REAL,
    PRIMARY KEY (card_number),
    FOREIGN KEY (customer_id) REFERENCES Customer
)
```

```
CREATE TABLE Coupon(
	coupon_code CHAR(10),
	expire_date CHAR(20),
	discount_amount REAL,
	PRIMARY KEY (coupon_code)
)


CREATE TABLE Receives(
	customer_id INTEGER,
	coupon_code CHAR(20),
	PRIMARY KEY (customer_id, coupon_code),
	FOREIGN KEY (customer_id) REFERENCES Customer,
	FOREIGN KEY (coupon_code) REFERENCES Coupon
)


CREATE TABLE Order (
	delivery_address CHAR(40),
	restaurant_address CHAR(40),
	food_subtotal REAL,
	delivery_fee REAL,
	courier_tips REAL,
	payment_method CHAR(20),
	menu_items CHAR(500),
	date_placed CHAR(20),
	order_number INTEGER,
	customer_id INTEGER,
	courier_id INTEGER,
	restaurant_id INTEGER,
	card_number INTEGER,
	coupon_code CHAR(10),
	PRIMARY KEY (order_number),
	FOREIGN KEY (customer_id) REFERENCES Customer,
	FOREIGN KEY (courier_id) REFERENCES Courier,
	FOREIGN KEY (restaurant_id) REFERENCES Restaurant
	FOREIGN KEY (card_number) REFERENCES Giftcard_Buys,
	FOREIGN KEY (coupon_code) REFERENCES Coupon
)
```

```
CREATE TABLE Order_fee (
	delivery_fee REAL,
	food_subtotal REAL,
	courier_tips RREAL,
	total_price REAL,
	PRIMARY KEY (delivery_fee, food_subtotal, courier_tips)
)


CREATE TABLE Restaurant(
	restaurant_id INTEGER,
	postal_code CHAR(10),
	name CHAR(20),
	category CHAR(20),
	rating REAL,
	street_address CHAR(100),
	PRIMARY KEY (restaurant_id),
	UNIQUE (street_address, postal_code),
	FOREIGN KEY (postal_code) REFERENCES Address
)

CREATE TABLE Courier (
	courier_id INTEGER,
	name CHAR(20),
	rating REAL,
	phone_number INTEGER
	PRIMARY KEY (courier_id)
)

CREATE TABLE Vehicle_Courier(
	valid_vehicle   CHAR(20),
	valid_insurance CHAR(20),
	drivers_license CHAR(20),
	courier_id INTEGER,
	PRIMARY KEY (courier_id),
	UNIQUE (drivers_license),
	FOREIGN KEY (courier_id) REFERENCES Courier
)
```

```
CREATE TABLE Bicycle_Courier(
	courier_id INTEGER,
	valid_bicycle CHAR(20),
	PRIMARY KEY (courier_id),
	FOREIGN KEY (courier_id) REFERENCES Courier
)

CREATE TABLE Foot_Courier(
	courier_id INTEGER,
	bus_pass INTEGER,
	PRIMARY KEY (courier_id),
	UNIQUE (bus_pass),
	FOREIGN KEY (courier_id) REFERENCES Courier
)

CREATE TABLE Vehicle_drives (
	vehicle_id INTEGER,
	type CHAR(20),
	courier_id INTEGER,
	PRIMARY KEY (vehicle_id),
	FOREIGN KEY (courier_id) REFERENCES Vehicle_Courier
)



CREATE TABLE Menu_Items_Has(
	name CHAR(20),
	description CHAR(100),
	ingredient CHAR(100),
	category CHAR(20),
	price REAL,
	restaurant_id INTEGER,
	PRIMARY KEY (restaurant_id, name),
	FOREIGN KEY (restaurant_id) REFERENCES Restaurant
)
```

Customer:

customer_id	email	age	phone_number	street_address	postal_code	name	reward_points	reward_tier	account_balance
100001	abcde@gmail.com	17	7781112223	2788 90th Avenue	N2H 5A5	Alice	400	VIB	11
100002	adfe@hotmail.com	21	7781112224	3521 Algonquin Blvd #400	H2A 2Z3	Jack	401	VIB	111
100003	hamm0nd@gmail.com	33	7781112225	57 49th Avenue	T0H 1N0	Waston	0	Insider	11
100004	tracer@outlook.com	19	7781112226	1441 5th Avenue	S4P 3Y2	Tracer	1001	Rouge	32
100005	dontawe@gmail.com	45	7781112227	4727 Ste. Catherine Ouest	G6P 5V8	Ark	1	Insider	103

Address:

postal_code	city	province
N2H 5A5	Kitchener	Ontario
H2A 2Z3	Montreal	Quebec
T0H 1N0	John D'or Prairie	Alberta
S4P 3Y2	Wollaston Lake	Saskatchewan
G6P 5V8	Arthabaska	Quebec

Giftcard_Buys:

customer_id	card_number	recipient_email	recipient_name	expiry_date	balance
100001	1234567890	abcde@gmail.com	Alice	20201130	100
100002	1234567891	adfe@hotmail.com	Jack	20201130	200
100003	1234567892	hamm0nd@gmail.com	Waston	20201130	300
100004	1234567893	tracer@outlook.com	Tracer	20201130	400
100005	1234567894	dontawe@gmail.com	Ark	20201130	500

Coupon:

coupon_code	expire_date	discount_amount
20203	20201230	50
20204	20211230	10
20205	20201130	5
20206	20210130	55
20207	20210101	100

Receives:

customer_id	coupon_code
100001	20203
100002	20204
100003	20205
100004	20206
100005	20207

Restaurant:

restaurant_id	postal_code	name	category	street_address	rating
200001	N2H 5A5	Qilin	Asian	2788 90th Avenue	5
200002	H2A 2Z3	McBur	Fast food	3521 Algonquin Blvd #400	4.45
200003	T0H 1N0	Tea House	Asian	57 49th Avenue	3.99
200004	S4P 3Y2	GrilledCk	Fast food	1441 5th Avenue	3.99
200005	G6P 5V8	Passione	Italian	4727 Ste. Catherine Ouest	4.5

Courier

courier_id	name	rating	phone_number
30001	Will	5	7661112223
30002	Mick	4.45	7661112224
30003	David	3.99	7661112225
30004	Cart	3.99	7661112226
30005	Meg	4.5	7661112227

Vehicle_Courier:

courier_id	driver_license	valid_insurance	valid_vehicle
30001	772910239	yes	yes
30002	772910238	yes	yes
30003	772910237	no	yes
30004	772910236	yes	yes
30005	772910235	yes	yes

Foot_Courier:

courier_id	bus_pass
30001	1928311238
30002	1928311237
30003	1928311236
30004	1928311232
30005	1928311231

Bicycle_Courier:

courier_id	valid_bicycle
30001	yes
30002	yes
30003	yes
30004	yes
30005	no

Vehicle_Drives:

courier_id	type	vehicle_id
30001	SUV	400001
30002	SUV	400002
30003	SUV	400003
30004	Coupe	400004
30005	Coupe	400005

Order

order_number	customer_id	courier_id	restaurant_id	coupon_code	card_number	delivery_fee	courier_tips	food_subtotal
500001	100001	30001	200001	20203	1234567890	1.99	1.99	19.99
500002	100002	30002	200002	20204	1234567891	2.99	3.99	33.2
500003	100003	30003	200003	20205	1234567892	3.99	4.99	20
500004	100004	30004	200004	20206	1234567893	2.99	3.99	11
500005	100005	30005	200005	20207	1234567894	2.99	3.99	5.99

Order con'd:

delivery_address	restaurant_address	payment_method	menu_items	date_placed
2788 90th Avenue	4727 Ste. Catherine Ouest	account balance	coffee	20191011
3521 Algonquin Blvd #400	1441 5th Avenue	account balance	coffee	20201011
57 49th Avenue	3521 Algonquin Blvd #400	account balance	burger	20200111
1441 5th Avenue	57 49th Avenue	account balance	fries	20201111
4727 Ste. Catherine Ouest	2788 90th Avenue	account balance	noodles	20191011

Order_fee:

delivery_fee	courier_tips	food_subtotal	total_price
1.99	1.99	19.99	23.97
2.99	3.99	33.2	40.18
3.99	4.99	20	28.98
2.99	3.99	11	17.98
2.99	3.99	5.99	12.97

Menu_Items_Has:

restaurant_id	name	description	ingredient	category	price
200001	coffee	Just order it	milk	beverages	3.99
200002	coffee	Just order it	milk	beverages	3.99
200003	burger	Just order it	chicken	dinner	8.99
200004	fries	Just order it	potato	dinner	3.99
200005	noodles	Just order it	flour	dinner	6.99

List of Proposed Queries:

1. **Insertion**: Register a customer's information to the app (i.e. create a new customer account).
    - The name, email, street address, postal code, city, province, and phone number fields are provided by the customer.
    - A unique customer_ID is generated by the system and assigned to the customer.
    - The customer's number of reward points and account balance is initialized to 0, and they start at the Basic level reward tier.

2. **Deletion**: Delete a customer's information from the app (i.e. delete a customer account).
    - Upon deleting a customer's information, all table rows that refer to that customer's *customer_ID* are to be deleted as well (cascade-on-delete). For example, all tuples in the Order table that have a *customer_ID* that matches that of the deleted customer will be deleted from the database as well.

3. **Update**: Update any of the following information fields of a customer: email, name, street address, postal code, city, province, phone number, age.
    - No other tables need to be modified upon the update of the aforementioned customer information fields. For example, any orders placed by a customer prior to the update of that customer's information will not need to be modified.

4. **Selection**: Find all restaurants that have a rating greater than a specified number (where 0.0 is the minimum rating a restaurant can have and 10.0 is the maximum).
    - For example, the user can find all restaurants that have a rating greater than 7.5.
    - Example SQL code:
      SELECT restaurant_ID
      FROM Restaurant
      WHERE rating > 7.5;

5. **Projection**: The user may specify a set of 3-5 fields from the Order relation that they want the query to return. For example, "Find the date placed, total price, and payment method for all orders." Any field not specified will not be returned by the query.
    - Example SQL code:
      SELECT date_placed, total_price, payment_method
      FROM Order
    - The above query will return a relation that contains only the date_placed, total_price, and payment_method fields of the Order

relation. Such a query would be useful for answering questions such as, “Do customers spend more money on certain days of the week?” and “Which payment methods are currently the most popular?”

6. **Join**: Join the Order, Customer, and Restaurant tables to find information about the spending habits and restaurant preferences of customers of a specific rewards tier.
    - Note: The rewards_tier field can contain one of 4 values: ‘Basic’, ‘Silver’, ‘Gold’, and ‘Platinum’. ‘Basic’ represents the lowest reward tier, and it is the reward tier that customers start with when they first register to the app. Customers achieve higher reward tiers after accumulating enough reward points (which is done by placing orders through the app).
    - This query can allow the user to see if
        1. There is any correlation between a customer’s reward tier and the amount they spend on orders, and
        2. If there is any correlation between a customer’s reward tier and the restaurants they like to order from.
    - For example, this query can be used to answer questions such as,
        1. “Do customers in the Platinum reward tier tend to place more expensive orders?”
        2. “What category (e.g. Fast Food) of restaurants do Platinum customers like to order from?”
        3. “Do Platinum customers tend to order from restaurants with higher ratings?”
    - Example SQL code:

```
SELECT order_ID, total_price, customer_ID, restaurant_ID, rating
FROM Customer JOIN Order
        ON Customer.customer_ID = Order.customer_ID
        JOIN Restaurant
        ON Restaurant.restaurant_ID = Order.restaurant_ID
WHERE rewards_tier = ‘platinum’;
```

7. **Division**: Find all customers who have ordered from all restaurants that are located in a specific city.
    - Example query: “Find all customers who have ordered from all restaurants in Vancouver, BC.”

8. **Aggregation with Group By**: Find the average rating of restaurants, where the restaurants are grouped by city.
    - Example SQL code:

```
SELECT city, AVG(rating)
FROM Restaurant
```

GROUP BY city, province;

9. **Aggregation with Having**: Find all couriers who have delivered at least X orders, where X is a number specified by the user.
    - Example SQL code (where X = 10):

```
SELECT courier_ID, COUNT(order_ID)
FROM Order
GROUP BY courier_ID
HAVING COUNT(order_ID) > 10;
```

10. **Nested Aggregation with Group By**: Find the restaurant that has the largest average order price.
    - Note: An "order price" refers to the "total_price" field in the Order relation.
    - For example, suppose we have the following Order relation (with some fields omitted for simplicity):

Order_ID	Restaurant_ID	Total_Price	Date_Placed
00001	12345	21.30	2020/10/20
00002	67890	40.90	2020/10/20
00003	12345	12.30	2020/10/21
00004	12345	33.51	2020/10/21
00005	67890	28.80	2020/10/22
00006	21385	98.21	2020/10/22
00007	21385	100.10	2020/10/22

In this case, the restaurant with restaurant_ID 12345 has an average order price of (21.30 + 12.30 + 33.51) / 3 = 22.37. The restaurant with restaurant_ID 67890 has an average order price of (40.90 + 28.80) / 2 = 34.85. And finally, the restaurant with restaurant_ID 21385 has an average order price of (98.21 + 100.10) / 2 = 99.16. Therefore, the restaurant with the *largest* average order price is the one with restaurant_ID 21385.

- Example SQL code:

```
SELECT restaurant_ID, avg_order_price
FROM (SELECT restaurant_ID, AVG(total_price) AS avg_order_price
   FROM Order
   GROUP BY restaurant_ID)
WHERE avg_order_price =
        (SELECT MAX(avg_order_price)
    FROM (SELECT restaurant_ID, AVG(total_price) as avg_order_price
                FROM Order
                GROUP BY restaurant_ID));
```