VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

DATABASE SYSTEMS (CO2013)

Assignment 1

Food Ordering System

Instructor(s):
Nguyễn Thị Ái Thảo, CSE-HCMUT

Team name:
Class: CCO2 - Group: MLN - Semester 241

Student(s):
Trịnh Anh Minh - 2252493
Cao Ngọc Lâm - 2252419
Hồ Khánh Nam - 2252500
Đặng Ngọc Phú - 2252617
Nguyễn Châu Hoàng Long - 2252444

HO CHI MINH CITY, OCTOBER 2024

Contents

1 Member list & workload

No.	Full name	Student ID	Assignment	Effort
1	Trịnh Anh Minh	2252493	Draw E(ERD) Model, Latex editor	100%
2	Cao Ngọc Lâm	2252419	Writing query for relational algebra operators Latex editor	100%
3	Hồ Khánh Nam	2252500	Mapping (E)ERD diagram, Latex Editor	100%
4	Đặng Ngọc Phú	2252617	Writing constraints not shown in (E)ER diagram Latex editor	100%
5	Nguyễn Châu Hoàng Long	2252444	Mapping (E)ERD diagram	100%

2 Introduction

In today's digital age, food ordering systems have become essential for restaurants, cafes, and delivery services to streamline their operations and enhance customer experience. A well-organized database for managing orders, clients, menus, and delivery logistics is the foundation of any such system.

At the core of any successful food ordering system is a well-structured and optimized database. The database acts as a central repository for all system-related information, acting as a bridge between the front-end, back-end operations, and external services such as payment gateways and real-time order tracking. A comprehensive and efficient database design is crucial for the smooth functioning of the entire food ordering system.

Therefore, our group chooses the topic of building a food ordering system database to process and manage large volumes of data, ensuring the system operates effectively when there is high demand, such as during peak hours or promotional campaigns. The ability to store, retrieve and update data quickly and accurately is essential for seamless order processing and improved customer satisfaction.

- These are our business description:

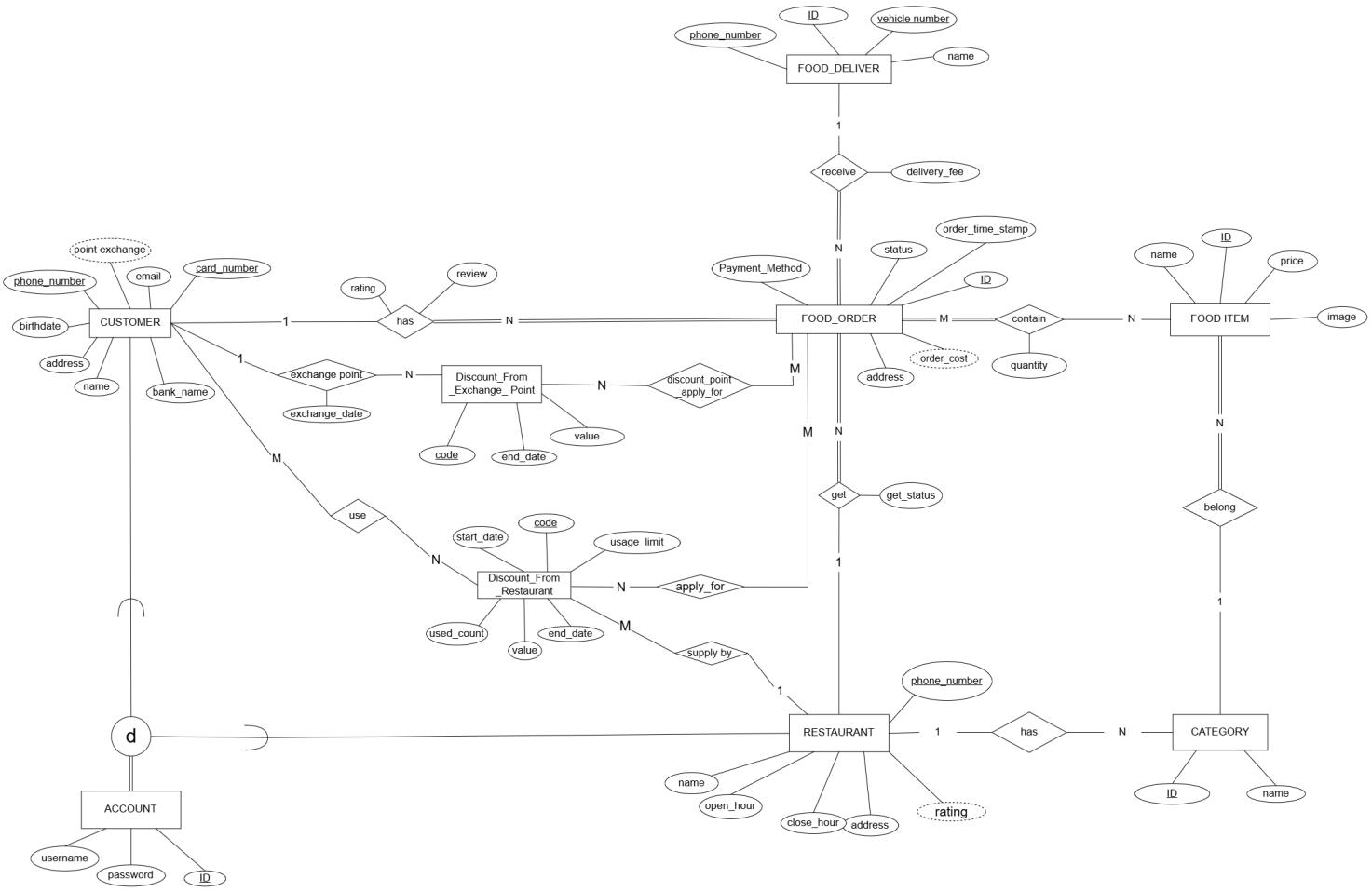
1. **Restaurants** contact the system to register relevant information such as: location, menu, dishes, prices, discounts, etc. (Each restaurant defines their own categories.)
2. **Customers** register an account on the system with basic information: name, phone number, address, email, etc.
3. **To order food**:
   (a) Search for the food you want on the system.
   (b) Select the food from the restaurant you want.
   (c) Choose the quantity, apply discounts (if available), and add notes for the shipper to inform the restaurant.
   (d) Confirm the order information.
   (e) Edit the address to match your delivery location.
   (f) Choose the payment method: cash on delivery, card payment, etc.
   (g) Complete the order.
4. **The food deliver** receives the order information, goes to the restaurant, requests the restaurant to prepare the order, and picks it up.
5. The system displays four statuses: "order received", "food being prepared" when the restaurant get the order and "in transit", and "delivered" for the food deliver.
6. Afterward, the shipper delivers the food to the customer by identified the vehicle number on the system.
7. **The total amount of the bill** will be calculated by:

```
receipt = (price of the food * quantity) + delivery fee - discount (if
                                    any)
```

8. **Exchanging point for discount:** Discount from redeeming points has 5 types (5%, 10%, 15%, 20%)
    - 20 points can be redeemed for 1 type of 5%, only applicable to orders of 50.000 vnd or more, maximum discount of 5.000 vnd
    - 40 points can be redeemed for 1 type of 10%, only applicable to orders of 100.000 vnd or more, maximum discount of 15.000 vnd
    - 80 points can be redeemed for 1 type of 15%, only applicable to orders of 150.000 vnd or more, maximum discount of 30.000 vnd
    - 120 points can be redeemed for 1 type of 20%, only applicable to orders of 250.000 vnd or more, maximum discount of 50.000 vnd
9. **Discounts:** Discount from restaurant: there are 3 types (5.000 vnd, 10.000 vnd, 20.000 vnd). **Total discount applied to 1 order cannot exceed 30% of order value.**
    - Discount 5.000 vnd only applies to bills of at least 50.000 vnd
    - Discount 10.000 vnd only applies to bills of at least 100.000 vnd
    - Discount 20.000 only applies to bills of at least 150.000 vnd
10. If orders over 200.000 vnd, the order will have the shipping fee = 0
11. **Ratings** (if any): If the customer wants to rate the service poorly, they can write a review on the system about the order. Customers can rate the food order 1,2,3,4,5 stars according to satisfaction. If they do not rate, the default will be 5 stars. In here, the food order will receive the rating from the customer and the restaurant will get the rating from all that order. So the restaurant rating will be calculated as the total rating of the orders from that restaurant divided by the average.

```
restaurant rating = (total rating point of food order) / (total  number of
                        food  order restaurant has)
```

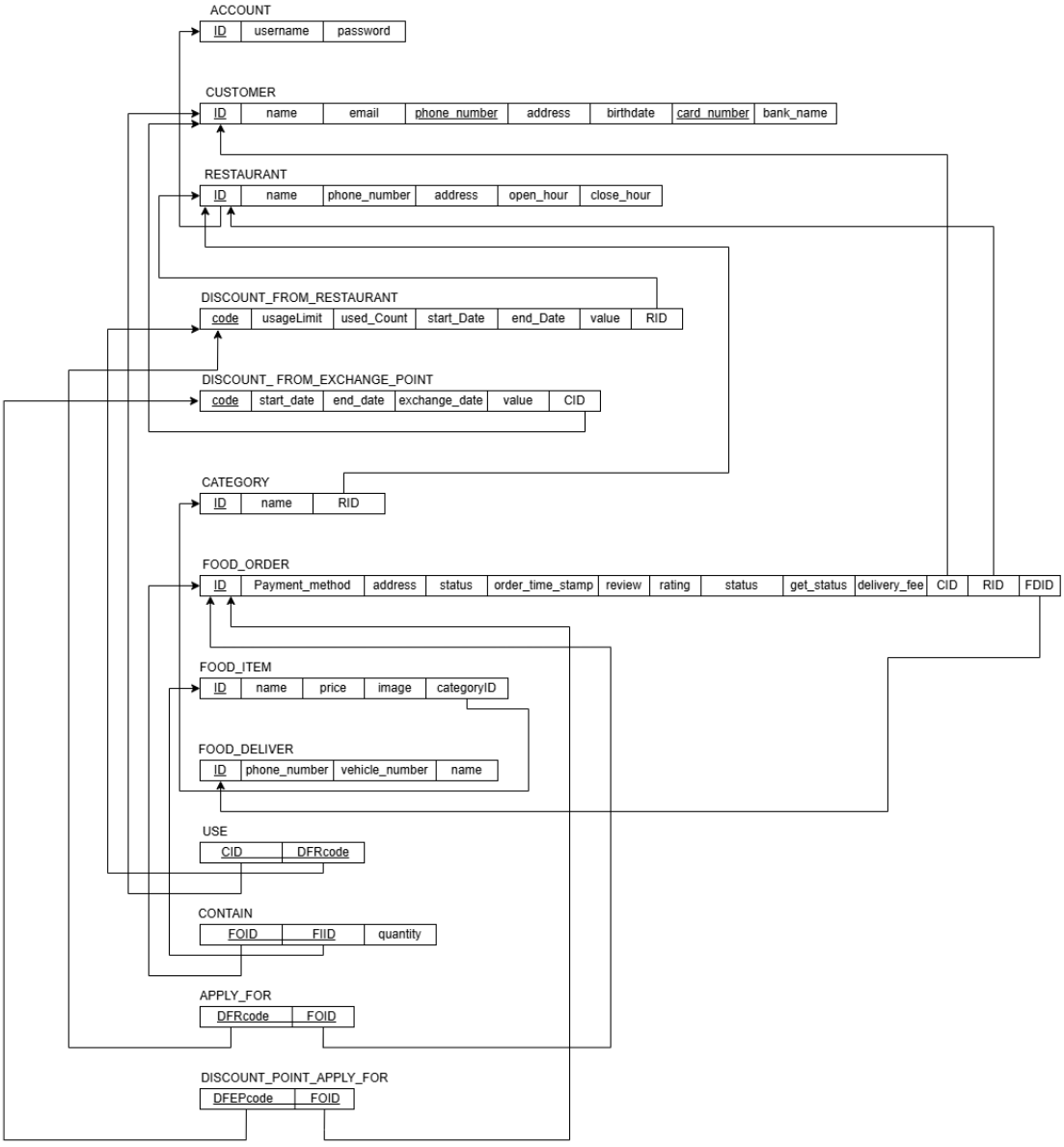
3 (E)ERD Model of Food Ordering System



- The Draw.io link of the (E)ERD model of food ordering system: https://drive.google.com/file/d/1OaxpMPvbzh7tl58hLVJ9TC0xoYaiU3kx/view

4 Mapping (E)ERD diagram to a relational database schema

4.1 Relational database schema



- The Draw.io link of the relational database schema: https://drive.google.com/file/d/1eZbJryRE2e1J_Jor_x2rO6gk4T5Bqkuq/view?usp=sharing

4.2 Details of mapping (E)ERD diagram

From the (E)ER diagram, we can map the relational database schema as follows:

- **ACCOUNT** (ID, username, password)
  Primary Key: ID
  Not Null: username, password
- **CUSTOMER** (ID, name, email, phone_number, address, birthdate, card_number, bank_name)
  Primary Key: ID
  Secondary (unique, not null) Key: phone_number, card_number
  Foreign key: ID to ACCOUNT.ID (SPECIALIZATION)
  Not Null: name, address, birthdate
- **RESTAURANT** (ID, name, phone_number, address, open_hour, close_hour)
  Primary Key: ID
  Secondary (unique, not null): phone_number
  Foreign key: ID to ACCOUNT.ID (SPECIALIZATION)
  Not Null: address
- **DISCOUNT_FROM_RESTAURANT** (code, usageLimit. used_Count, start_date, end_date, value, RID)
  Primary Key: code
  Foreign key: RID to RESTAURANT.ID
  Not Null: percentage, used_Count, start_date, end_date
- **DISCOUNT_FROM_EXCHANGE_POINT** (code, start_date, end_date, exchange_date, value, CID)
  Primary Key: code
  Foreign Key: code to DISCOUNT.code (SPECIALIZATION),CID to CUSTOMER.ID
  Not Null: CID
- **CATEGORY** (ID, name, RID)
  Primary Key: ID
  Foreign key: RID to RESTAURANT.ID
  Not Null: RID
- **FOOD_ORDER** (ID, Payment_method, address, status, order_time_stamp, review, rating, ship_status, get_status, delivery_fee, CID, RID, FDID)
  Primary Key: ID
  Foreign key: CID to CUSTOMER.ID; RID to RESTAURANT.ID; FDID to FOOD_DELIVER.ID
  Not Null: RID, CID, FDID, address, delivery_fee, order_time_stamp
- **FOOD_ITEM** (ID, name, price, image, categoryID)
  Primary Key: ID
  Foreign key: categoryID to CATEGORY.ID
  Not null: categoryID, price
- **FOOD_DELIVER** (ID, phone_number, vehicle_number, name)
  Primary Key: ID
  Secondary (unique, not null) Key: vehicle_number, phone_number
  Not null: name

- **USE** (CID, DFRcode)
  Primary Key: (CID, DFRcode)
  Foreign key: CID to CUSTOMER.ID, DFRcode to DISCOUNT_FROM_RESTAURANT.code
- **CONTAIN** (FOID,FIID, quantity)
  Primary Key: (FOID,FIID)
  Foreign key: FOID to FOOD_ORDER.ID, FIID to FOOD_ITEM.ID
  Not null: quantity
  Constraint: Check total participation of FOOD_ORDER
- **APPLY_FOR** (DFRcode, FOID)
  Primary Key: (DFRcode, FOID)
  Foreign key: DFRcode to DISCOUNT_FROM_RESTAURANT.code, FOID to FOOD_ORDER.ID
- **DISCOUNT_POINT_APPLY_FOR** (DFEPcode, FOID)
  Primary Key: (DFEPcode, FOID)
  Foreign key: Foreign key: DFEPcode to DISCOUNT_FROM_EXCHANGE_POINT.code,
  FOID to FOOD_ORDER.ID

5 Relational algebra operators

Before defining relational algebra operators, we will create a table of aliases for tables with long names to simplify the process of writing algebraic queries.

Table Name	Alias
DISCOUNT_FROM_RESTAURANT	DFR
DISCOUNT_FROM_EXCHANGE_POINT	DFEP
FOOD_ORDER	FO
FOOD_ITEM	FI
FOOD_DELIVER	FD
DISCOUNT_POINT_APPLY_FOR	DPAF
APPLY_FOR	AF

Table 1: Table name alias

5.1 Retrieve all the discounts supplied by a specific restaurant that a specific user has used

Case description: We need to retrieve a list of all the discounts supplied by a specific restaurant with `specific_restaurant_id` that a specific customer with `specific_customer_id` has used.

Query:

$$E1 \leftarrow \sigma_{\text{DFR.RID=specific\_restaurant\_id,USE.CID=specific\_customer\_id}}(\text{USE}) \bowtie_{\text{USE.DFRcode=DFR.code}} (\text{DFR})$$

$$ANS \leftarrow \pi_{\text{code,percentage,used\_date}}(E2)$$

5.2 Retrieve all the discounts made by exchange points used by a specific customer apply for a specific order

Case description: We need to retrieve a list of all the discounts made by exchange points that a specific customer with `specific_customer_id` has used, applied for a specific order with `order_id`

Query:

$$E1 \leftarrow \sigma_{\text{DFEP.CID=specific\_customer\_id}}(\text{DFEP} \bowtie \text{DPAF} \bowtie \text{BO})$$

$$ANS \leftarrow \pi_{\text{code,percentage,used\_date}}(E1)$$

5.3 Get all the foods belonging to a specific category in a restaurant

Case description: We need to retrieve all food items (ID, name, price) that belong to a given category name: `category_name` for a particular restaurant `restaurant_id`. This case occurs when the user views restaurant information on the application.

Query:

$$E1 \leftarrow \sigma_{\text{CATEGORY.name=}\texttt{category\_name}\wedge\text{CATEGORY.RID=}\texttt{restaurant\_id}}(\text{CATEGORY})$$

$$E2 \leftarrow E1 \bowtie_{\text{CATEGORY.ID=FI.categoryID}} (\text{FI})$$

$$ANS \leftarrow \pi_{\text{FI.ID,FI.name,FI.price}}(E2)$$

5.4 Calculate the Total Cost of an Order After Applying Shipping Fee and Discount

Case Description: We need to calculate the total cost of a given order identified by `order_id`, after applying shipping fees and discounts (if the customer chooses to apply all available discounts on that order). This calculation is typically required when the user wants to see the final price of the order.

Query:

To compute the total cost, we break the process down into four steps.

Step 1: Calculate the total cost of all food items in the order:

$$\textbf{FoodCost} \leftarrow \gamma_{FO.ID\rightarrow id,\ \text{SUM(FI.price}\times\text{FI.quantity)}\rightarrow\text{cost}}\Big(\sigma_{FO.ID=\texttt{order\_id}}(FO \bowtie \text{contain} \bowtie FI)\Big)$$

Step 2: Retrieve the shipping fee associated with `order_id`:

$$\textbf{ShippingFee} \leftarrow \pi_{\text{ship\_fee},\ FO.ID\rightarrow id}\Big(\sigma_{FO.ID=\texttt{order\_id}}(FO)\Big)$$

Step 3: Calculate the restaurant discount:

$$\textbf{E1} \leftarrow \sigma_{\text{AF.FOID=}\texttt{order\_id}}(\text{AF} \bowtie \text{DFR})$$

$$\textbf{RD} \leftarrow \gamma_{\text{AF.FOID}\rightarrow\text{id},\ \text{SUM(DFR.value)}\rightarrow\text{rd}}\Big(E1\Big)$$

Step 4: Calculate the exchange-point discount:

$$\textbf{E1} \leftarrow \sigma_{\text{DPAF.FOID=}\texttt{order\_id}}(\text{DPAF} \bowtie \text{DFEP})$$

$$\textbf{EPD} \leftarrow \gamma_{\text{DPAF.FOID}\rightarrow\text{id},\ \text{SUM(DFEP.value)}\rightarrow\text{epd}}\Big(E1\Big)$$

Step 4: Calculate the final cost for the order:

$$\textbf{E1} \leftarrow \text{FoodCost} \bowtie \text{ShippingFee} \bowtie \text{RD} \bowtie \text{EPD}$$

$$\textbf{FinalCost} \leftarrow \pi_{\text{FoodCost.FOID},\ \text{food\_cost}+\text{deliver\_fee}-\min\left(\frac{\text{food\_cost}\times\text{epd}}{100}+\text{rd},\ \text{food\_cost}\times 0.3\right)}(E2)$$

5.5 Get the current information of the deliver that delivering an order

Case description: We need to retrieve the information of the food delivery (name, phone_number, vehicle_number) currently delivering a specific order with id `order_id`. This case happens when the user wants to see the information of the delivery person responsible for delivering his/her order.

Query:

$$E1 \leftarrow \sigma_{\text{FO.ID=}\texttt{order\_id}}((FO) \bowtie_{FO.FDID=FD.ID} (FD))$$

$$ANS \leftarrow \sigma_{FD.ID,FD.name,FD.phone\_number,FD.vehicle\_number}(E1)$$

5.6 Get information on all the discounts that will expire in the next 2 days for a specific customer

Case description: We need to retrieve the information of the discount that will expire in the next 2 days when a customer with `customer_id` visits this restaurant. This case occurs when the system want to notify the user about upcoming expiring discounts.

Query:

$$\textit{RestaurantDiscounts} \leftarrow \sigma_{\text{DFR.end\_date}\geq\texttt{current\_date}\wedge\text{DFR.end\_date}\leq\texttt{current\_date}+2\wedge\text{DFR.RID=}\texttt{restaurant\_id}}(\text{DFR})$$

$$\textit{PointDiscounts} \leftarrow \sigma_{\text{DFEP.end\_date}\geq\texttt{current\_date}\wedge\text{DFEP.end\_date}\leq\texttt{current\_date}+2\wedge\text{DFEP.RID=}\texttt{restaurant\_id}}(\text{DFEP})$$

$$ANS \leftarrow \pi_{\text{code,usageLimit,used\_count,percentage}}(\textit{RestaurantDiscounts} \cup \textit{PointDiscounts})$$

5.7 Retrieve all receipts made by a specific customer with a specific bank within a time range

Case description: In this case, we need to find all receipts made by the specific `cid` with a specific bank_name `name` within a time range `start`, `end`.

Query:

$$\text{E1} \leftarrow \sigma_{\text{CID=cid}\wedge\text{bank\_name=}\texttt{name}\wedge\text{order\_time\_stamp}\geq\text{start}\wedge\text{order\_time\_stamp}\leq\text{end}}(\text{FO} \bowtie \text{CUSTOMER})$$

$$\text{ANS} \leftarrow \pi_{\text{Payment\_method, address, status}}(\text{E1})$$

5.8 Retrieve All Food Items With Prices Above a Specific Amount for a Restaurant

Case description: We need to retrieve all food items (ID, name, price) for a specific restaurant, identified by `restaurant_id`, where the price of each item is above a specified amount, `price_threshold`.

Query:

$$E1 \leftarrow \sigma_{CATEGORY.RID=\texttt{restaurant\_id}}(CATEGORY)$$

$$E2 \leftarrow E1 \bowtie_{CATEGORY.ID=FI.categoryID} (FOOD\_ITEM)$$

$$ANS \leftarrow \sigma_{FI.price\geq\texttt{price\_threshold}}(\pi_{FI.ID,FI.name,FI.price})(E2)$$

5.9 Retrieve All Food Items Purchased by a Customer Within a Specific Date Range

Case Description: We need to retrieve all food items (ID, name, quantity) purchased by a specific customer, `customer_id`, within a given date range [`start_date`, `end_date`].

Query:

$$E1 \leftarrow \sigma_{\text{FO.CID=}\texttt{customer\_id}\wedge\text{FO.order\_time}\geq\texttt{start\_date}\wedge\text{FO.order\_time}\leq\texttt{end\_date}}(\text{FO})$$

$$E2 \leftarrow E1 \bowtie_{\text{FO.ID=CONTAIN.FOID}} (\text{contain} \bowtie \text{FI})$$

$$ANS \leftarrow \pi_{\text{FI.ID,FI.name,CONTAIN.quantity}}(E2)$$

5.10 Retrieve all orders history made by a customer

Case description: We need to retrieve all the orders' basic information (id, order_time, status, quantity) made by a specific customer with `customer_id`. This happens when the customer wants to track his/her history of orders.

Query:

$$\pi_{\text{ID, Order\_time}}(\sigma_{\text{CID=customer\_id}}(\text{ORDER}))$$

6 Constraints not shown in (E)ER diagram

6.1 Semantic Constraint

- **Constraint on Food Order Items**: Every food item listed under a restaurant must belong to one of the restaurant's predefined categories, meaning each FOOD_ITEM entry must reference a CATEGORY associated with its RESTAURANT. Additionally, all items in a FOOD_ORDER must come from the same restaurant, as indicated by the ResID field in FOOD_ORDER. This constraint will be enforced through a trigger that checks each item in the CONTAIN table to ensure it aligns with the ResID specified in the FOOD_ORDER.
- During any FOOD_ORDER transaction where discounts from DISCOUNT_CHANGE_POINT or DISCOUNT_FROM_RESTAURANT are applied, the sum of all discount percentages should be calculated. If the combined discount percentage is greater than 30% of the order's total value, adjust the discounts so that the total discount remains within the 30% limit.

6.2 Participation Constraints

- **Mandatory Participation in payment_method:** Each order must be paid using only one payment_method (card or cash) and should have an associated receipt. If card payment is selected, a valid card number and bank name are required; if cash, the cash amount should be recorded.

6.3 Precedence Constraints

- **Order Status Progression:** The status of a FOOD_ORDER must follow a specific progression, such as 'pending' → 'confirmed' → 'delivered'. It should not jump from 'pending' to 'delivered' without confirmation.

6.4 Rating Calculation

Restaurant Rating Calculation: Customers can rate the food order 1,2,3,4,5 stars according to satisfaction. If they do not rate, the default will be 5 stars. In here, the food order will receive the rating from the customer and the restaurant will get the rating from all that order. So the restaurant rating will be calculated as the total rating of the orders from that restaurant divided by the average.

```
restaurant rating = (total rating point of food order) / (total  number of food
                                order restaurant has)
```

The average restaurant rating can be:

- 4.5-5 points: 5 stars
- 3.5-4.5 points: 4 stars
- 2.5-3.5 points: 3 stars
- 1.5-2.5 points: 2 stars
- $<$ 1.5 points: 1 star

The formula for calculating the rating points could be based on the total number of orders the restaurant has processed, which can be updated dynamically as new orders are added.

6.5 Total Bill Calculation

Total Bill: For each order, calculate the total bill in the ORDER table:

- total_bill = (food_price * quantity) + delivery_fee - discount
- The food price is multiplied by the quantity, then the shipping fee is added. Finally, any applicable discount (from the DISCOUNT table) is subtracted.

6.6 Exchange Points for Discounts

Customer Points Exchange: For every order, the customer earns 20 points. The customer will exchange the point for discount as following policy:

- 5% Discount:
  - Requires 20 points to redeem.
  - Applies only to orders with a minimum value of 50,000VND.
  - Maximum discount cap: 5,000VND.
- 10% Discount:
  - Requires 40 points to redeem.
  - Applies only to orders with a minimum value of 100,000VND.
  - Maximum discount cap: 15,000VND.
- 15% Discount:
  - Requires 80 points to redeem.
  - Applies only to orders with a minimum value of 150,000VND.
  - Maximum discount cap: 30,000VND.
- 20% Discount:
  - Requires 120 points to redeem.
  - Applies only to orders with a minimum value of 250,000VND.
  - Maximum discount cap: 50,000VND.

The attribute end_date of the entity DISCOUNT_FROM_EXCHANGE_POINT will be 7 days after start_date and be calculated as the following formula:

$$\text{end\_date} = \text{start\_date} + 7$$

6.7 Discounts Provided by Restaurant

Restaurant discounts are available in fixed values of 5,000VND, 10,000VND, and 20,000VND, each requiring a minimum order value to be applicable as following policy:

- 5,000VND Discount:
  - Applicable only to orders with a minimum value of 50,000VND.
- 10,000VND Discount:

  - Applicable only to orders with a minimum value of 100,000VND.

- 20,000VND Discount:
  - Applicable only to orders with a minimum value of 150,000VND.

6.8 Domain Constraints

- **Order Status:** The status in the FOOD_ORDER table should be limited to predefined states like 'pending', 'confirmed', 'delivered', and 'canceled'.
- **Discount Dates:** In the DISCOUNT table, start_Date must be earlier than or equal to end_Date to ensure valid promotional periods.

6.9 Referential Integrity Constraints

- **Cascade Deletes/Updates:** When a CUSTOMER or RESTAURANT is deleted. Related rows in FOOD_DELIVER, DISCOUNT, and CATEGORY tables should also be deleted to maintain consistency.
- **Foreign Key Integrity:** In the FOOD_ORDER table, each order should reference valid values for CID (Customer ID), RID (Restaurant ID), and FDID (Food Deliverer ID) in their respective tables (CUSTOMER, RESTAURANT, and FOOD_DELIVER).
- **Constraint on Food Order Items**: All items in a FOOD_ORDER must belong to the same RESTAURANT. The ResID field in FOOD_ORDER specifies the restaurant for the order. This constraint will be enforced through a trigger that checks each item in the CONTAIN table to ensure it aligns with the ResID of the FOOD_ORDER.

6.10 Default Values

- **FOOD_ORDER status:** Set default value of status in FOOD_ORDER to 'pending'.

6.11 Mutual Exclusion Constraints

- **Payment Methods:** Each FOOD_ORDER can be paid using either CARD or CASH, but not both. If card_number is filled in RECEIPT, no CASH field should be active, and vice versa.

6.12 Temporal Constraints

- **Order Time:** The order_time in FOOD_ORDER must be earlier than or equal to the current system time, preventing orders with future timestamps.
- **Discount Validity:** A discount can only be applied if the current date falls within start_Date and end_Date in the DISCOUNT table.

7 Conclusion

The food ordering system designed and implemented in this project provides a comprehensive and scalable solution for managing interactions between customers, restaurants, food delivery personnel, and payment methods. The system encompasses essential entities such as CUSTOMER, RESTAURANT, FOOD_ORDER, and FOOD_DELIVER, with clear relationships mapped through primary keys, foreign keys, and unique constraints. This structure ensures accurate and reliable data management, enabling seamless operations across all aspects of the system.

One of the primary strengths of this design is its focus on data integrity. Through the enforcement of various constraints—such as referential integrity, domain constraints, and participation constraints—the system effectively eliminates potential data anomalies, such as orphaned orders, invalid payment records, and inconsistencies in customer and restaurant information. This ensures that each order is linked to valid customers, restaurants, and delivery personnel, maintaining consistency throughout the entire workflow.

The system also incorporates key business rules that align with real-world operations, such as restricting the progression of order statuses to a logical sequence (e.g., from 'pending' to 'confirmed' to 'delivered'), enforcing the usage limits of discount codes, and validating payment methods. Additionally, mutual exclusion constraints ensure that an order cannot be paid using both CARD and CASH, reflecting standard payment practices.

Furthermore, the system is designed to handle many-to-many relationships effectively through tables like Has_category and Use_discount, which connect restaurants to food categories and customers to discounts, respectively. These relationships allow for flexibility in the system's functionality, enabling diverse interactions and supporting a wide range of user needs.

From a scalability perspective, the system is robust enough to accommodate future expansions, such as adding new categories, restaurants, or features like loyalty programs or additional payment methods. The well-structured database design facilitates ease of maintenance and updates, ensuring that new functionalities can be integrated without compromising data integrity.

In conclusion, this food ordering system provides a reliable, efficient, and adaptable framework for managing the complexities of food orders, delivery services, and payments. Its strong focus on data accuracy, integrity, and real-world usability makes it a powerful tool that can scale with the growing needs of any food service business. By adhering to well-established constraints and ensuring the validity of all key operations, the system offers a solid foundation for future growth and success in the food delivery industry.

8 References

References

[1] R. Elmasri, S. R. Navathe, Fundamentals of Database Systems- 7th Edition, Pearson, 2016.

[2] S. Chittayasothorn, Relational Database Systems: Language, Conceptual Modeling and Design for Engineers, Nutcha Printing Co. Ltd, 2017.