University of Macau

2020/2021

CISC3000(001) Introduction to Database

Group Project

Outline

Part A: Choosing an application

Description:

Our database *project_3000* is designed for restaurants. Using this database, the manager in this restaurant can see for example：

1.Improve the design of food in restaurants
2.Learn about members' information and references to enhance stickiness
3.Know the work situation of the waiters in the restaurant
4.Find out the details of the restaurant's order
5.The turnover of a restaurant over a period of time

For the tables：
1. food:
Attributes: food_id, food_name, food_cruise
Primary key: food_id
Foreign key: None
Constraint: all length of food_id should be 5

2. food_price
Attributes:food_id, size, unit_price
Primary key: food_id, size
Foreign key: (food_id) references food (food_id),
Constraint: unit_price should be greater than 0, food_size should be 1 or 2

3.member:
Attributes: member_id, member_name, member_birth, member_tel, member_email,
Primary key: member_id
Foreign key: None
Constraint: the length of member_id should be 5 and member name cannot be null

4.Waiter:
Attributes:waiter_id, waiter_name, waiter_join_date, waiter_left_date, waiter_birth, waiter_tel,
Primary key: waiter_id
Foreign key: None
Constraint: the length of waiter_id should be 5, waiter_name cannot be null, waiter_join_date cannot be null, waiter_tel cannot be null

5.payment_method_info:

Attributes: method, description
Primary key: method
Foreign key: None
Constraint: Description cannot be null

6. order_list
Attributes:order_id, member_id, waiter_id, order_date, takeaway
Primary key: order_id
Foreign key: (member_id) references member(member_id)
(waiter_id) references waiter(waiter_id)
Constraint: the length of order_id should be 5, order_date cannot be null

7.order_info:
Attributes:order_id, food_id, size, quantity
Primary key: None
Foreign key: (order_id) references order_list(order_id)
(food_id) references food(food_id)
Constraint: quantity should be greater than 0 and size = 1 or size = 2 to represent different size of the dish.

8. payment_method:
Attributes: order_id, method
Primary key: order_id, method
Foreign key: (order_id) references order_list(order_id)
(method) references payment_method_info(method)
Constraint: None

9.discount_rate
Attributes: member_id_initial, discount _rate
Primary key: member_id_initial
Foreign key: None
Constraint: discount_rate should be between 0 and 1

Part B: Specify the assumptions about the database

Specify the assumptions about the database in English (informally).

Description:

We would like to create a database for restaurants. It constructs 9 entity sets and 6 relationship sets. The main function of personal relationship sets is to meet the needs of daily restaurant applications. Due to time and scale constraints, we assume that the restaurant has only one sub-branch and no other branches. If the restaurant develops and expands later, the database model of each branch can be analogized according to this database, which has strong extensibility.
In part B, we will explain the important assumptions we have for these entities and relationships, and why we choose some of their attribute, keys and why we set a certain relationship between two entities. For some relationship sets, we make it simple and for other relationships, we would like to describe something about the relationship and mapping cardinality as well.

For Entities:

1. food with attributes（food_id, food_name, food_cruise)

The food entity contains information about all food provided by the restaurant. We made food_id to be char variable to limit id in the same form. And varchar for food name to save space. According to different categories of food, we specialize foods into several cruises: drink, snack, noodles, rice, and package. (add package here because we need to get the money of package as well). However there are still other kind of food in real life, and if we do not get what cruise is, we define it as general.

2. food_price with attributes(food_id, size, unit_price)

The food_price entity inherits the price of each food the restaurant,with additional attribute size. As the price may varies by time, the best solution is to record the date duration, however it is not very practical for foreign key constraints and for the automated-calculation of subtotal. Therefore, we will periodically store historical data as pure string with the use of JDBC in practice. We add size here to specialize each food in big or general size. General size is represented by 1 and big size is represented by 2. We assume that the unit_price must be greater than 0.

3. member with attributes(member_id, member_name, member_birth, member_tel, member_email)

a. The member entity inherits the attributes of members in this restaurant, containing name, birth, telephone number and email of the member. Using constraint that we need a name to input the member otherwise we cannot address member correctly. For other information if the member is unwilling to give the information, we can make it null.
b. The length of member_id must equal to 5 to keep same form.

c. All the waiter using company card will be grouped under “B0001” as “waiter” and all walk-in guest will be grouped under “C0001” as “walk in guest”. Normal member will always have an initial ‘A’.

4. waiter with attributes(<u>waiter_id</u>, waiter_name, waiter_join_date ,waiter_left_date, waiter_birth, waiter_tel)

The waiter entity contains all information about all waiters in this restaurant.

a. The waiter’s name and telephone number must be NOT NULL.

b. Telephone numbers among different areas are different, so the telephone number is defined by varchar to store information.

c. For those employees who have resigned, we won’t delete their data because deleting the data directly will result in the corresponding waiter_id of the previous order_ id does not exist.Through this way, it will track down which waiter was responsible for a previous order. As in the most case of realistic, if necessary we will periodically store historical data as pure string in other extension languages such as Java.

d.The length of waiter_id must equal to 5 to keep same form. Remark that one waiter working at the restaurant can also be a member in the member table. If they choose to use their member discount, it will counted as normal member. Else, amount they spend will be counted into member id ”B0001” as “waiter”. It is not necessary for each waiter to be members.

5. payment_method_info with attributes(<u>method</u>, description)

As the abbreviations are more concise, we use numbers to represent payment methods. description gives information about what the number represents. Description must always exist as not null.

6. order_list with attributes(<u>order_id</u>, member_id, waiter_id, order_date ,takeaway)

The order_list entity contains all information about all order information in this restaurant.

a. Order_ list is the general table of an order, and other contents are in the order_info table with attributes (order_ id, food_ ID, size, quantity). So we apply decomposition to avoid redundancy.

b. Since an order may have two payment methods in real case, we re-establish a payment_method table with attributes (order_id, method), which can also avoid redundancy.

c. When there is a customer who needs to take away food, ‘1’ will be displayed in the order_list to represent that this order have takeaway option, and if we do not have takeaway in one order, we set the number of takeaway to be 0 represent eating at hall.

7. discount_rate(<u>member_id_initial</u>, discount_rate)

Different people will have different discounts according to different member_id_initial.

For example, waiters discount with employee cards. We constraint the discount rate in 0 to 1 in case we will get an not normal price in real application.

8.order_info(order_id, food_id, size, quantity)

a. This table represents the specific information about the dishes in the order.

b. Customers must select the size and the number of dishes while ordering.There are only two size to choose from: {1,2}

8. This database satisfies BCNF, and each BCNF relationship satisfies 3NF.

(1). view_subtotal_list

In this view, we can query the price of one typical kind of dishes selected by customers before the discount is finished.

(2). view_normal_member

In this view, we can list members' ID according to member_id_initial. All information get by this view is those member who has a member identify but is not a waiter or a walk-in guest.

(3). view_total_list

This view can query the total price after discount and all the relevant information about it.

For relationships:

1. order_info_order_list

This relationship connects entity order_info and entity order_list. The mapping cardinality of this relationship should be one-to-many, and entity order_info should be full participation. Because, for a dish to be ordered we must have its size general or special. (Every order_info should have 1 .. 1 order_id, every order_id should have 1 to n order_info.)

2. member_order_list

This relationship connects entity member and entity order_list. The mapping cardinality of this relationship should be many-to-one, and entity order_list should be total participation.. Because, in our definition of the schema one order must have one and only one customer to take. (Every member_id can have 0 .. * order_list while every order_id can have 1 .. 1 member_id.)

3. order_list_waiter

This relationship connects entity order_list and entity waiter. The mapping cardinality of this relationship should be many-to-one, and entity order should be total participation. Because in this schema we assume that there always exists a waiter who is respond to one particular order. (Every order_id can have 1 .. 1 waiter while every waiter can have 0 .. * order_id.)

4. order_list_payment_method

This relationship connects entity order_list and entity payment_method. The mapping cardinality of this relationship should be many-to-one, and entity order_list should be total participation. Because one order can have more than one payment method.(Every order_id can have 1 .. * payment method while every payment method can have 0 .. * order_id.)

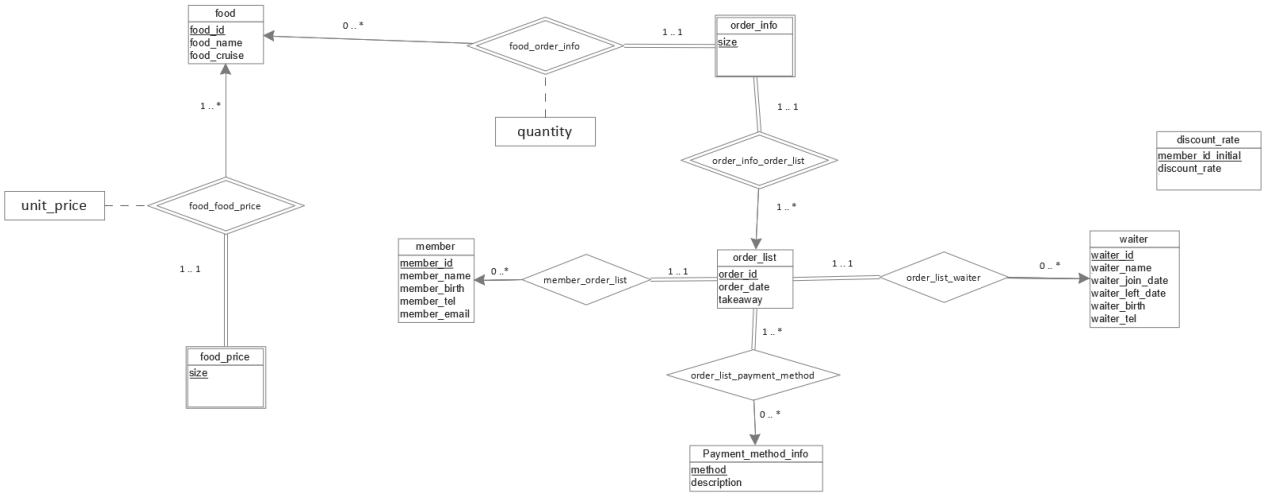
5. food_food_price

This relationship connects entity food and entity food_price. The mapping cardinality of this relationship should be many-to-one, and entity food_price with entity food should be total participation. Because every price must have one dish to correspond, and one dish should have its price to sell. The relationship will have an attribute unit_price.(Every size can have 1 .. 1 food_id while every food id can have 1 .. * size.)

6. food_order_info

This relationship connects entity order_info and entity food. The mapping cardinality of this relationship should be one to many, and entity order_info should be total participation. Because in every order we will have at least 1 dish to be in the ordering list. The relationship will have an attribute quantity.(Every size can have 1 .. 1 food_id while every food id can have 0 .. * size.)

Part C ER-Diagram



Part D Relational Database Schema

Description:

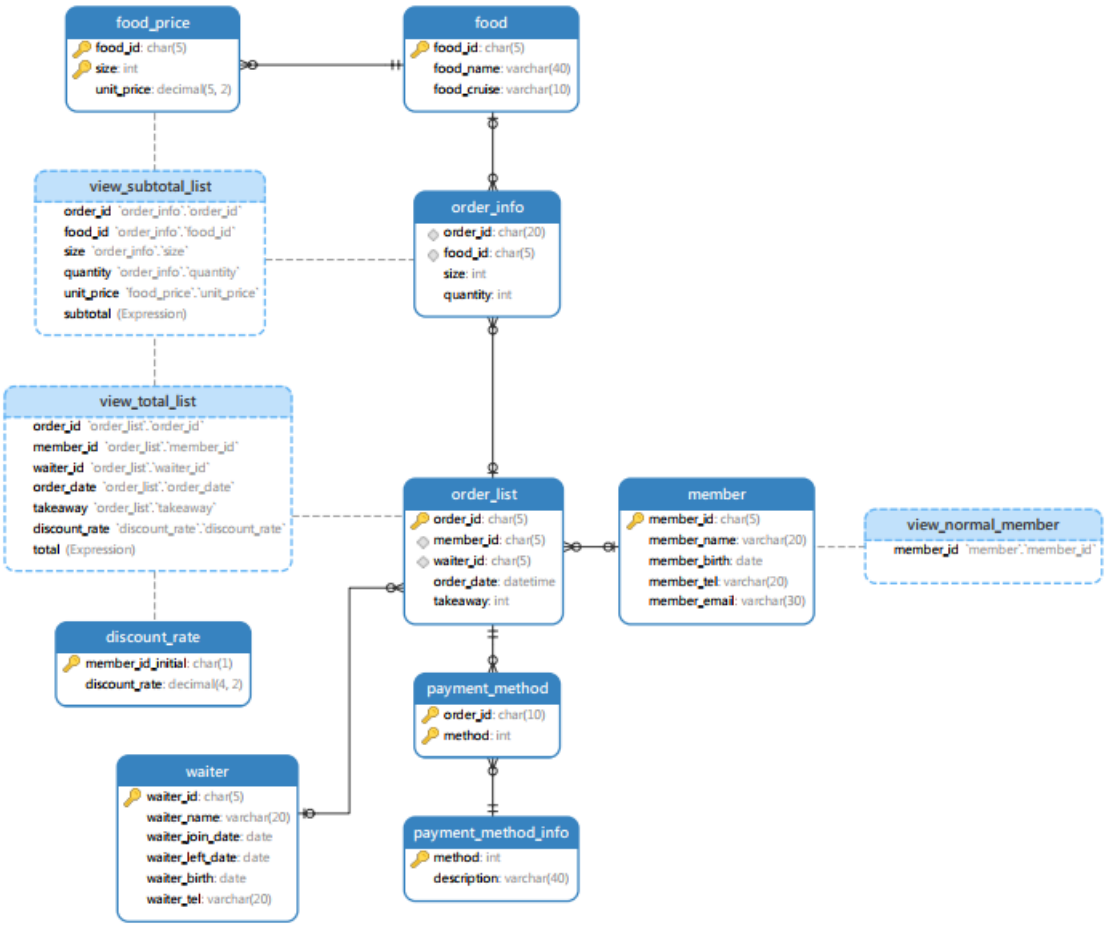
For the SQL statements for creating or altering tables, inserting or updating tables and so on, please check the three files with the expanded name ".sql".

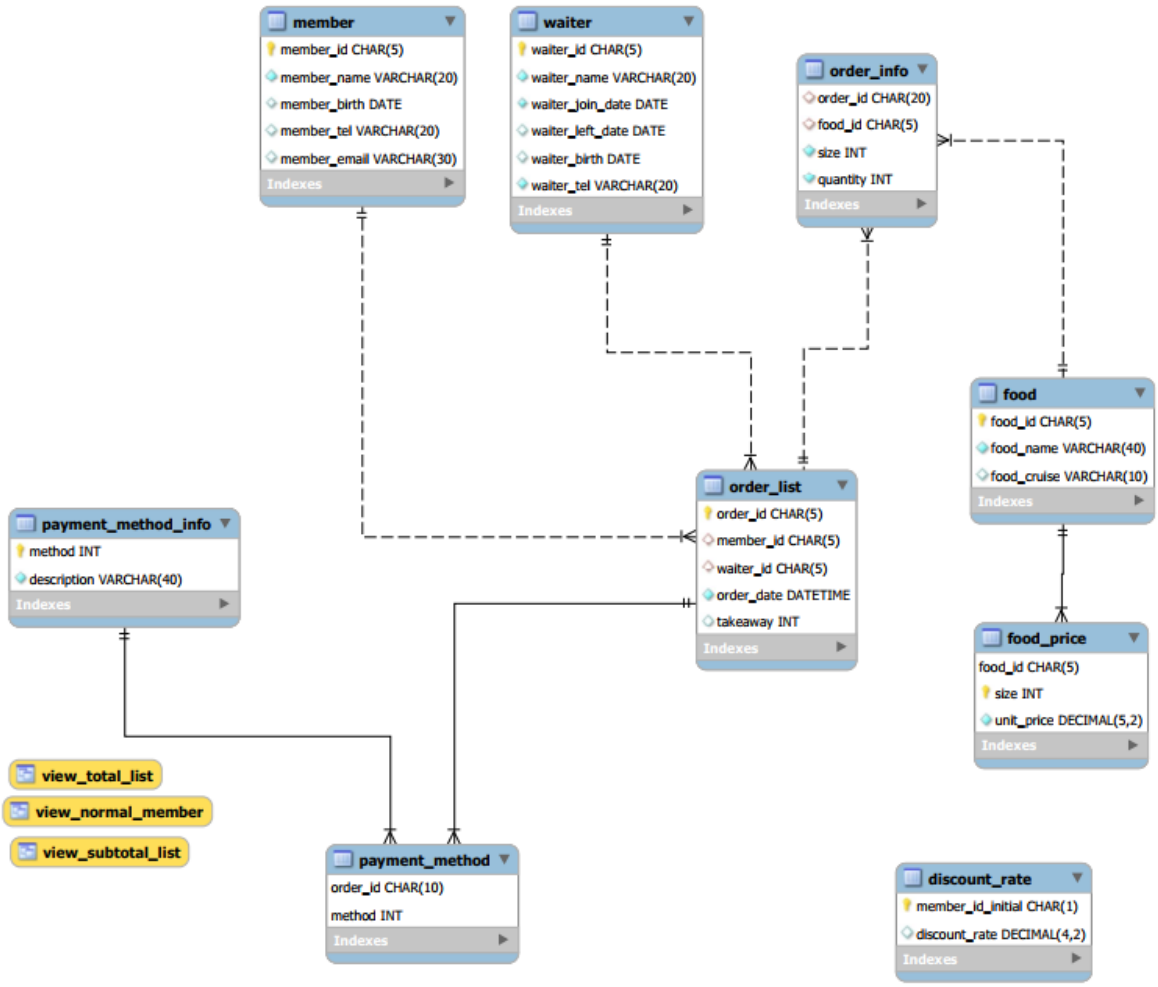
The file "00_CISC3000_Group 1_ddl (project_3000).sql" contains the statements that create database and tables.

The "01_CISC3000_Group 1_insert_values.sql" contains the statements that insert data into the database.

The "02_CISC3000_Group 1_10 Reasonable Queries.sql" contains the statements for 10 reasonable queries .

The following are screen captures of the SQL statement.


food_price
food_id: char(5)
size: int
unit_price: decimal(5, 2)
food
food_id: char(5)
food_name: varchar(40)
food_cruise: varchar(10)
view_subtotal_list
order_id `order_info`.`order_id`
food_id `order_info`.`food_id`
size `order_info`.`size`
quantity `order_info`.`quantity`
unit_price `food_price`.`unit_price`
subtotal (Expression)
order_info
order_id: char(20)
food_id: char(5)
size: int
quantity: int
view_total_list
order_id `order_list`.`order_id`
member_id `order_list`.`member_id`
waiter_id `order_list`.`waiter_id`
order_date `order_list`.`order_date`
takeaway `order_list`.`takeaway`
discount_rate `discount_rate`.`discount_rate`
total (Expression)
order_list
order_id: char(5)
member_id: char(5)
waiter_id: char(5)
order_date: datetime
takeaway: int
member
member_id: char(5)
member_name: varchar(20)
member_birth: date
member_tel: varchar(20)
member_email: varchar(30)
view_normal_member
member_id `member`.`member_id`
discount_rate
member_id_initial: char(1)
discount_rate: decimal(4, 2)
payment_method
order_id: char(10)
method: int
waiter
waiter_id: char(5)
waiter_name: varchar(20)
waiter_join_date: date
waiter_left_date: date
waiter_birth: date
waiter_tel: varchar(20)
payment_method_info
method: int
description: varchar(40)


member
member_id CHAR(5)
member_name VARCHAR(20)
member_birth DATE
member_tel VARCHAR(20)
member_email VARCHAR(30)
Indexes
waiter
waiter_id CHAR(5)
waiter_name VARCHAR(20)
waiter_join_date DATE
waiter_left_date DATE
waiter_birth DATE
waiter_tel VARCHAR(20)
Indexes
order_info
order_id CHAR(20)
food_id CHAR(5)
size INT
quantity INT
Indexes
food
food_id CHAR(5)
food_name VARCHAR(40)
food_cruise VARCHAR(10)
Indexes
order_list
order_id CHAR(5)
member_id CHAR(5)
waiter_id CHAR(5)
order_date DATETIME
takeaway INT
Indexes
payment_method_info
method INT
description VARCHAR(40)
Indexes
food_price
food_id CHAR(5)
size INT
unit_price DECIMAL(5,2)
Indexes
view_total_list
view_normal_member
view_subtotal_list
payment_method
order_id CHAR(10)
method INT
Indexes
discount_rate
member_id_initial CHAR(1)
discount_rate DECIMAL(4,2)
Indexes

Statements for creating tables (Screenshots)

food

```
CREATE TABLE food
  (
     food_id     CHAR(5),
     food_name   VARCHAR(40) NOT NULL,
     food_cruise VARCHAR(10) DEFAULT 'General',
     PRIMARY KEY (food_id),
     CHECK (LENGTH(food_id) = 5)
  );
```

Food_price

```
CREATE TABLE food_price
  (
     food_id    CHAR(5) REFERENCES food(food_id) ON UPDATE CASCADE ON DELETE CASCADE,
     size       INT DEFAULT 0,
     unit_price DECIMAL(5, 2) NOT NULL,
     PRIMARY KEY (food_id, size),
     FOREIGN KEY (food_id) REFERENCES food(food_id),
     CHECK (unit_price > 0)
  );
```

member

```
CREATE TABLE member
  (
     member_id    CHAR(5),
     member_name  VARCHAR(20) NOT NULL,
     member_birth DATE DEFAULT NULL,
     member_tel   VARCHAR(20) DEFAULT NULL,
     member_email VARCHAR(30) DEFAULT NULL,
     PRIMARY KEY (member_id),
     CHECK (LENGTH(member_id) = 5)
  );
```

waiter

```
CREATE TABLE waiter
  (
     waiter_id        CHAR(5),
     waiter_name      VARCHAR(20) NOT NULL,
     waiter_join_date DATE NOT NULL,
     waiter_left_date DATE DEFAULT NULL,
     waiter_birth     DATE DEFAULT NULL,
     waiter_tel       VARCHAR(20) NOT NULL,
     PRIMARY KEY (waiter_id),
     CHECK (LENGTH(waiter_id) = 5)
  );
```

payment_method_info

```
CREATE TABLE payment_method_info
  (
     method      INT,
     description VARCHAR(40) NOT NULL,
     PRIMARY KEY(method)
  );
```

order_list

```
CREATE TABLE order_list
(
    order_id   CHAR(5),
    member_id  CHAR(5) ,
    waiter_id  CHAR(5) ,
    order_date DATETIME NOT NULL,
    takeaway   INT DEFAULT 0,
    PRIMARY KEY (order_id),
    FOREIGN KEY (member_id) REFERENCES member(member_id) ON DELETE SET NULL ON UPDATE CASCADE,
    FOREIGN KEY (waiter_id) REFERENCES waiter(waiter_id) ON DELETE SET NULL ON UPDATE CASCADE,
    CHECK (LENGTH(order_id) = 5)
);
```

order_info

```
CREATE TABLE order_info
(
    order_id CHAR(20) ,
    food_id  CHAR(5) ,
    size     INT NOT NULL,
    quantity INT NOT NULL,
    FOREIGN KEY (order_id) REFERENCES order_list(order_id) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (food_id) REFERENCES food(food_id),
    CHECK (quantity > 0),
    CHECK ((size = 1) OR (size = 2))
);
```

payment_method

```
CREATE TABLE payment_method
(
    order_id CHAR(10) ,
    method   INT DEFAULT 0 ,
    PRIMARY KEY (order_id, method),
    FOREIGN KEY(order_id) REFERENCES order_list(order_id) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(method) REFERENCES payment_method_info(method) ON UPDATE CASCADE
);
```

discount_rate

```
CREATE TABLE discount_rate
(
    member_id_initial CHAR(1),
    discount_rate     DECIMAL(4, 2),
    PRIMARY KEY (member_id_initial),
    CHECK (discount_rate <= 1),
    CHECK (discount_rate >= 0)
);
```

view_subtotal_list

```
CREATE view view_subtotal_list
AS
  (SELECT order_id,
          food_id,
          size,
          quantity,
          unit_price,
          ( unit_price * quantity ) AS subtotal
   FROM   order_info
          JOIN food_price USING(food_id, size));
```

View normal member

```
CREATE view view_normal_member
AS
  (SELECT member_id
   FROM   member
   WHERE  member_id LIKE "a%");
```

View_total_list

```
CREATE view view_total_list
AS
  (SELECT order_id,
          member_id,
          waiter_id,
          order_date,
          takeaway,
          discount_rate.discount_rate,
          FORMAT(Sum(subtotal) * discount_rate,2) AS total
   FROM   order_list
          JOIN view_subtotal_list USING(order_id)
          JOIN discount_rate
            ON ( LEFT(member_id, 1) = discount_rate.member_id_initial)
   GROUP  BY ( order_id ));
```

Statements for inserting tables

food

```
insert into food values('dr001','milktea','drink');
insert into food values('dr002','lemon_tea','drink');
insert into food values('dr003','cola','drink');
insert into food values('sn001','sausage','snack');
insert into food values('sn002','French_fries','snack');
insert into food values('nd001','beef_noodle','noodle');
insert into food values('nd002','chicken_noodle','noodle');
insert into food values('nd003','meatball_noodle','noodle');
insert into food values('rc001','chicken_wing_rice','rice');
insert into food values('rc002','fried_rice','rice');
insert into food values('pk001','package_box','package');
insert into food values('pk002','package_bottle','package');
```

food_price

```
insert into food_price values('dr001',1,20);
insert into food_price values('dr001',2,22);
insert into food_price values('dr002',1,16);
insert into food_price values('dr002',2,18);
insert into food_price values('dr003',1,12);
insert into food_price values('dr003',2,14);
insert into food_price values('sn001',1,8);
insert into food_price values('sn002',1,16);
insert into food_price values('nd001',1,38);
insert into food_price values('nd002',1,36);
insert into food_price values('nd003',1,36);
insert into food_price values('rc001',1,40);
insert into food_price values('rc002',1,33);
insert into food_price values('pk001',1,2);
insert into food_price values('pk002',1,1);
```

member

```
insert into member values('A0001','Ann','2000-01-01','11111111','12345@gmail.com');
insert into member values('A0002','Bob','1999-01-01','13543000000','54321@qq.com');
insert into member values('A0003','Jacky','2003-05-20','63030000','xx000@gmail.com');
insert into member values('A0004','Ms.Wang','1980-01-01',NULL,NULL);
insert into member values('A0005','Mr.Li','1978-12-31','66061234',NULL);
insert into member values('A0006','Jayson','1968-10-01','65432123','bb000@gmail.com');
insert into member values('A0007','Iris','2002-01-01','99998888',NULL);
insert into member values('A0008','Mr.a',NULL,NULL,NULL);
insert into member values('B0001','waiter',NULL,NULL,NULL);
insert into member values('C0001','Walk in guest',NULL,NULL,NULL);
```

waiter

```
insert into waiter values('W0001','Amy','2019-01-01',NULL,NULL,'2025550163');
insert into waiter values('W0002','Derek','2019-02-01','2020-2-1',NULL,'00009999');
insert into waiter values('W0003','Zhang','2020-03-01',NULL,NULL,'2025550150');
insert into waiter values('W0004','Kiki','2018-01-01','2019-05-01','1999-01-01','22334455');
```

payment_method_info

```
insert into payment_method_info values(0,'Unknown');
insert into payment_method_info values(1,'Cash');
insert into payment_method_info values(2,'Card');
insert into payment_method_info values(3,'Wechat');
insert into payment_method_info values(4,'Mpay');
insert into payment_method_info values(5,'Alipay');
```

order_list

```
insert into order_list values('00001','A0001','W0001','2021-01-01',0);
insert into order_list values('00002','A0002','W0001','2021-01-01',0);
insert into order_list values('00003','A0003','W0002','2021-01-02',1);
insert into order_list values('00004','A0007','W0002','2021-01-02',0);
insert into order_list values('00005','B0001','W0002','2021-01-02',1);
insert into order_list values('00006','C0001','W0003','2021-01-03',0);
insert into order_list values('00007','A0005','W0003','2021-01-03',0);
insert into order_list values('00008','A0006','W0003','2021-01-03',0);
insert into order_list values('00009','A0001','W0003','2021-01-04',0);
insert into order_list values('00010','B0001','W0003','2021-01-04',1);
```

order_info

```
insert into order_info values('00001','rc001',1,1);
insert into order_info values('00002','nd002',1,1);
insert into order_info values('00002','sn001',1,1);
insert into order_info values('00003','rc001',1,2);
insert into order_info values('00003','pk001',1,3);
insert into order_info values('00004','dr001',1,1);
insert into order_info values('00004','dr001',2,1);
insert into order_info values('00005','dr002',1,2);
insert into order_info values('00005','pk002',1,2);
insert into order_info values('00006','nd003',1,1);
insert into order_info values('00006','dr002',1,1);
insert into order_info values('00007','nd002',1,1);
insert into order_info values('00007','rc002',1,1);
insert into order_info values('00008','nd001',1,1);
insert into order_info values('00008','sn002',1,1);
insert into order_info values('00009','dr002',1,1);
insert into order_info values('00010','dr001',2,2);
insert into order_info values('00010','pk002',1,2);
```

payment_method

```
85 •  insert into payment_method values('00001',2);
86 •  insert into payment_method values('00002',1);
87 •  insert into payment_method values('00003',1);
88 •  insert into payment_method values('00004',5);
89 •  insert into payment_method values('00005',4);
90    insert into payment_method values('00006',1);
91 •  insert into payment_method values('00007',1);
92 •  insert into payment_method values('00008',1);
93 •  insert into payment_method values('00009',5);
94 •  insert into payment_method values('00010',4);
```

discount_rate

```
96 •  insert into discount_rate values('A',0.9);
97 •  insert into discount_rate values('B',0.8);
98 •  insert into discount_rate values('C',1);
```

Statements for creating triggers ：

Part E Ten Reasonable Queries

A). List all relations and their contents:

```
1 •  SELECT * FROM food;
2 •  SELECT * FROM food_price;
3 •  SELECT * FROM member;
4 •  SELECT * FROM waiter;
5 •  SELECT * FROM payment_method_info;
6 •  SELECT * FROM order_list;
7 •  SELECT * FROM order_info;
8 •  SELECT * FROM payment_method;
9 •  SELECT * FROM view_subtotal_list;
10 •  SELECT * FROM view_total_list;
11 •  SELECT * FROM view_normal_member;
12 •  SELECT * FROM discount_rate;
```

food

food_id	food_name	food_cruise
dr001	milktea	drink
dr002	lemon_tea	drink
dr003	cola	drink
nd001	beef_noodle	noodle
nd002	chicken_noodle	noodle
nd003	meatball_noodle	noodle
pk001	package_box	package
pk002	package_bottle	package
rc001	chicken_wing_rice	rice
rc002	fried_rice	rice
sn001	sausage	snack
sn002	French_fries	snack

food_price

food_id	size	unit_price
dr001	1	20.00
dr001	2	22.00
dr002	1	16.00
dr002	2	18.00
dr003	1	12.00
dr003	2	14.00
nd001	1	38.00
nd002	1	36.00
nd003	1	36.00
pk001	1	2.00
pk002	1	1.00
rc001	1	40.00
rc002	1	33.00
sn001	1	8.00
sn002	1	16.00

member

member_id	member_name	member_birth	member_tel	member_email
A0001	Ann	2000-01-01	11111111	12345@gmail.com
A0002	Bob	1999-01-01	13543000000	54321@qq.com
A0003	Jacky	2003-05-20	63030000	xx000@gmail.com
A0004	Ms.Wang	1980-01-01	NULL	NULL
A0005	Mr.Li	1978-12-31	66061234	NULL
A0006	Jayson	1968-10-01	65432123	bb000@gmail.com
A0007	Iris	2002-01-01	99998888	NULL
A0008	Mr.a	NULL	NULL	NULL
B0001	waiter	NULL	NULL	NULL
C0001	Walk in guest	NULL	NULL	NULL

waiter

waiter_id	waiter_name	waiter_join_date	waiter_left_date	waiter_birth	waiter_tel
W0001	Amy	2019-01-01	NULL	NULL	2025550163
W0002	Derek	2019-02-01	2020-02-01	NULL	00009999
W0003	Zhang	2020-03-01	NULL	NULL	2025550150
W0004	Kiki	2018-01-01	2019-05-01	1999-01-01	22334455

payment_method_info

method	description
0	Unknown
1	Cash
2	Card
3	Wechat
4	Mpay
5	Alipay

order_list

order_id	member_id	waiter_id	order_date	takeaway
00001	A0001	W0001	2021-01-01 00:00:00	0
00002	A0002	W0001	2021-01-01 00:00:00	0
00003	A0003	W0002	2021-01-02 00:00:00	1
00004	A0007	W0002	2021-01-02 00:00:00	0
00005	B0001	W0002	2021-01-02 00:00:00	1
00006	C0001	W0003	2021-01-03 00:00:00	0
00007	A0005	W0003	2021-01-03 00:00:00	0
00008	A0006	W0003	2021-01-03 00:00:00	0
00009	A0001	W0003	2021-01-04 00:00:00	0
00010	B0001	W0003	2021-01-04 00:00:00	1

order_info

order_id	food_id	size	quantity
00001	rc001	1	1
00002	nd002	1	1
00002	sn001	1	1
00003	rc001	1	2
00003	pk001	1	3
00004	dr001	1	1
00004	dr001	2	1
00005	dr002	1	2
00005	pk002	1	2
00006	nd003	1	1
00006	dr002	1	1
00007	nd002	1	1
00007	rc002	1	1
00008	nd001	1	1
00008	sn002	1	1
00009	dr002	1	1
00010	dr001	2	2
00010	pk002	1	2

Payment_method

order_id	method
00002	1
00003	1
00006	1
00007	1
00008	1
00001	2
00005	4
00010	4
00004	5
00009	5

View_subtotal_list

order_id	food_id	size	quantity	unit_price	subtotal
00004	dr001	1	1	20.00	20.00
00004	dr001	2	1	22.00	22.00
00010	dr001	2	2	22.00	44.00
00005	dr002	1	2	16.00	32.00
00006	dr002	1	1	16.00	16.00
00009	dr002	1	1	16.00	16.00
00008	nd001	1	1	38.00	38.00
00002	nd002	1	1	36.00	36.00
00007	nd002	1	1	36.00	36.00
00006	nd003	1	1	36.00	36.00
00003	pk001	1	3	2.00	6.00
00005	pk002	1	2	1.00	2.00
00010	pk002	1	2	1.00	2.00
00001	rc001	1	1	40.00	40.00
00003	rc001	1	2	40.00	80.00
00007	rc002	1	1	33.00	33.00
00002	sn001	1	1	8.00	8.00
00008	sn002	1	1	16.00	16.00

View_total_list

order_id	member_id	waiter_id	order_date	takeaway	discount_rate	total
00001	A0001	W0001	2021-01-01 00:00:00	0	0.90	36.00
00002	A0002	W0001	2021-01-01 00:00:00	0	0.90	39.60
00003	A0003	W0002	2021-01-02 00:00:00	1	0.90	77.40
00004	A0007	W0002	2021-01-02 00:00:00	0	0.90	37.80
00007	A0005	W0003	2021-01-03 00:00:00	0	0.90	62.10
00008	A0006	W0003	2021-01-03 00:00:00	0	0.90	48.60
00009	A0001	W0003	2021-01-04 00:00:00	0	0.90	14.40
00005	B0001	W0002	2021-01-02 00:00:00	1	0.80	27.20
00010	B0001	W0003	2021-01-04 00:00:00	1	0.80	36.80
00006	C0001	W0003	2021-01-03 00:00:00	0	1.00	52.00

View_normal_id

member_id
A0001
A0002
A0003
A0004
A0005
A0006
A0007
A0008

discount_rate

member_id_initial	discount_rate
A	0.90
B	0.80
C	1.00

B). Ten reasonable queries:

1) Display the highest-selling food (in amount) with the percentage proportion. Remember only to display those that were ordered before.

```
-- Question 01
SELECT view_subtotal_list.food_id,
       food.food_name,
       Sum(view_subtotal_list.subtotal) AS "Total sales for each food",
       COUNT(view_subtotal_list.subtotal) AS "Total count ",
       CONCAT(FORMAT(100 * Sum(view_subtotal_list.subtotal) / (SELECT Sum(view_subtotal_list.subtotal) FROM view_subtotal_list),2),"%")  AS "Percentage of sales"
FROM   view_subtotal_list
       RIGHT OUTER JOIN food USING(food_id)
WHERE  ( LEFT(view_subtotal_list.food_id, 2) <> 'pk' )
GROUP  BY food.food_id
ORDER  BY Sum(view_subtotal_list.subtotal) DESC;
```

food_id	food_name	Total sales for each food	Total count	Percentage of sales
rc001	chicken_wing_rice	120.00	2	24.84%
dr001	milktea	86.00	3	17.81%
nd002	chicken_noodle	72.00	2	14.91%
dr002	lemon_tea	64.00	3	13.25%
nd001	beef_noodle	38.00	1	7.87%
nd003	meatball_noodle	36.00	1	7.45%
rc002	fried_rice	33.00	1	6.83%
sn002	French_fries	16.00	1	3.31%
sn001	sausage	8.00	1	1.66%

2) Display the number of member's birthdays and the percentage proportion in each month. Only show those months with member's birthday in it. Remember to sort it descending to find out the most common month member's birthday in.

```
14    -- Question 02
15    SELECT Month(member_birth) AS "Members' birthday month",
16           Count(Month(member_birth)) AS 'Count',
17           Concat(Format(100 * Count(Month(member_birth)) / (SELECT Count(Month(member_birth)) FROM member
18                        WHERE NOT Isnull(member_birth)), 2), "%") AS "Percentage"
19    FROM   member
20    WHERE  NOT Isnull(member_birth)
21    GROUP  BY Month(member_birth)
22    ORDER  BY Month(member_birth);
```

Members' birthday month	Count	Percentage
1	4	57.14%
5	1	14.29%
10	1	14.29%
12	1	14.29%

3) Find out the most popular (in quantity) food/beverage by displaying the total quantity sold for each one in descending order. Remember to filter out the package group with food_cruise starts with "pk".

```
23    -- Question 03
24    -- Without size
25    SELECT view_subtotal_list.food_id,
26           food.food_name,
27           Sum(view_subtotal_list.quantity) AS "total quantity for each food"
28    FROM   view_subtotal_list
29           JOIN food USING(food_id)
30    WHERE  ( LEFT(view_subtotal_list.food_id, 2) <> 'pk' )
31    GROUP  BY view_subtotal_list.food_id
32    ORDER  BY view_subtotal_list.quantity DESC;
```

	food_id	food_name	total quantity for each food
▶	dr002	lemon_tea	4
	dr001	milktea	4
	nd001	beef_noodle	1
	nd002	chicken_noodle	2
	nd003	meatball_noodle	1
	rc001	chicken_wing_rice	3
	rc002	fried_rice	1
	sn001	sausage	1
	sn002	French_fries	1

4) Display the total sales and the percentage proportion of each waiter. Remember to include the new waiter with no order as “NULL”. Sort them in descending order to find out the most contributed waiter.

```
36    -- Question 04
37 •  SELECT view_total_list.waiter_id,
38           waiter.waiter_name,
39           FORMAT(Sum(view_total_list.total),2) AS "Total sales of each waiter",
40           CONCAT(FORMAT(100 * Sum(total) / (SELECT Sum(total) FROM view_total_list),2),"%") AS 'Total Sales Percentage'
41    FROM   view_total_list
42           RIGHT JOIN waiter USING (waiter_id)
43    GROUP  BY view_total_list.waiter_id
44    ORDER  BY Sum(view_total_list.total) DESC;
```

	waiter_id	waiter_name	Total sales of each waiter	Total Sales Percentage
▶	W0003	Zhang	213.90	49.53%
	W0002	Derek	142.40	32.97%
	W0001	Amy	75.60	17.50%
	NULL	Kiki	NULL	NULL

5) Display the total count paid by each payment method and the percentage proportion. Remember to include those which hadn’t been paid yet as “NULL” total count. Remember an order paid with two payment methods or more is counted multiple times.

```sql
-- Question 05
SELECT payment_method.method,
       payment_method_info.description,
       COUNT(total) AS "Total count",
       CONCAT(FORMAT(100 * COUNT(total) / (SELECT COUNT(total) FROM view_total_list),2),"%") AS 'Total Count Percentage'
FROM   view_total_list
       JOIN payment_method USING(order_id)
       RIGHT JOIN payment_method_info USING (method)
GROUP  BY payment_method.method
ORDER  BY Sum(total) DESC;
```

	method	description	Total count	Total Count Percentage
▶	1	Cash	5	50.00%
	4	Mpay	2	20.00%
	5	Alipay	2	20.00%
	2	Card	1	10.00%
	NULL	Unknown	0	0.00%

6) Display the total sales for 3 main groups of members (normal member, waiter, and walk-in guest) and show the proportion of them. Remember to sort in descending to find which group spends the most.

```sql
-- Question 06
SELECT       LEFT(view_total_list.member_id, 1) AS member_initial,
             IF(LEFT(view_total_list.member_id, 1) = 'A', 'member', member.member_name) AS 'member type',
             FORMAT(Sum(view_total_list.total),2) AS "Total spend",
             CONCAT(FORMAT(100 * Sum(total) / (SELECT Sum(total) FROM view_total_list),2),"%") AS 'Total Sales Percentage'
FROM   view_total_list
       JOIN member USING(member_id)
GROUP  BY LEFT(view_total_list.member_id, 1)
ORDER  BY view_total_list.member_id;
```

	member_initial	member type	Total spend	Total Sales Percentage
▶	A	member	315.90	73.14%
	B	waiter	64.00	14.82%
	C	Walk in guest	52.00	12.04%

7) Display the amount and percentage proportion in sales for each food cruise group. Remember to sort them in descending order to find out the highest sales food cruise group.

```sql
-- Question 07
SELECT LEFT(food_id, 2) AS 'Id Group',
       food.food_cruise,
       Sum(subtotal) AS 'Total Sales',
       Concat(Format(100 * Sum(subtotal) / (SELECT Sum(subtotal) FROM view_subtotal_list), 2), "%") AS 'Total Sales Percenta
       FROM   view_subtotal_list
       JOIN food using(food_id)
GROUP  BY LEFT(food_id, 2)
ORDER  BY Sum(subtotal) DESC;
```

	Id Group	food_cruise	Total Sales	Total Sales Percentage
▶	rc	rice	153.00	31.68%
	dr	drink	150.00	31.06%
	nd	noodle	146.00	30.23%
	sn	snack	24.00	4.97%
	pk	package	10.00	2.07%

8) Display the waiter ID, waiter name, and the accumulated working days of each waiter. For those who are still employed it is calculated until the current date. Remember to sort them in descending order to find out the longest working waiter.

```
75    -- Question 08
76 •  SELECT waiter_id,
77           waiter_name,
78           IF(Isnull(waiter_left_date), Datediff(Curdate(), waiter_join_date),
79           Datediff(waiter_left_date, waiter_join_date)) AS 'Accumulated working days'
80    FROM   waiter
81    ORDER BY IF(Isnull(waiter_left_date), Datediff(Curdate(), waiter_join_date),
82           Datediff(waiter_left_date, waiter_join_date)) DESC;
```

	waiter_id	waiter_name	Accumulated working days
▶	W0001	Amy	859
	W0004	Kiki	485
	W0003	Zhang	434
	W0002	Derek	365

9) Display the total amount of sales and the percentage proportion for both group takeaway and not takeaway.

```
70    -- Question 09
71    SELECT takeaway,
72           FORMAT(Sum(total),2) AS "Total Sales",
73           CONCAT(FORMAT(100 * Sum(total) / (SELECT Sum(total) FROM view_total_list),2),"%") AS 'Total Sales Percentage'
74    FROM   view_total_list
75    GROUP  BY takeaway
76    ORDER  BY Sum(total) DESC;
```

	takeaway	Total Sales	Total Sales Percentage
▶	0	290.50	67.26%
	1	141.40	32.74%

10) Display the total number, percentage proportion in number, total amount, and the average sales per order for

both group takeaway and not takeaway.

```
92    -- Question 10
93    SELECT takeaway,
94           Count(total) AS 'Count',
95           CONCAT(FORMAT(100 * Count(total) / (SELECT Count(total) FROM view_total_list),2),"%") AS 'Count Percentage',
96           SUM(total) AS 'Total Amount',
97           FORMAT(SUM(total)/COUNT(total),2) AS 'Average sales each'
98    FROM   view_total_list
99    GROUP  BY view_total_list.takeaway;
```

	takeaway	Count	Count Percentage	Total Amount	Average sales each
▶	0	7	70.00%	290.5	41.50
	1	3	30.00%	141.4	47.13