Noodles Restaurant management Database

For Tan Viet Noodles House Restaurant

Nguyen Bich Ha Thach

StudentID: 24608832

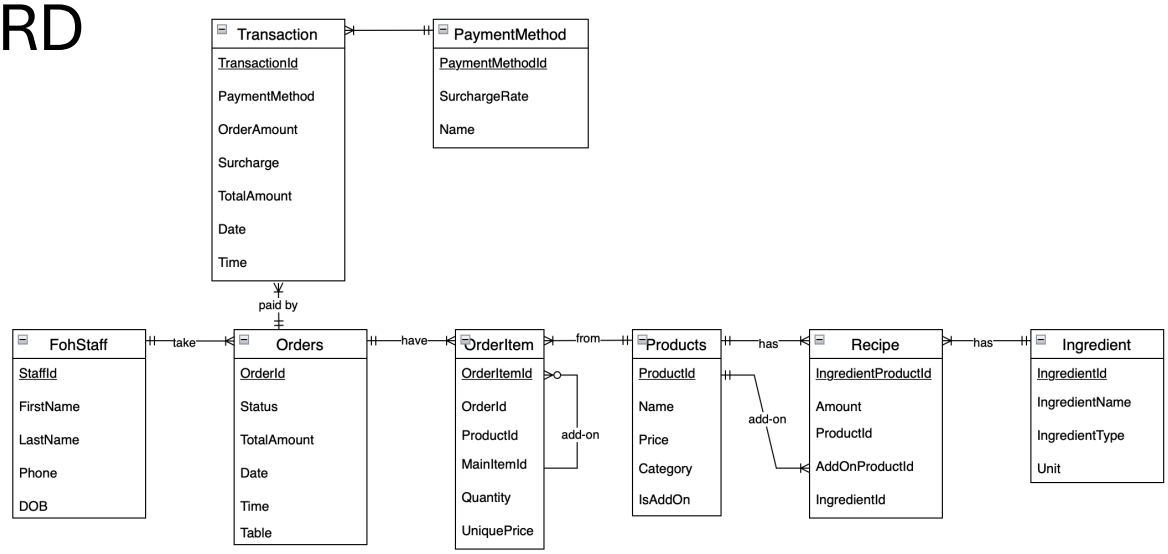
1

Case study description

- Database is built on a real Vietnamese noodles restaurant where I have worked as a part-time job. This is a Vietnamese restaurant, where noodles and rice can be customized. https://tanviet.com.au/menu/
- There are many staffs.
- 1 staff can make many orders for customer. Customer information is not recorded.
- 1 order (as a bill) can be paid by many transactions.
- 1 order can have many items from menu. 1 items can be ordered by many orders.
- 1 item have many ingredients. 1 item have specific recipe with related ingredients.
- There are many payment methods. A transaction can choose 1 of them. Surcharge varies on payment methods which effects value of the order.
- Data is built my students for studying, not related to real data from the restaurant.

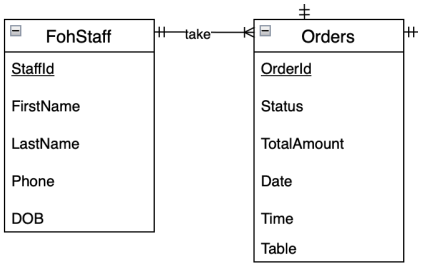
2

ERD



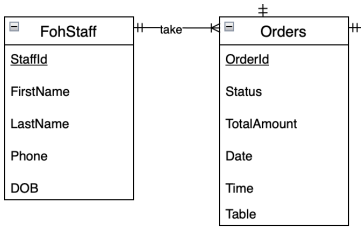
Single to many relationship

- Relationship between FohStaff and Orders
- One staff can take many orders




```
CREATE TABLE FohStaff (
        StaffId INT PRIMARY KEY,
        FirstName VARCHAR(30),
        LastName VARCHAR(30),
        Phone CHAR(20),
        DOB DATE
);
CREATE TABLE Orders(
        OrderId CHAR(10) PRIMARY KEY,
        Status VARCHAR(30),
        TotalAmount DECIMAL(10, 2),
        Date DATE,
        Time TIME,
        TableNo INT,
        Staff INT,
        FOREIGN KEY (Staff) REFERENCES FohStaff(StaffId));
```

Single to many relationship




```
postgres=#
SELECT * FROM FohStaff WHERE staffId=1234567;
 staffid | firstname | lastname |        phone         |    dob
---------+-----------+----------+----------------------+------------
 1234567 | Harley    | Thach    | 0412 345 678         | 1997-08-24
(1 row)
```

```
postgres=#
SELECT * FROM orders WHERE staff=1234567;;
  orderid   | status | totalamount |    date    |   time   | tableno |  staff
------------+--------+-------------+------------+----------+---------+---------
 10005      | Done   |      154.50 | 2024-03-25 | 18:08:13 |      16 | 1234567
 10019      | Done   |      190.50 | 2024-04-03 | 13:12:19 |      16 | 1234567
 10021      | Done   |       82.00 | 2024-04-04 | 12:32:33 |      10 | 1234567
(3 rows)
```

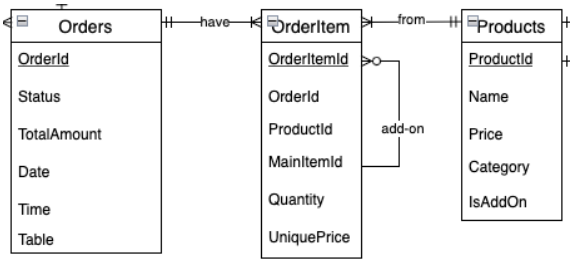
5

```
CREATE TABLE Orders(
        OrderId CHAR(10) PRIMARY KEY,
        Status VARCHAR(30),
        TotalAmount DECIMAL(10, 2),
        Date DATE,
        Time TIME,
        TableNo INT,
        Staff INT,
        FOREIGN KEY (Staff) REFERENCES FohStaff(StaffId));

CREATE TABLE Products(
        ProductId CHAR(30) PRIMARY KEY,
        Name VARCHAR(100),
        Price DECIMAL(10, 2),
        Category VARCHAR(50),
        IsAddOn BOOLEAN);

CREATE TABLE OrderItem (
        OrderItemId CHAR(10),
        OrderId CHAR(10),
        ProductId CHAR(10),
        MainItemId CHAR(10),
        Quantity INT,
        UniquePrice DECIMAL(10, 2),
        PRIMARY KEY (OrderItemId),
        FOREIGN KEY (OrderId) REFERENCES Orders(OrderId),
        FOREIGN KEY (ProductId) REFERENCE Products(ProductId));
```

Many to many



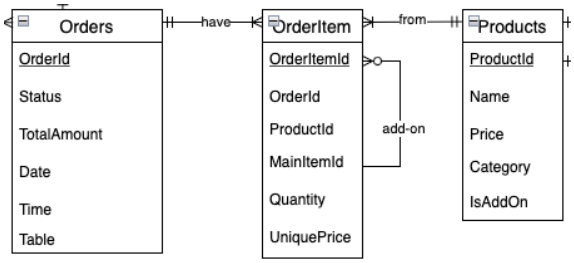
6

Many to many

```
postgres=# SELECT * FROM orders WHERE orderid='10001';
  orderid   | status | totalamount |    date    |   time   | tableno |  staff
------------+--------+-------------+------------+----------+---------+---------
 10001      | Done   |      111.50 | 2024-03-24 | 17:12:50 |       8 | 1234572
(1 row)
```

```
postgres=# SELECT * FROM orderitem WHERE orderid='10001';
 orderitemid |  orderid   | productid | mainitemid | quantity | uniqueprice
-------------+------------+-----------+------------+----------+-------------
 O100010     | 10001      | F23       |            |        2 |       20.00
 O100011     | 10001      | F41       | O100011    |        2 |        0.00
 O100012     | 10001      | F53       |            |        3 |        5.00
 O100013     | 10001      | F37       |            |        1 |       18.50
 O100014     | 10001      | F55       |            |        3 |        6.00
 O100015     | 10001      | F29       |            |        1 |       20.00
 O100016     | 10001      | F50       | O100016    |        1 |        0.00
(7 rows)
```

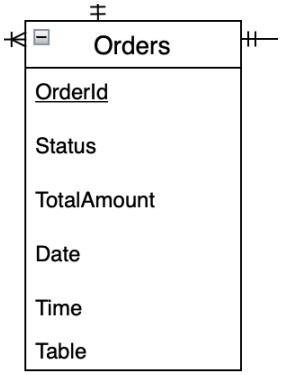
```
postgres=# SELECT * FROM products WHERE productid='F41' OR productid='F23' ;
          productid           |          name          | price | category | isaddon
------------------------------+------------------------+-------+----------+---------
 F23                          | Goat curry  with noodles | 20.00 | Noodles  | f
 F41                          | Egg noodles with soup    |  0.00 | Add-on   | t
(2 rows)
```



7

Simple query

Question:
Show the orders in 25th March 2024



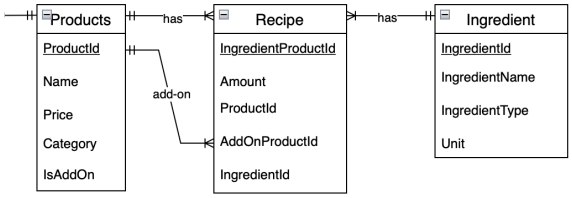
1. Check Orders table
2. Simple query with WHERE

```
postgres=# SELECT * FROM orders WHERE date ='2024-03-25';
  orderid   | status | totalamount |    date    |   time   | tableno |  staff
------------+--------+-------------+------------+----------+---------+---------
 10003      | Done   |       94.00 | 2024-03-25 | 14:28:23 |       6 | 1234577
 10004      | Done   |       81.00 | 2024-03-25 | 18:05:59 |      17 | 1234575
 10005      | Done   |      154.50 | 2024-03-25 | 18:08:13 |      16 | 1234567
(3 rows)
```

8

A query with NATURAL JOIN OR CROSS PRODUCT

Question:
Products having Crispy skin chicken



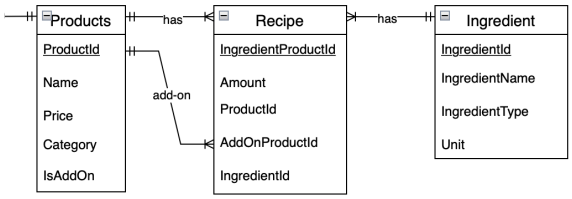
1. Check IngredientId of Crispy skin chicken ➔ I21
2. ProductId from Products is foreign key to Recipe table => using Natural Join or Cross Product from two table Products and Recipe.

```
postgres=# SELECT * FROM ingredient WHERE ingredientname ='Crispy skin chicken
';
 ingredientid |   ingredientname    | ingredienttype | unit
--------------+---------------------+----------------+-------
 I21          | Crispy skin chicken | Toppings       | piece
(1 row)
```

9

A query with NATURAL JOIN OR CROSS PRODUCT

Question:
Products having Crispy skin chicken



Natural Join

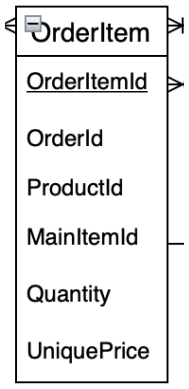
```
postgres=#  SELECT DISTINCT productid, name FROM products NATURAL JOIN recipe
WHERE ingredientid='I21';
           productid            |              name
--------------------------------+---------------------------------
 F1                             | Crispy skin chicken with noodles
 F25                            | Crispy skin chicken with rice
 F32                            | Crispy skin chicken Only
(3 rows)
```

Cross product

```
postgres=# SELECT DISTINCT products.productid, products.name FROM products, recipe
 WHERE recipe.ingredientId='I21' AND products.productid=recipe.mainproductid;
 productid  |              name
------------+---------------------------------
 F1         | Crispy skin chicken with noodles
 F25        | Crispy skin chicken with rice
 F32        | Crispy skin chicken Only
(3 rows)
```

10

A query using GROUP BY

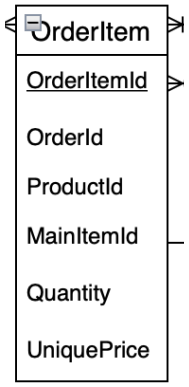


Question:
Determine the order which have the highest number of items

1. Choose table OrderItem which lists detailed orders: orders, quantity of each item in each order.

```
postgres=# SELECT * FROM orderitem LIMIT 5;
 orderitemid |  orderid   | productid  | mainitemid | quantity | uniqueprice
-------------+------------+------------+------------+----------+-------------
 O100010     | 10001      | F23        |            |        2 |       20.00
 O100011     | 10001      | F41        | O100010    |        2 |        0.00
 O100012     | 10001      | F53        |            |        3 |        5.00
 O100013     | 10001      | F37        |            |        1 |       18.50
 O100014     | 10001      | F55        |            |        3 |        6.00
(5 rows)
```


A query using GROUP BY



Question:
Determine the order which have the highest number of items

1. Group BY OrderId
2. OrderItemId can be a main item or a customized one.

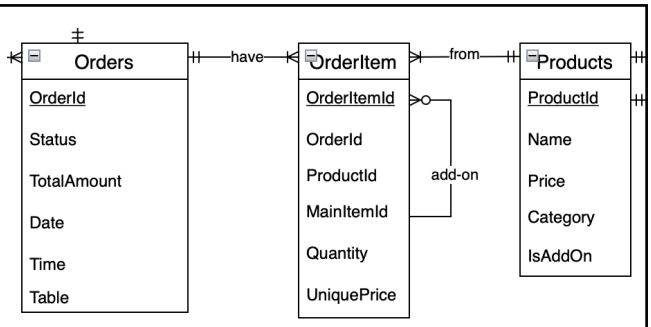
➔ Need to add condition MainItemId IS NULL

3. Aggregate function: SUM()

```
postgres=# SELECT orderid, sum(quantity) AS biggest_order FROM orderitem
WHERE mainitemid IS NULL GROUP BY orderid  ORDER BY sum(quantity) DESC
LIMIT 1;
  orderid   | biggest_order
------------+---------------
 10029      |            20
(1 row)
```

A query using SUB QUERY

Question:
Determine date of order which have the highest number of Special Beef pho



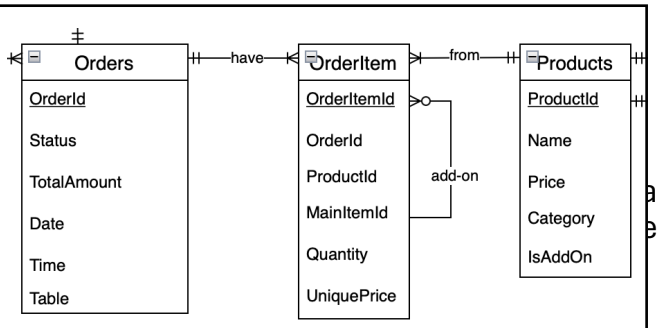
1. Determine ProductId of Special Beef Pho is F3.
2. Choose table OrderItem which lists detailed orders: orders, quantity of each item in each order.
3. Query to find the highest number of Special Beef pho in an order

```
postgres=# SELECT max(quantity) FROM orderitem  WHERE productid='F3';
 max
-----
   4
(1 row)
```

13

A query using SUB QUERY

Question:
Determine date of order which have the highest number of Special Beef pho



Using sub query

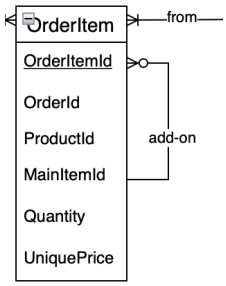
```
postgres=# SELECT orderid, quantity FROM orderitem WHERE
quantity= (SELECT max(quantity) FROM orderitem  WHERE pro
ductid='F3') AND productid='F3'
;
  orderid   | quantity
------------+----------
 10014      |        4
(1 row)
```

14

A query can be implemented by CROSS PRODUCT, cannot by NATURAL JOIN

Question:

Show a list of items in order 10001 which need to customize and show its customization



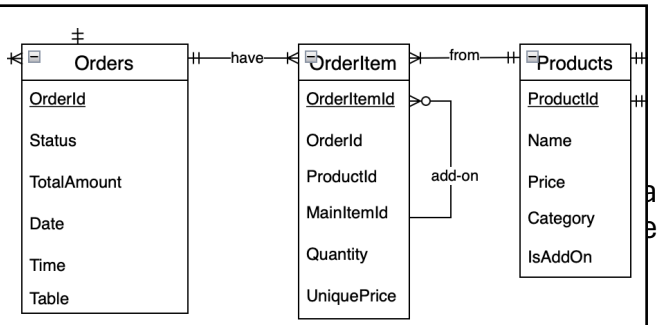
- Because MainItemId and ProductitemId have different name, NATURAL JOIN cannot be used.
- Show OrderItem data in orderId 10001

```
postgres=# SELECT * FROM orderitem WHERE orderid='10001';
 orderitemid |  orderid   | productid  | mainitemid | quantity | uniqueprice
-------------+------------+------------+------------+----------+-------------
 O100010     | 10001      | F23        |            |        2 |       20.00
 O100011     | 10001      | F41        | O100010    |        2 |        0.00
 O100012     | 10001      | F53        |            |        3 |        5.00
 O100013     | 10001      | F37        |            |        1 |       18.50
 O100014     | 10001      | F55        |            |        3 |        6.00
 O100015     | 10001      | F29        |            |        1 |       20.00
 O100016     | 10001      | F50        | O100015    |        1 |        0.00
(7 rows)
```


A query can be implemented by CROSS PRODUCT, cannot by NATURAL JOIN

Question:

Show a list of item of order 10001 which a mainItem going along with customized one.



Using the CROSS PRODUCT to join OrderItem with itself.

```
postgres=# SELECT main.orderid, main.productid AS maindish, addon.productid AS customize
 FROM orderitem AS main, orderitem AS addon
 WHERE main.orderitemid=addon.mainitemid AND main.orderid='10001';
  orderid   |  maindish  | customize
------------+------------+-----------
 10001      | F23        | F41
 10001      | F29        | F50
(2 rows)
```

CHECK Statement

```
CREATE TABLE Orders(
    OrderId CHAR(10) PRIMARY KEY,
        Status VARCHAR(30),
        TotalAmount DECIMAL(10, 2),
        CONSTRAINT TotalAmount  CHECK (TotalAmount > 0),
        Date DATE,
        Time TIME,
        TableNo INT,
        Staff INT,
        FOREIGN KEY (Staff) REFERENCES FohStaff(StaffId),
        CONSTRAINT TableNo CHECK ((TableNo >= 0) AND (TableNo <30 )));
```

```
CREATE TABLE Products(
        ProductId CHAR(30) PRIMARY KEY,
        Name VARCHAR(100),
        Price DECIMAL(10, 2),
        CONSTRAINT  Price CHECK (Price >= 0),
        Category VARCHAR(50),
        IsAddOn BOOLEAN
);
```

```
CREATE TABLE OrderItem (
        OrderItemId CHAR(10),
    OrderId CHAR(10),
    ProductId CHAR(10),
        MainItemId CHAR(10),
        Quantity INT,
        UniquePrice DECIMAL(10, 2),
        PRIMARY KEY (OrderItemId),
        FOREIGN KEY (OrderId) REFERENCES Orders(OrderId),
        FOREIGN KEY (ProductId) REFERENCES Products(ProductId),
        CONSTRAINT  UniquePrice CHECK (UniquePrice >= 0),
        CONSTRAINT Quantity CHECK (Quantity > 0));
```

17

ACTION Statement

```
CREATE TABLE OrderItem (
        OrderItemId CHAR(10),
    OrderId CHAR(10),
    ProductId CHAR(10),
        MainItemId CHAR(10),
        Quantity INT,
        UniquePrice DECIMAL(10, 2),
        PRIMARY KEY (OrderItemId),
        FOREIGN KEY (OrderId) REFERENCES Orders(OrderId) ON DELETE RESTRICT,
        FOREIGN KEY (ProductId) REFERENCES Products(ProductId) ON DELETE RESTRICT,
        CONSTRAINT  UniquePrice CHECK (UniquePrice >= 0),
        CONSTRAINT Quantity CHECK (Quantity > 0));
```

```
CREATE TABLE Recipe (
    IngredientProductId CHAR(10) PRIMARY KEY,
    Amount DECIMAL(10,2),
    CONSTRAINT Amount CHECK (Amount > 0),
    ProductId CHAR(10),
    AddOnProductId CHAR(10),
    IngredientId CHAR(10),
    FOREIGN KEY (IngredientId) REFERENCES Ingredient(IngredientId) ON DELETE RESTRICT,
    FOREIGN KEY (ProductId) REFERENCES Products(ProductId) ON DELETE RESTRICT,
    FOREIGN KEY (AddOnProductId ) REFERENCES Products(ProductId) ON DELETE RESTRICT
);
```

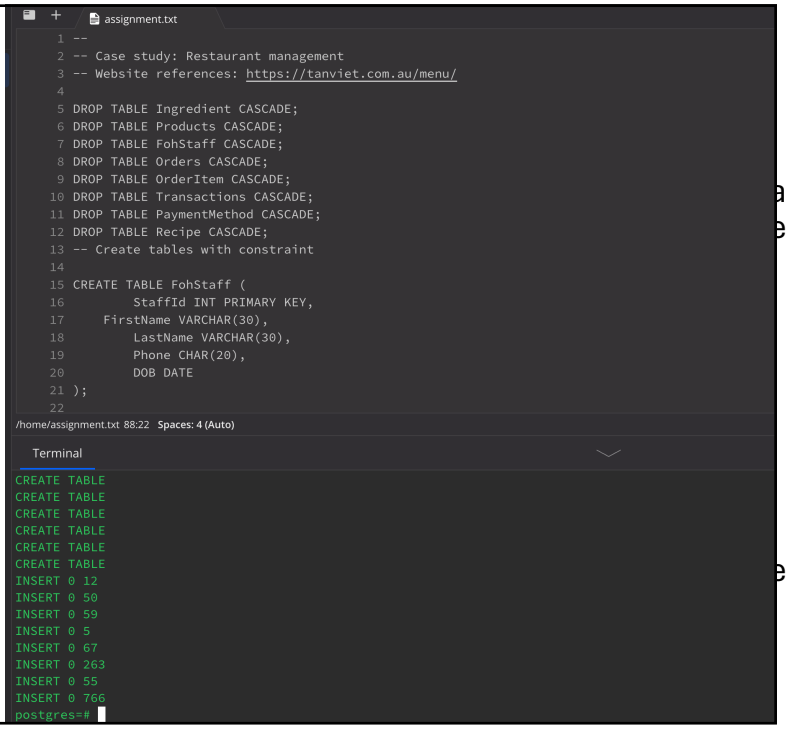
18

ACTION Statement

```
CREATE TABLE Transactions (
    TransactionId CHAR(10) PRIMARY KEY,
        OrderId CHAR(10),
        OrderAmount DECIMAL(10, 2),
        Surcharge DECIMAL(10, 2),
    TotalAmount DECIMAL(10, 2),
        Date DATE,
        Time TIME,
        PaymentMethod VARCHAR(30),
        FOREIGN KEY (OrderId) REFERENCES Orders(OrderId) ON DELETE CASCADE,
        CONSTRAINT  OrderAmount CHECK (OrderAmount > 0),
        CONSTRAINT TotalAmount  CHECK (TotalAmount > 0),
         CONSTRAINT Surcharge  CHECK (Surcharge > 0));
```

19

A view in my SQL



20