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

DATABASE SYSTEMS (CO2013)

Assignment 2

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	Create Tables and Sample Data, Writing Trigger 1 Writing Procedures to Insert, Delete, and Update data from FOOD ITEM table, Latex editor	100%
2	Cao Ngọc Lâm	2252419	Writing Functions for Applications Making Interface Applications to demonstrate procedure and function, Latex editor	100%
3	Hồ Khánh Nam	2252500	Create Tables and Sample Data, Writing Trigger 2 Writing Procedures to Insert, Delete, and Update data from FOOD ITEM table, Latex editor	100%
4	Đặng Ngọc Phú	2252617	Writing trigger for semantic constraints Writing Procedure 1 and Procedure 2, Latex editor	100%
5	Nguyễn Châu Hoàng Long	2252444	Writing Functions for Applications Making Interface Applications to demonstrate procedure and function	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.

3 Create Tables and Sample Data

3.1 Design and Create the Data Table

Requirements

Write the SQL statements to create the tables, including primary keys, foreign keys, data constraints, and semantic constraints specified in Assignment 1 (using `CHECK` or `TRIGGER`).

- For this task, we will create these data tables:
  - **Table Account:**

```sql
CREATE TABLE ACCOUNT (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL
);
```

  - **Table of Customer:**

```sql
CREATE TABLE CUSTOMER (
    ID INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    exchange_points INT DEFAULT 0,
    phone_number VARCHAR(10) UNIQUE NOT NULL,
    address VARCHAR(255) NOT NULL,
    birthdate DATE NOT NULL,
    card_number VARCHAR(30) UNIQUE NOT NULL,
    bank_name VARCHAR(100),
    FOREIGN KEY (ID) REFERENCES ACCOUNT(ID) ON DELETE
        CASCADE
```

```
);
```

– **Table of Restaurant:**

```
CREATE TABLE RESTAURANT (
    ID INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    phone_number VARCHAR(10) UNIQUE NOT NULL,
    address VARCHAR(255) NOT NULL,
    open_hour TIME,
    close_hour TIME,
    FOREIGN KEY (ID) REFERENCES ACCOUNT(ID) ON DELETE
        CASCADE
);
```

– **Table of Discount From The Restaurant:**

```
CREATE TABLE DISCOUNT_FROM_RESTAURANT (
    code VARCHAR(100) UNIQUE PRIMARY KEY,
    usageLimit INT,
    used_Count INT NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    value INT NOT NULL,
    RID INT NOT NULL,
    FOREIGN KEY (RID) REFERENCES RESTAURANT(ID),
    CHECK (value > 0),
    CHECK (usageLimit >= 0 AND used_Count <= usageLimit),
    CHECK (end_date = DATE_ADD(start_date, interval 7 day)
        )
);
```

– **Table of Discount From Exchange Point:**

```
CREATE TABLE DISCOUNT_FROM_EXCHANGE_POINT (
    code VARCHAR(100) UNIQUE PRIMARY KEY,
    exchange_date DATETIME NOT NULL,
    end_date DATETIME NOT NULL,
    value INT NOT NULL,
    CID INT NOT NULL,
    FOREIGN KEY (CID) REFERENCES CUSTOMER(ID) ON DELETE
        CASCADE,
    CHECK (value > 0)
);
```

– **Table of Category:**

```
CREATE TABLE CATEGORY (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    RID INT NOT NULL,
```

```
    FOREIGN KEY (RID) REFERENCES RESTAURANT(ID) ON DELETE
        CASCADE
);
```

– **Table of Food Item:**

```
CREATE TABLE FOOD_ITEM (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    price INT NOT NULL,
    image VARCHAR(256),
    categoryID INT NOT NULL,
    FOREIGN KEY (categoryID) REFERENCES CATEGORY(ID) ON
        DELETE CASCADE,
    CHECK (price > 0)
);
```

– **Table of Food Deliver:**

```
CREATE TABLE FOOD_DELIVER (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    phone_number VARCHAR(20) UNIQUE NOT NULL,
    vehicle_number VARCHAR(50) UNIQUE NOT NULL,
    name VARCHAR(255) NOT NULL
);
```

– **Table of Food Order:**

```
CREATE TABLE FOOD_ORDER (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    Payment_method ENUM('CARD', 'CASH'),
    address VARCHAR(255) NOT NULL,
    order_status ENUM('pending', 'confirmed', 'delivered',
        'cancelled') DEFAULT 'pending',
    order_time_stamp TIMESTAMP NOT NULL,
    review TEXT,
    rating INT,
        -- gia tong
        orderCost INT NOT NULL DEFAULT 0,
    ship_status ENUM('delivering', 'delivered'),
    get_status ENUM('in_process', 'finished', 'cancelled')
        ,
    delivery_fee INT NOT NULL,
    CID INT NOT NULL,
    RID INT NOT NULL,
    FDID INT NOT NULL,
    FOREIGN KEY (CID) REFERENCES CUSTOMER(ID) ON DELETE
        CASCADE,
    FOREIGN KEY (RID) REFERENCES RESTAURANT(ID) ON DELETE
        CASCADE,
```

```
    FOREIGN KEY (FDID) REFERENCES FOOD_DELIVER(ID) ON
       DELETE CASCADE,
    CHECK (rating >= 1 AND rating <= 5),
    CHECK (delivery_fee >= 0)
);
```

– **Table of 'use':**

```
CREATE TABLE `use` (
    CID INT NOT NULL,
    DFRcode VARCHAR(100) NOT NULL,
    PRIMARY KEY (CID, DFRcode),
    FOREIGN KEY (CID) REFERENCES CUSTOMER(ID) ON DELETE
       CASCADE,
    FOREIGN KEY (DFRcode) REFERENCES
       DISCOUNT_FROM_RESTAURANT(code) ON DELETE CASCADE
);
```

– **Table of 'contain':**

```
CREATE TABLE `contain` (
    FOID INT NOT NULL,
    FIID INT NOT NULL,
    quantity INT NOT NULL,
    PRIMARY KEY (FOID, FIID),
    FOREIGN KEY (FOID) REFERENCES FOOD_ORDER(ID) ON DELETE
        CASCADE,
    FOREIGN KEY (FIID) REFERENCES FOOD_ITEM(ID) ON DELETE
       CASCADE,
    CHECK (quantity > 0)
);
```

– **Table of apply_for:**

```
CREATE TABLE apply_for (
    DFRcode VARCHAR(100) NOT NULL,
    FOID INT NOT NULL,
    PRIMARY KEY (DFRcode, FOID),
    FOREIGN KEY (DFRcode) REFERENCES
       DISCOUNT_FROM_RESTAURANT(code) ON DELETE CASCADE,
    FOREIGN KEY (FOID) REFERENCES FOOD_ORDER(ID) ON DELETE
        CASCADE
);
```

– **Table of discount_point_apply_for:**

```
CREATE TABLE discount_point_apply_for (
    DFEPcode VARCHAR(100) NOT NULL,
    FOID INT NOT NULL,
    PRIMARY KEY (DFEPcode, FOID),
```

```
    FOREIGN KEY (DFEPcode) REFERENCES
       DISCOUNT_FROM_EXCHANGE_POINT(code) ON DELETE
       CASCADE,
    FOREIGN KEY (FOID) REFERENCES FOOD_ORDER(ID) ON DELETE
        CASCADE
);
```

3.2 Create Sample Data

Requirements

Insert meaningful data for all tables (you can enter data through the interface or write SQL statements).

- Here are the SQL statements and screenshots of the data tables that our team
  - **Insert data into Account:**

ID	username	password
1	hknam	Nam12345678
2	taminh	Minh12345678
3	cnlam	Lam12345678
4	nchlong	Long12345678
5	dnphu	Phu12345678
6	banhmihuynhhoa	HuynhHoa12345678
7	comtamPLT	PLT12345678
8	bunthitnuongHang	bunthitnuongHang123456
9	phoVietNam	phoVietNam123456
10	pizzaCompany	pizzacompany123456
11	JDoe	14564556456
12	JSmith	456489456
13	AliceNguyen	afafafafadfd
14	NhahangHoangTam	1465456456
15	DimTuTac	546579841
16	SanFoulu	8789632123
17	BotChien	3215648974
18	PizzaHut	p56456231231

Figure 1: Sample data for Account

– **Insert data into Customer:**

ID	name	email	exchange_points	phone_number	address	birthdate	card_number	bank_name
1	Hồ Khánh Nam	hknam@gmail.com	100	0913133811	123 Đường Nam Kỳ Khởi Nghĩa, Q.1, TP.HCM	2004-08-15	1111222233334444	Vietcombank
2	Trịnh Anh Minh	taminh@yahoo.com	60	0922222222	456 Đường Cách Mạng Tháng 8, Q.3, TP.HCM	2004-11-05	5555666677778888	none
3	Cao Ngọc Lâm	cnlam@hotmail.com	80	0933764233	789 Đường Lê Văn Sỹ, Q.3, TP.HCM	2004-05-20	9999000011112222	Agribank
4	Nguyễn Châu Hoàng Long	nchlong@gmail.com	0	0944568144	321 Đường Nguyễn Thị Minh Khai, Q.1, TP.HCM	2004-07-12	3333444455556666	BIDV
5	Đặng Ngọc Phú	dnphu@outlook.com	200	0955442555	654 Đường Hai Bà Trưng, Q.1, TP.HCM	2004-03-25	7777888899990000	none
11	John Doe	john@example.com	100	0913133691	69 Le Loi, Q.1, TP.HCM	1990-01-01	3123123123213213	BIDV
12	Jane Smith	jane@example.com	60	0995133811	456 Nguyen Canh Chan, Q.1, TP.HCM	1992-06-15	321321312321321	TMCP
13	Alice Nguyen	alice@example.com	80	0913133631	12 Nguyen Thuong Hien, Q.10, TP.HCM	1988-03-22	32232352546452324	ACB

Figure 2: Sample data for Customer

– **Insert data into Restaurant:**

ID	name	phone_number	address	open_hour	close_hour
6	Bánh Mì Huynh Hoa	0966666666	26 Lê Thị Riêng, Q.1, TP.HCM	06:00:00	22:00:00
7	Cơm Tấm Phúc Lộc Thọ	0977777777	345 Nguyễn Trãi, Q.5, TP.HCM	08:00:00	22:00:00
8	Bún Thịt Nướng Hằng	0988888888	12 Nguyễn Thị Nghĩa, Q.1, TP.HCM	08:00:00	20:00:00
9	Phở Việt Nam	0999999999	56 Cao Thắng, Q.3, TP.HCM	06:30:00	17:00:00
10	Pizza Company	1900633606	207-209 Bàu Cát, Phường 13, Tân Bình, Hồ Chí ...	08:00:00	22:00:00
14	NhahangHoangTam	0966666667	26 Lý Thường Kiệt, Q.10, TP.HCM	06:00:00	22:00:00
15	DimTuTac	0977777778	345 Trần Hưng Đạo, Q.5, TP.HCM	08:00:00	22:00:00
16	SanFoulu	0988888889	12 Huyền Trân Công Chúa, Q.1, TP.HCM	08:00:00	20:00:00
17	BotChien	0999999991	56 Lý Chính Thắng, Q.3, TP.HCM	06:30:00	17:00:00
18	PizzaHut	1900633604	207-209 Bà Huyện Thanh Quan, Q.3, Hồ Chí Minh	08:00:00	22:00:00

Figure 3: Sample data for Restaurant

– **Insert data into Discount From The Restaurant:**

code	usageLimit	used_Count	start_date	end_date	value	RID
DIS_BotChien10	20	1	2024-12-01	2024-12-08	10000	17
DIS_BTN20	30	5	2024-12-01	2024-12-08	20000	8
DIS_COMPANY1	20	5	2024-11-30	2024-12-07	20000	10
DIS_DIMTUTAC10	50	0	2024-12-01	2024-12-08	10000	15
DIS_HH5	100	11	2024-12-01	2024-12-08	5000	6
DIS_HT5	100	0	2024-12-01	2024-12-08	5000	14
DIS_PLT10	10	8	2024-12-01	2024-12-08	10000	7
DIS_PVN10	20	1	2024-12-01	2024-12-08	10000	9
DIS_SanFoulu20	30	0	2024-12-01	2024-12-08	20000	16
DIS_SPECIAL	20	0	2024-12-01	2024-12-08	20000	6
DIS_SPECIAL2	20	4	2024-12-01	2024-12-08	20000	18

Figure 4: Sample data for Discount From The Restaurant

– **Insert data into Discount From The Exchange Point:**

code	exchange_date	end_date	value	CID
EXC_ALICE40	2024-12-01 00:00:00	2024-12-08 23:59:59	15000	13
EXC_DOE20	2024-12-01 00:00:00	2024-12-08 23:59:59	5000	11
EXC_LAM40	2024-12-01 00:00:00	2024-12-08 23:59:59	15000	3
EXC_MINH80	2024-12-01 00:00:00	2024-12-08 23:59:59	30000	2
EXC_NAM20	2024-12-01 00:00:00	2024-12-08 23:59:59	5000	1
EXC_PHU120	2024-12-01 00:00:00	2024-12-08 23:59:59	50000	5
EXC_SMITH80	2024-12-01 00:00:00	2024-12-08 23:59:59	30000	12

Figure 5: Sample data for Discount From The Exchange Point

– **Insert data into Category:**

ID	name	RID
1	Bánh Mì	6
2	Cơm	7
3	Bún	8
4	Phở	9
5	Nước uống	6
6	Nước uống	7
7	Nước uống	8
8	Nước uống	9
9	pizza	10
10	mì	10
11	Bánh	14
12	Mì	15
13	Dimsum	16
14	Đồ Chiên	17
15	Pizza	18
16	Gà	18
17	Mì	18
18	Salad	18
19	Khoai tây ...	18
20	Nước uống	18
21	Bánh	6
22	Bơ	6
23	Chà bông	6
24	Cơm	6
25	Chả lụa	6
26	Pate	6
27	Dăm bông	6

Figure 6: Sample data for Category

– **Insert data into Food Item:**

ID	name	price	image	categoryID
1	Bánh Mì Thập Cẩm	68000	https://phapluatbanquyen.phaply.vn/uploads/i...	1
2	Bánh Mì Ốp La	30000	https://img-global.cpcdn.com/recipes/01914f4b...	1
3	Bánh Mì Phá Lấu	40000	https://media-cdn-v2.laodong.vn/storage/news...	1
4	Cơm mực dồn thịt	90000	https://comtamthuankieu.com.vn/wp-content/u...	2
5	Cơm Tấm Sườn	60000	https://comtamthuankieu.com.vn/wp-content/u...	2
6	Cơm bì chả	45000	https://cdn.tgdd.vn/2020/08/CookProduct/52-...	2
7	Cơm ba rọi nướng	60000	https://thenguyen.vn/files/products/product_1...	2
8	Cơm sườn cọng nướng	80000	https://comtamthuankieu.com.vn/wp-content/u...	2
9	Bún Thịt Nướng Đặc Biệt	40000	https://encrypted-tbn0.gstatic.com/images?q=...	3
10	Phở Tái	45000	https://bizweb.dktcdn.net/thumb/grande/100/4...	4
11	Coca	20000	https://encrypted-tbn0.gstatic.com/images?q=...	7
12	Trà đá	10000	https://image.tienphong.vn/w890/Uploaded/20...	8
13	Pizza hải sản pesto	169000	http://thepizzacompany.vn/images/thumbs/000...	9
14	Mỳ Ý sốt kem cà chua	139000	http://thepizzacompany.vn/images/thumbs/000...	10
15	Bánh xèo	69000	https://www.huongnghiepaau.com/wp-content...	11
16	Mì xào vịt	350000	https://vit29.com/media/news/1502_mtrnvtqua...	12
17	Há cảo	40000	https://tiki.vn/blog/wp-content/uploads/2023/0...	13
18	Bột chiên trứng	30000	https://i.ytimg.com/vi/q7is9fGsGuw/maxresdef...	14
19	Pizza pepperoni	200000	https://file.hstatic.net/1000389344/article/pep...	15
20	Cánh gà giòn Xốt Cay (...	79000	https://cdn.pizzahut.vn/images/Web_V3/Produ...	16
21	Mì Ý bò Bằm Xốt Cà Chua	120000	https://cdn.pizzahut.vn/images/Web_V3/Produ...	17
22	Xà Lách Da Cá Hồi	89000	https://cdn.pizzahut.vn/images/Web_V3/Produ...	15
23	Pizza Hải Sản Nhiệt Đới	229000	https://cdn.pizzahut.vn/images/Web_V3/Produ...	15
24	Pizza Thập Cẩm	229000	https://cdn.pizzahut.vn/images/Web_V3/Produ...	15
25	Pizza Cơn lốc Hải Sản	239000	https://cdn.pizzahut.vn/images/Web_V3/Produ...	15
26	Khoai Tây Chiên	59000	https://cdn.pizzahut.vn/images/Web_V3/Produ...	19
27	Pepsi không calo 320ml	30000	https://cdn.pizzahut.vn/images/Web_V3/Produ...	20
28	Pepsi Lemon 320ml	30000	https://cdn.pizzahut.vn/images/Web_V3/Produ...	20
29	Bánh bao đặc biệt	35000	https://banhmihuynhhoa.vn/wp-content/upload...	1
30	Bơ tươi Huynh Hoa	65000	https://cdn.pizzahut.vn/images/Web_V3/Produ...	1
31	Chả đặc biệt	120000	https://banhmihuynhhoa.vn/wp-content/upload...	1
32	Cơm cháy siêu cay	50000	https://banhmihuynhhoa.vn/wp-content/upload...	1
33	Chả lụa đặc biệt	140000	https://banhmihuynhhoa.vn/wp-content/upload...	1
34	Pate Huynh Hoa	90000	https://banhmihuynhhoa.vn/wp-content/upload...	1
35	Dăm bông vai vuông	140000	https://banhmihuynhhoa.vn/wp-content/upload...	1
36	Pizza Tôm Cocktail	159000	http://thepizzacompany.vn/images/thumbs/000...	9
37	Pizza Hải Sản Nhiệt Đới	159000	http://thepizzacompany.vn/images/thumbs/000...	9
38	Pizza Aloha	149000	http://thepizzacompany.vn/images/thumbs/000...	9
39	Pizza Thịt Xông Khói	149000	http://thepizzacompany.vn/images/thumbs/000...	9
40	Pizza Hải Sản Cao Cấp	149000	http://thepizzacompany.vn/images/thumbs/000...	9
41	Mì Ý Chay Sốt Marinara	109000	http://thepizzacompany.vn/images/thumbs/000...	10

Figure 7: Sample data for Food Item

– **Insert data into Food Deliver:**

ID	phone_number	vehicle_number	name
1	0911223344	59A-12345	Nguyễn Văn Tài
2	0922334455	59B-67890	Nguyễn Minh Mẫn
3	0921134425	59B-66490	Hồ Minh Huy
4	0331134625	60B-66490	Lê Tuấn Anh
5	0335534345	60B-66992	Hà Minh Giang

Figure 8: Sample data for Food Deliver

– **Insert data into The Discount of the restaurant applied to the Food Order:**

ID	Payment_method	address	order_status	order_time_stamp	review	rating	orderCost	ship_status	get_status	delivery_fee	CID	RID	FDID
1	CARD	123 Đường Nam Kỳ Khởi Nghĩa, Q.1, TP.HCM	delivered	2024-12-01 14:30:00	Dịch vụ rất tốt!	5	219000	delivered	finished	15000	1	6	1
2	CASH	456 Đường Cách Mạng Tháng 8, Q.3, TP.HCM	delivered	2024-12-01 14:30:00	Dịch vụ tệ!	1	195000	delivered	finished	15000	2	7	2
3	CASH	789 Đường Lê Văn Sỹ, Q.3, TP.HCM	pending	2024-12-01 14:30:00		5	249000	NULL	NULL	15000	3	6	1
4	CARD	123 Đường Nam Kỳ Khởi Nghĩa, Q.1, TP.HCM	pending	2024-12-01 14:30:00		5	492000	NULL	NULL	15000	1	10	3
5	CARD	69 Le Loi, Q.1, TP.HCM	delivered	2024-02-13 15:31:00	Dịch vụ hoàn hảo!	5	155000	delivered	finished	25000	11	6	4
6	CASH	456 Nguyen Canh Chan, Q.1, TP.HCM	delivered	2024-09-01 10:30:00	Dịch vụ không tốt !	3	720000	delivered	finished	20000	12	15	5
7	CASH	12 Nguyen Thuong Hien, Q.10, TP.HCM	pending	2024-08-24 16:20:00	Dịch vụ oke	5	130000	delivered	finished	10000	13	16	5
8	CARD	654 Đường Hai Bà Trưng, Q.1, TP.HCM	delivered	2024-12-02 18:45:00	Hương vị tuyệt vời!	5	384000	delivered	finished	20000	5	6	1
9	CASH	321 Đường Nguyễn Thị Minh Khai, Q.1, TP.HCM	delivered	2024-12-02 19:00:00	Nhân viên giao hàng thân thiện.	4	435000	delivered	finished	15000	4	7	2
10	CARD	207-209 Bàu Cát, Phường 13, Tân Bình, TP.HCM	delivered	2024-12-03 12:30:00	Pizza ngon!	5	1550000	delivered	finished	20000	1	10	3
11	CASH	12 Nguyễn Thị Nghĩa, Q.1, TP.HCM	delivered	2024-12-03 13:15:00	Bún thịt nướng siêu ngon!	5	135000	delivered	finished	15000	3	8	4
12	CARD	56 Cao Thắng, Q.3, TP.HCM	delivered	2024-12-04 17:00:00	Phở hơi mặn.	3	100000	delivered	finished	10000	11	9	1
13	CARD	345 Nguyễn Trãi, Q.5, TP.HCM	pending	2024-12-04 14:30:00		5	475000	NULL	NULL	15000	2	7	2
14	CASH	26 Lê Thị Riêng, Q.1, TP.HCM	pending	2024-12-04 16:00:00		5	287000	NULL	NULL	15000	12	6	3
15	CARD	56 Lý Chính Thắng, Q.3, TP.HCM	pending	2024-12-04 18:45:00		5	70000	NULL	NULL	10000	13	17	4
16	CASH	789 Đường Lê Văn Sỹ, Q.3, TP.HCM	confirmed	2024-12-01 20:15:00		5	275000	delivering	in_process	15000	4	7	1
17	CARD	207-209 Bà Huyện Thanh Quan, Q.3, TP.HCM	confirmed	2024-12-02 22:00:00		5	355000	delivering	in_process	15000	11	6	5
18	CARD	654 Đường Hai Bà Trưng, Q.1, TP.HCM	confirmed	2024-12-02 21:00:00		5	301000	delivering	in_process	15000	5	6	5
19	CARD	654 Đường Hai Bà Trưng, Q.1, TP.HCM	confirmed	2024-12-02 19:00:00		5	705000	delivering	in_process	15000	5	7	2
20	CARD	654 Đường Hai Bà Trưng, Q.1, TP.HCM	confirmed	2024-12-02 16:00:00		5	135000	delivering	in_process	15000	5	8	1
21	CARD	456 Nguyen Canh Chan, Q.1, TP.HCM	confirmed	2024-12-02 16:00:00		5	945000	delivering	in_process	15000	12	6	3
22	CARD	456 Nguyen Canh Chan, Q.1, TP.HCM	confirmed	2024-12-02 17:00:00		5	94000	delivering	in_process	14000	12	7	4
23	CARD	456 Nguyen Canh Chan, Q.1, TP.HCM	confirmed	2024-12-02 18:30:00		5	95000	delivering	in_process	15000	12	8	5
24	CASH	12 Nguyen Thuong Hien, Q.10, TP.HCM	confirmed	2024-12-02 08:15:00		5	710000	delivering	in_process	15000	13	10	1
25	CASH	12 Nguyen Thuong Hien, Q.10, TP.HCM	confirmed	2024-12-02 09:00:00		5	191000	delivering	in_process	13000	13	18	2
26	CASH	12 Nguyen Thuong Hien, Q.10, TP.HCM	confirmed	2024-12-02 20:00:00		5	183000	delivering	in_process	13000	13	6	2
27	CASH	123 Đường Nam Kỳ Khởi Nghĩa, Q.1, TP.HCM	delivered	2024-12-02 08:15:00		5	241000	delivered	finished	15000	1	6	3
28	CASH	123 Đường Nam Kỳ Khởi Nghĩa, Q.1, TP.HCM	delivered	2024-12-02 09:00:00		5	120000	delivered	finished	15000	1	7	4
29	CASH	123 Đường Nam Kỳ Khởi Nghĩa, Q.1, TP.HCM	delivered	2024-12-02 20:00:00		5	305000	delivered	finished	25000	1	8	5
30	CARD	456 Đường Cách Mạng Tháng 8, Q.3, TP.HCM	confirmed	2024-12-02 16:00:00		5	335000	delivering	in_process	15000	2	6	3
31	CARD	456 Đường Cách Mạng Tháng 8, Q.3, TP.HCM	confirmed	2024-12-02 17:00:00		5	375000	delivering	in_process	15000	2	7	1
32	CARD	456 Đường Cách Mạng Tháng 8, Q.3, TP.HCM	confirmed	2024-12-02 18:30:00		5	95000	delivering	in_process	15000	2	8	1
33	CASH	789 Đường Lê Văn Sỹ, Q.3, TP.HCM	confirmed	2024-12-02 08:15:00		5	522000	delivering	in_process	15000	3	10	1
34	CASH	789 Đường Lê Văn Sỹ, Q.3, TP.HCM	confirmed	2024-12-02 09:00:00		5	462000	delivering	in_process	17000	3	18	2
35	CASH	789 Đường Lê Văn Sỹ, Q.3, TP.HCM	confirmed	2024-12-02 20:00:00		5	385000	delivering	in_process	15000	3	6	3
36	CASH	207-209 Bàu Cát, Phường 13, Tân Bình, TP.HCM	confirmed	2024-12-02 08:15:00		5	799000	delivering	in_process	14000	4	10	1
37	CASH	207-209 Bàu Cát, Phường 13, Tân Bình, TP.HCM	confirmed	2024-12-02 09:00:00		5	194000	delivering	in_process	16000	4	18	5
38	CASH	207-209 Bàu Cát, Phường 13, Tân Bình, TP.HCM	confirmed	2024-12-02 20:00:00		5	430000	delivering	in_process	16000	4	6	5

Figure 9: Sample data for apply_for

– **Insert data into the contain of the food order:**

FOID	FIID	quantity
1	1	3
2	4	2
3	1	3
3	2	1
4	13	2
4	14	1
5	30	2
6	16	2
7	17	3
8	1	3
8	2	4
8	3	1
9	4	3
9	5	1
9	6	2
10	36	2
10	37	2
10	38	1
10	39	5
11	9	3
12	10	2
13	7	1
13	8	5
14	1	4
15	18	2
16	7	3
16	8	1
17	1	5
18	1	2
18	2	5
19	4	5
19	5	4
20	9	3
21	32	2
21	33	4
21	34	3
22	8	1
23	9	2
24	14	5
25	22	2
26	29	3
26	30	1
27	1	2
27	2	3

28	5	1
28	6	1
29	9	7
30	2	4
30	3	5
31	5	2
31	8	3
32	9	2
33	13	3
34	22	5
35	32	2
35	34	3
36	13	3
36	14	2
37	22	2
38	1	3
38	2	3
38	3	3

Figure 10: Sample data for contain

– **Insert data that discount code of the restaurant applied for the food order:**

DFRcode	FOID
DIS_HH5	1
DIS_PLT10	2
DIS_HH5	3
DIS_COMPANY1	4
DIS_HH5	5
DIS_PLT10	9
DIS_COMPANY1	10
DIS_BTN20	11
DIS_PVN10	12
DIS_PLT10	13
DIS_HH5	14
DIS_BotChien10	15
DIS_PLT10	16
DIS_SPECIAL2	17
DIS_HH5	18
DIS_PLT10	19
DIS_BTN20	20
DIS_HH5	21
DIS_PLT10	22
DIS_BTN20	23
DIS_COMPANY1	24
DIS_SPECIAL2	25

DIS_HH5	26
DIS_HH5	27
DIS_PLT10	28
DIS_BTN20	29
DIS_HH5	30
DIS_PLT10	31
DIS_BTN20	32
DIS_COMPANY1	33
DIS_SPECIAL2	34
DIS_HH5	35
DIS_COMPANY1	36
DIS_SPECIAL2	37
DIS_HH5	38

Figure 11: Sample data for apply_for

3.3 Trigger for Semantic Constraints

3.3.1 Trigger 1

- **Description**: The **check_food_order_restaurant** trigger ensures data consistency by validating that the food item being added to an order belongs to the same restaurant as the order itself. Before inserting a new row into the contain table, the trigger checks if the restaurant ID associated with the food item matches the restaurant ID of the order. If not, it raises an error to prevent the insertion, maintaining the integrity of restaurant-specific orders.

- **Query**:

```
DELIMITER $$

CREATE TRIGGER check_food_order_restaurant
BEFORE INSERT ON food_ordering_db.contain
FOR EACH ROW
BEGIN
    DECLARE res_id INT;
    -- Retrieve the restaurant ID of the food order
    SELECT RID INTO res_id FROM food_ordering_db.food_order
        WHERE ID = NEW.FOID;

    -- Check if the restaurant ID of the FOOD_ITEM matches the
        FOOD_ORDER
    IF NOT EXISTS (
        SELECT 1
        FROM food_ordering_db.food_item FI
        JOIN food_ordering_db.category C ON FI.categoryID = C.
            ID
        WHERE FI.ID = NEW.FIID AND C.RID = res_id
    ) THEN
```

```
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Food item does not belong to the
            same restaurant as the food order.';
    END IF;
END$$

DELIMITER ;
```

- **Testing the Query**:

```
-- Insert data into `contain`
INSERT INTO `contain` (FOID, FIID, quantity)
VALUES
(4, 13, 2),
(4, 14, 1);

INSERT INTO `contain` (FOID, FIID, quantity)
VALUES
(4, 15, 1);
```

- **Test Result**:

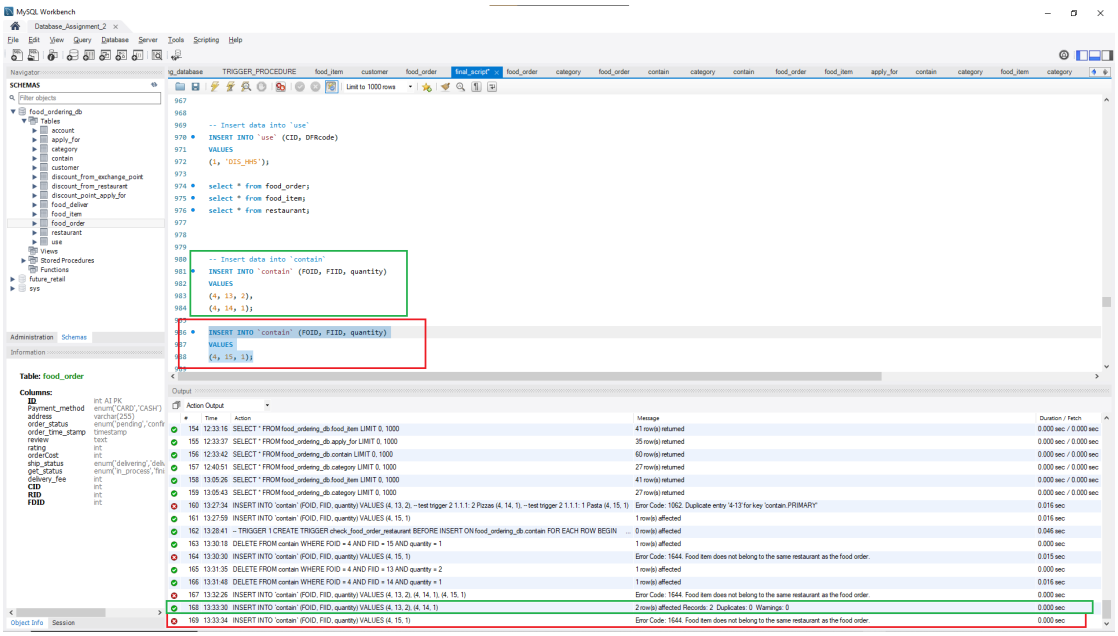


Figure 12: Inserting Food Item and Food Order into the contain table

Food items 13 and 14 are from the same restaurant, so they can be inserted successfully. However, food item 15 is from another restaurant, so it cannot be inserted into the same order.

3.3.2 Trigger 2

- **Description**: The **check_restaurant_discount_limit** trigger enforces a rule to ensure that restaurant discounts do not exceed 30% of the total price of a food order. Before inserting a new row into the APPLY_FOR table, the trigger calculates the total price of the order and retrieves the discount value. If the discount exceeds the allowed limit, it raises an error to prevent the insertion, maintaining fair discount policies and financial consistency.

- **Query**:

```
DELIMITER $$

CREATE TRIGGER check_restaurant_discount_limit
BEFORE INSERT ON APPLY_FOR
FOR EACH ROW
BEGIN
    DECLARE total_order_price INT;
    DECLARE restaurant_discount INT;

    -- Calculate the total price of the food order
    SELECT SUM(FI.price * C.quantity)
    INTO total_order_price
    FROM CONTAIN C
    JOIN FOOD_ITEM FI ON C.FIID = FI.ID
    WHERE C.FOID = NEW.FOID;

    -- Get the restaurant discount value
    SELECT DFR.value
    INTO restaurant_discount
    FROM DISCOUNT_FROM_RESTAURANT DFR
    WHERE DFR.code = NEW.DFRcode;


    -- Check if the total discount exceeds 30% of the total
       price
    IF restaurant_discount > total_order_price * 0.30 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'The discount exceeds 30% of the
           total food order price.';
    END IF;
END$$

DELIMITER ;
```

- **Testing the Query**:

```
INSERT INTO `contain` (FOID, FIID, quantity)
VALUES
(2, 2, 1);
```

```
INSERT INTO `apply_for` (DFRcode, FOID)
VALUES
('DIS_SPECIAL', 2);
```

- **Test Result**:

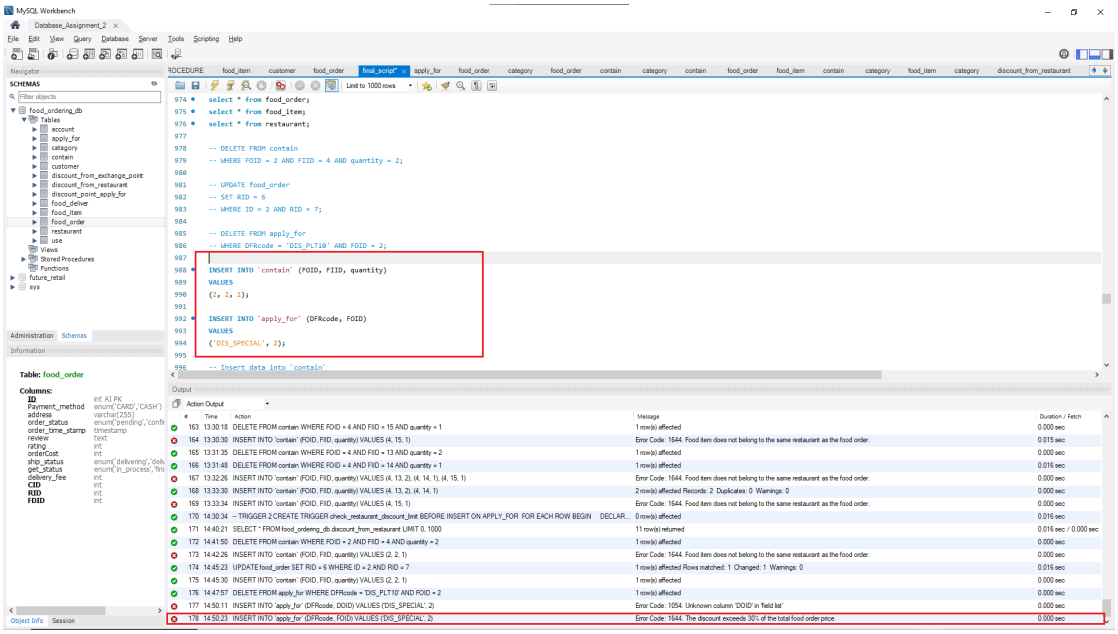


Figure 13: Inserting Restaurant Discount Code and Food Order into the apply_for table

When food order ID = 2 is created, we insert food item ID = 2 (which represents 'banh mi') with a quantity of 1, and the price of the food item is 30,000 VND. Therefore, the total price of order ID = 2 will be 30,000 VND. When we apply the discount code DIS_SPECIAL, which has a value of 20,000 VND, to order ID = 2, the discount exceeds 30% of the total price of the order, resulting in an error being thrown.

4 Implement Applications

4.1 Procedures to Insert, Delete, and Update data from FOOD ITEM table

4.1.1 Insert Procedure

- **Description:** The AddFoodItem procedure is used when we want to add new food item from the restaurant into the database. The procedure uses name, price, image of the food and the category ID as the parameters and it checks the input validation of the passing parameters. The validation checks include:
  - If the name of the food is empty, the database management system will show validation errors: "The food name field cannot be blank"
  - If the price is not greater than 0, the database management system show validation errors: "The food price must be greater 0"
  - If the category ID does not exist in the ‘Category‘ table, then the validation errors will be shown: "The new category has not been added"
- **Query:**

```
DELIMITER $$

CREATE PROCEDURE AddFoodItem(
        IN namefood VARCHAR(255),
    IN pricefood INT,
    IN imagefood VARCHAR(255),
    IN cateId INT
)
BEGIN
        IF (namefood = "") THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = '
                   The food name field cannot be blank';
    ELSEIF (pricefood <= 0) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'The food
           price must be greater 0';
    ELSEIF (NOT EXISTS(SELECT 1 FROM CATEGORY WHERE ID =
       cateId)) THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = '
                   The new category has not been added';
    ELSE
        INSERT INTO FOOD_ITEM (
            name, price, image, categoryID
        ) VALUES (
            namefood, pricefood, imagefood, cateId
        );
    END IF;
END$$

DELIMITER ;
```

- **Testing the query:**

```
CALL AddFoodItem("Com tam suon bi cha", 70000, 'https://
    product.hstatic.net/200000523823/product/
    com_tam_suon_bi_cha_c50c564871254d7ea56b6d82344ae9bf_master
    .jpg',2);
```

- **Test Result:**

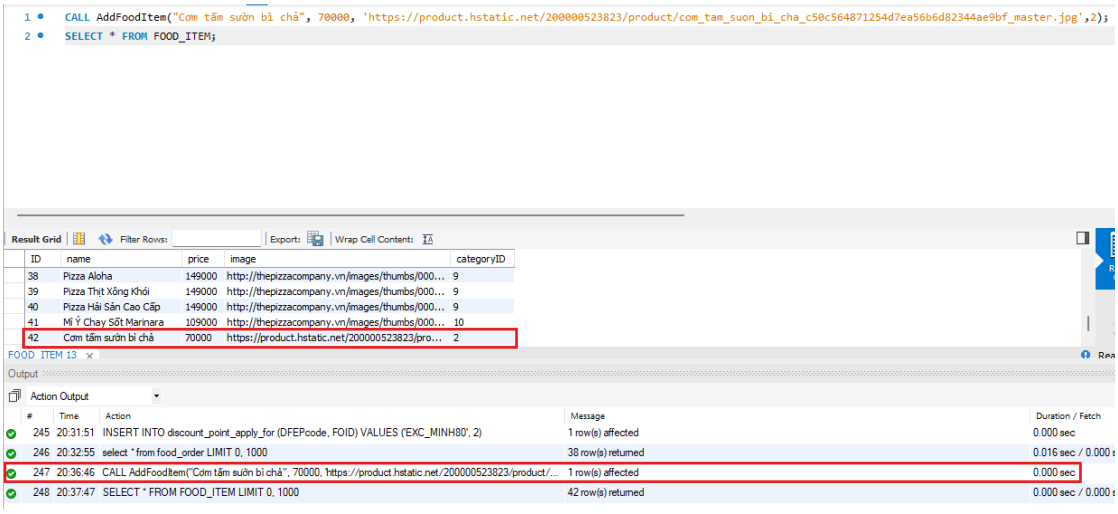


Figure 14: AddFoodItem procedure for inserting new food item

4.1.2 Delete Procedure

- **Description:** The DeleteFoodItem procedure is used when we want to delete the food item from the restaurant in the database. The procedure use Food item ID as the parameter, and it checks whether the food item exists in the FOOD_ITEM table. If it is not in the table, the database system will show the error: "The food item does not exist".

- **Query:**

```
DELIMITER $$

CREATE PROCEDURE DeleteFoodItem(
        IN idfood INT
)
BEGIN
    IF NOT EXISTS (SELECT 1 FROM FOOD_ITEM WHERE ID = idfood)
       THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'The food
           item does not exist';
    ELSE
        DELETE FROM FOOD_ITEM WHERE ID = idfood;
```

```
    END IF;
END$$

DELIMITER ;
```

- **Testing the Query:**

```
CALL DeleteFoodItem(7);
```

- **Test Result:**

```
select * from food_item;
```

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

ID	name	price	image	categoryID
5	Cơm Tấm Sườn	60000	https://comtamthuankieu.com.vn/wp-content/u...	2
6	Cơm bì chả	45000	https://cdn.tgdd.vn/2020/08/CookProduct/52-...	2
7	Cơm ba rọi nướng	60000	https://thenguyen.vn/files/products/product_1...	2
8	Cơm sườn cọng nướng	80000	https://comtamthuankieu.com.vn/wp-content/u...	2
9	Bún Thịt Nướng Đặc Biệt	40000	https://encrypted-tbn0.gstatic.com/images?q=...	3

food_item 19

Output

Action Output

#	Time	Action	Message
365	21:51:21	INSERT INTO `contain` (FOID, FIID, quantity) VALUES (1, 1, 3), (2, 4, 2), (4, 13, 2), -- test trigger 2 1.1.1...	60 row(s) affected Records: 60 Duplicates: 0 Warnings: 0
366	21:51:21	INSERT INTO apply_for (DFRcode, FOID) VALUES ('DIS_HH5', 1), ('DIS_HH5', 3), ('DIS_HH5', 5), ('DIS...	35 row(s) affected Records: 35 Duplicates: 0 Warnings: 0
367	21:51:21	INSERT INTO discount_point_apply_for (DFEPcode, FOID) VALUES ('EXC_MINH80', 2)	1 row(s) affected
368	21:51:25	select * from food_item LIMIT 0, 1000	41 row(s) returned

Figure 15: The 'FOOD_ITEM' table before using DeleteFoodItem procedure

```
1 •  CALL DELETEFOODITEM(7);
2
3 •  select * from food_item;
```

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

ID	name	price	image	categoryID
4	Cơm mực dồn thịt	90000	https://comtamthuankieu.com.vn/wp-content/u...	2
5	Cơm Tấm Sườn	60000	https://comtamthuankieu.com.vn/wp-content/u...	2
6	Cơm bì chả	45000	https://cdn.tgdd.vn/2020/08/CookProduct/52-...	2
8	Cơm sườn cọng nướng	80000	https://comtamthuankieu.com.vn/wp-content/u...	2
9	Bún Thịt Nướng Đặc Biệt	40000	https://encrypted-tbn0.gstatic.com/images?q=...	3

food_item 20

Output

Action Output

#	Time	Action	Message
367	21:51:21	INSERT INTO discount_point_apply_for (DFEPcode, FOID) VALUES ('EXC_MINH80', 2)	1 row(s) affected
368	21:51:25	select * from food_item LIMIT 0, 1000	41 row(s) returned
369	21:52:39	CALL DELETEFOODITEM(7)	1 row(s) affected
370	21:52:39	select * from food_item LIMIT 0, 1000	40 row(s) returned

Figure 16: The ‘FOOD_ITEM‘ table after using DeleteFoodItem procedure

4.1.3 Update Procedure

- **Description:** The UpdateFoodItem procedure is used when we want to update the food item information in the database. The procedure uses the food item ID, the food name, price, image, and category ID as the parameters, and it checks the input validation:
  - If the food item does not exist in the ‘FOOD_ITEM‘ table, then the database show the error: "The food item does not exists"
  - If the price of the food is not greater than 0, the database show the error: "The food price must be greater 0"
  - If the category data (corresponding to the category ID) does not exist in the CATEGORY table, the database show the error: "The category does not exist"
- **Query:**

```
DELIMITER $$

CREATE PROCEDURE UpdateFoodItem(
        IN idfood INT,
        IN namefood VARCHAR(255),
```

```
    IN pricefood INT,
    IN imagefood VARCHAR(255),
    IN cateID INT
)
BEGIN
    IF NOT EXISTS (SELECT 1 FROM FOOD_ITEM WHERE ID = idfood)
       THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'The food
           item does not exists';
    ELSEIF (pricefood <= 0 ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'The food
           price must be greater 0';
        ELSEIF NOT EXISTS (SELECT 1 FROM CATEGORY WHERE ID =
           cateID) THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = '
                   The category does not exist';
    ELSE
                UPDATE FOOD_ITEM
        SET name = namefood, price = pricefood, image =
           imagefood, categoryID = cateID
        where ID = idfood;
        END IF;
END$$

DELIMITER ;
```

- **Testing the query:** Update the name and the price of the food item with ID = 1

```
CALL UpdateFoodItem(1, 'Banh Mi Day Du', 70000, 'https://
   nghenghiepcuocsong.vn/wp-content/uploads/2023/01/
   z4025571501634_d4354c69e24aa7d1ee884433db8b4763.jpg', 6);
```

- **Test Result:**

```
1 •  SELECT * FROM FOOD_ITEM;
```

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

ID	name	price	image	categoryID
1	Bánh Mì Thập Cẩm	68000	https://phapluatbanquyen.phaply.vn/uploads/i...	1
2	Bánh Mì Op La	30000	https://img-global.cpcdn.com/recipes/01914f4b...	1
3	Bánh Mì Phá Lấu	40000	https://media-cdn-v2.laodong.vn/storage/news...	1
4	Cơm mực dồn thịt	90000	https://comtamthuankieu.com.vn/wp-content/u...	2
5	Cơm Tấm Sườn	60000	https://comtamthuankieu.com.vn/wp-content/u...	2

Figure 17: The 'FOOD_ITEM' table before using UpdateFoodItem procedure

```
1 •  CALL UpdateFoodItem(1, 'Bánh Mì Đầy đủ', 70000, 'https://nghenghiepcuocsong.vn/wp-content/uploads/2023/01/z4025571501634_d4354c69e24aa7d1ee884433db8b4763.jpg', 6);
2
3 •  SELECT * FROM FOOD_ITEM;
4
```

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

ID	name	price	image	categoryID
1	Bánh Mì Đầy đủ	70000	https://nghenghiepcuocsong.vn/wp-content/up...	6
2	Bánh Mì Op La	30000	https://img-global.cpcdn.com/recipes/01914f4b...	1
3	Bánh Mì Phá Lấu	40000	https://media-cdn-v2.laodong.vn/storage/news...	1
4	Cơm mực dồn thịt	90000	https://comtamthuankieu.com.vn/wp-content/u...	2
5	Cơm Tấm Sườn	60000	https://comtamthuankieu.com.vn/wp-content/u...	2

FOOD_ITEM 22

Output

Action Output

#	Time	Action	Message	Duration / Fetch
427	22:13:09	INSERT INTO discount_point_apply_for (DFEPcode, FOID) VALUES ('EXC_MINH80', 2)	1 row(s) affected	0.015 sec
428	22:14:06	SELECT * FROM FOOD_ITEM LIMIT 0, 1000	41 row(s) returned	0.000 sec / 0.0
429	22:14:42	CALL UpdateFoodItem(1, 'Bánh Mì Đầy đủ', 70000, 'https://nghenghiepcuocsong.vn/wp-content/upload...	1 row(s) affected	0.016 sec
430	22:14:51	SELECT * FROM FOOD_ITEM LIMIT 0, 1000	41 row(s) returned	0.000 sec / 0.0

Figure 18: The 'FOOD_ITEM' table after using UpdateFoodItem procedure

4.2 Writing Triggers

4.2.1 Trigger 1

- **Description:** using trigger to update the exchange points of the customer after the order is finished. Specifically, it checks whether if the order status changes from 'confirmed' to 'delivered', the exchange point of the customer who makes that food order will increase by 20 points

- **Query:**

```
DELIMITER $$

CREATE TRIGGER update_customer_exchange_point
AFTER UPDATE ON FOOD_ORDER
FOR EACH ROW
BEGIN

    IF NEW.order_status = 'delivered' AND OLD.order_status = '
       confirmed' THEN

        UPDATE CUSTOMER
        SET exchange_points = COALESCE(exchange_points, 0) +
           20
        WHERE ID = NEW.CID;
    END IF;
END$$

DELIMITER ;
```

- **Testing the query:**

```
INSERT INTO FOOD_ORDER (Payment_method, address, order_status,
    order_time_stamp, review, rating, ship_status, get_status,
    delivery_fee, CID, RID, FDID)
VALUES
('CASH', '123 Ba Huyen Thanh Quan , Q.3, TP.HCM', 'confirmed',
   '2024-12-01 14:30:00', 'Dich vu te!', 1, 'delivered', '
   finished', 15000, 4, 9, 2);

UPDATE FOOD_ORDER
SET order_status = 'delivered'
WHERE ID = 39;
```

- **Test result:**

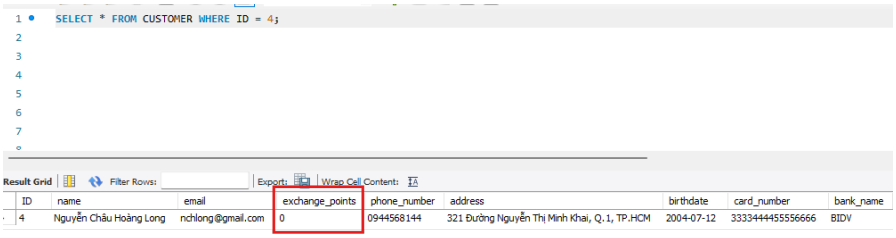


Figure 19: The customer with ID = 4 before updating food order with id = 39

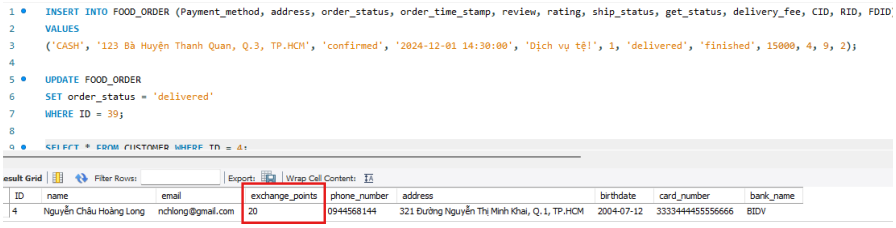


Figure 20: The customer with ID = 4 after updating food order with id = 39

4.2.2 Trigger 2

- **Description:** using trigger to update the total cost to pay (not including the discount) for the order after inserting new ‘contain‘ data (this will simulate the function used when the customer presses "Add to Cart", the payment will show the total cost of the order).

- **Query:**

```
DELIMITER $$

CREATE TRIGGER update_order_cost_after_insert_contain
AFTER INSERT ON ‘contain‘
FOR EACH ROW
BEGIN
        DECLARE totalFoodCost INT DEFAULT 0;
        SET totalFoodCost = (
            SELECT SUM(FI.price * C.quantity)
            FROM FOOD_ITEM FI, contain C
            WHERE FI.ID = C.FIID AND C.FOID = NEW.FOID
        );
        UPDATE FOOD_ORDER
        SET orderCost = totalFoodCost + delivery_fee
        WHERE ID = NEW.FOID;
END$$
```

```
DELIMITER ;
```

- **Testing the query:**

```
INSERT INTO `contain` (FOID, FIID, quantity)
values
(3, 1, 2),
(3, 2, 1);
```

- **Test Result:**

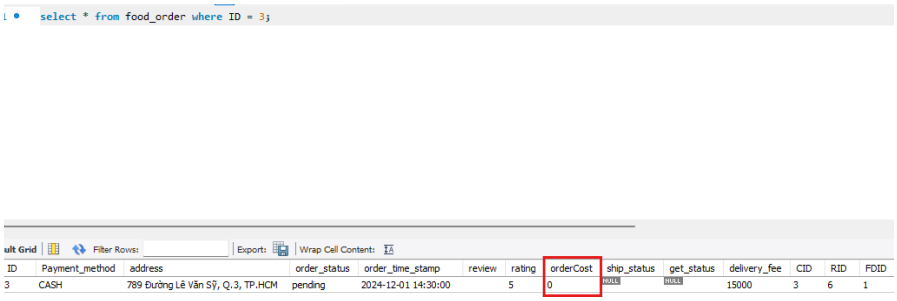


Figure 21: The food order with ID = 3 before inserting data into ‘contain‘

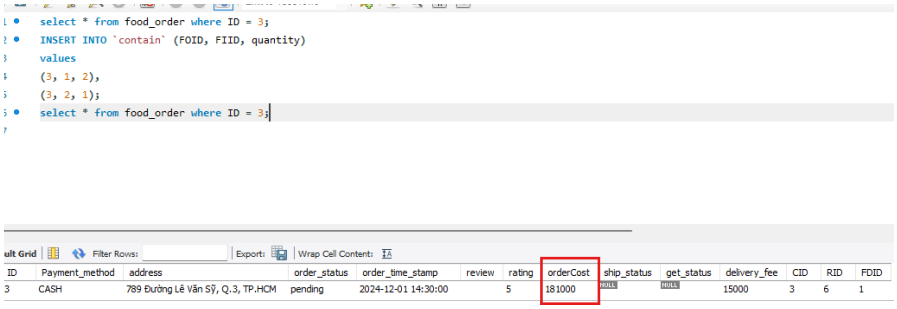


Figure 22: The food order with ID = 3 after inserting data into ‘contain‘

4.3 Writing Procedures

4.3.1 Procedure 1

- **Description**: The **RetrieveOrdersByCustomerWithDiscounts** procedure retrieves detailed information about orders placed by a specific customer, including order details, total price, and discounts applied. It calculates two types of discounts—restaurant discounts and exchange point discounts—and provides their sum as the total discount. The procedure only includes orders with at least one applied discount and aggregates the total item price for each order. This is useful for generating order summaries with pricing and discount details.

- **Query**:

```
DELIMITER //

CREATE PROCEDURE RetrieveOrdersByCustomerWithDiscounts(
    IN p_CustomerID INT
)
BEGIN
    SELECT
        FO.ID AS OrderID,
        FO.Payment_method,
        FO.address AS OrderAddress,
        FO.order_status AS OrderStatus,
        FO.order_time_stamp AS OrderTimestamp,
        FO.review,
        FO.rating,
        FO.ship_status AS ShippingStatus,
        FO.get_status AS GetStatus,
        FO.delivery_fee,
        SUM(FI.price * C.quantity) AS TotalOrderPrice,

        -- Calculate restaurant discount
        COALESCE((
            SELECT DFR.value
            FROM DISCOUNT_FROM_RESTAURANT DFR
            JOIN APPLY_FOR A ON DFR.code = A.DFRcode
            WHERE A.FOID = FO.ID
        ), 0) AS RestaurantDiscount,

        -- Calculate exchange point discount as a fixed value
        COALESCE((
            SELECT DFEP.value
            FROM DISCOUNT_FROM_EXCHANGE_POINT DFEP
            JOIN DISCOUNT_POINT_APPLY_FOR DPAF ON DFEP.code =
               DPAF.DFEPcode
            WHERE DPAF.FOID = FO.ID
        ), 0) AS ExchangePointDiscount,
```

```
        -- Calculate total discount as the sum of restaurant
           and exchange point discounts
        COALESCE((
            SELECT DFR.value
            FROM DISCOUNT_FROM_RESTAURANT DFR
            JOIN APPLY_FOR A ON DFR.code = A.DFRcode
            WHERE A.FOID = FO.ID
        ), 0) + COALESCE((
            SELECT DFEP.value
            FROM DISCOUNT_FROM_EXCHANGE_POINT DFEP
            JOIN DISCOUNT_POINT_APPLY_FOR DPAF ON DFEP.code =
               DPAF.DFEPcode
            WHERE DPAF.FOID = FO.ID
        ), 0) AS TotalDiscount

    FROM
        FOOD_ORDER FO
    LEFT JOIN CONTAIN C ON C.FOID = FO.ID
    LEFT JOIN FOOD_ITEM FI ON C.FIID = FI.ID
    WHERE
        FO.CID = p_CustomerID
        AND (
            EXISTS (
                SELECT 1
                FROM APPLY_FOR A
                JOIN DISCOUNT_FROM_RESTAURANT DFR ON A.DFRcode
                    = DFR.code
                WHERE A.FOID = FO.ID
            )
            OR EXISTS (
                SELECT 1
                FROM DISCOUNT_POINT_APPLY_FOR DPAF
                JOIN DISCOUNT_FROM_EXCHANGE_POINT DFEP ON DPAF
                   .DFEPcode = DFEP.code
                WHERE DPAF.FOID = FO.ID
            )
        )
    GROUP BY
        FO.ID;
END //


DELIMITER ;
```

- **Testing the query**:
  - Query 1: Testing for Existing Customer ID (Customer ID 1)

```
CALL RetrieveOrdersByCustomerWithDiscounts(1);
```

  - Query 2: Testing for not Existing Customer ID (Customer ID 6)

```
CALL RetrieveOrdersByCustomerWithDiscounts(6);
```

- **Test Result**:
  - Result of Query 1:

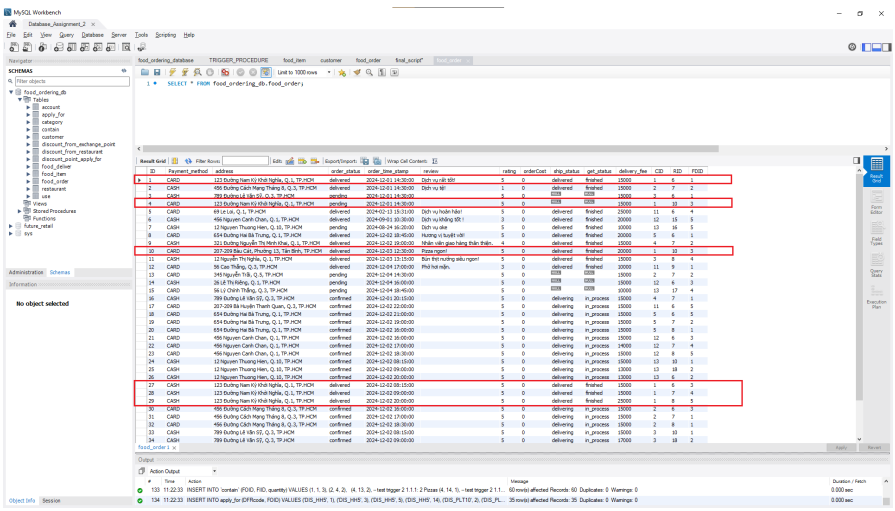


Figure 23: List of Food Orders for Customer ID = 1 in the food_order Table

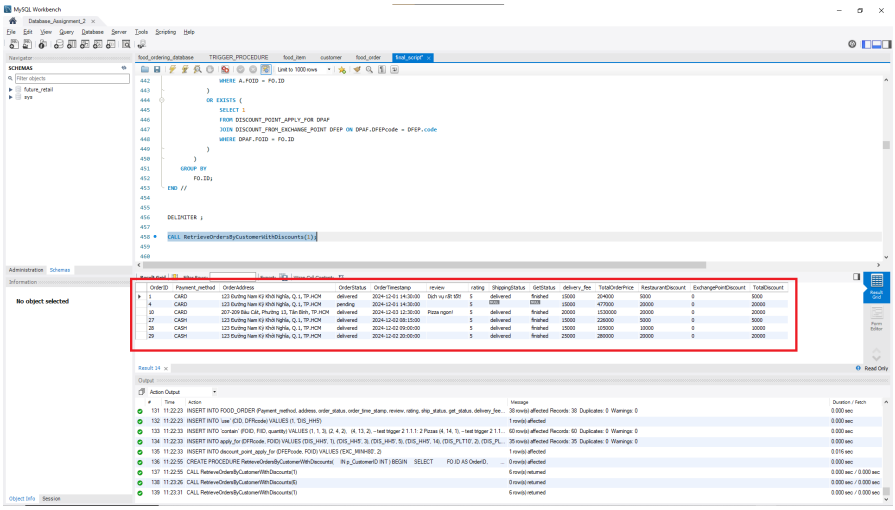


Figure 24: List of Orders Retrieved for Customer ID = 1

– Result of Query 2:

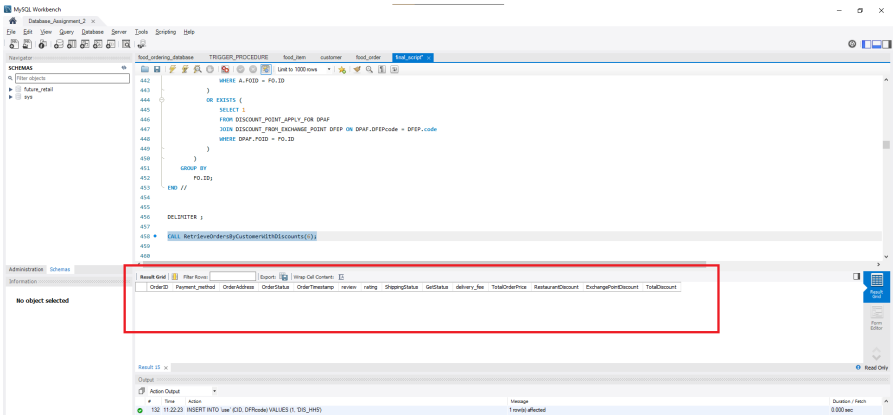


Figure 25: No Orders Retrieved for Customer ID = 6

4.3.2 Procedure 2

- **Description**: The **RetrieveOrdersByCategory** procedure retrieves summary details of all food orders containing items from a specified category. It accepts a category ID as input and returns each order's ID, timestamp, customer name, total items, order status, and total revenue. The procedure filters orders to include only those with items in the given category, calculates total items and revenue per order, and sorts the results by revenue in descending order. This is useful for analyzing category-specific sales performance.
- **Query**:

```
DELIMITER //

CREATE PROCEDURE RetrieveOrdersByCategory(
    IN p_categoryID INT
)
BEGIN
    SELECT
        FO.ID AS OrderID,
        FO.order_time_stamp AS OrderTime,
        FO.rating,
        CU.name AS CustomerName,
        SUM(C.quantity) AS TotalItems,
        FO.order_status AS OrderStatus,
        SUM(C.quantity * FI.price) AS TotalRevenue
    FROM FOOD_ORDER FO
    JOIN CUSTOMER CU ON FO.CID = CU.ID
    JOIN CONTAIN C ON FO.ID = C.FOID
    JOIN FOOD_ITEM FI ON C.FIID = FI.ID
    WHERE FI.categoryID = p_categoryID
    GROUP BY FO.ID, FO.order_time_stamp, CU.name, FO.
       order_status, FO.rating
    HAVING TotalItems > 0
    ORDER BY TotalRevenue DESC;
END //

DELIMITER ;
```

- **Testing the query**:
  - Query 1: Testing for Existing Category ID (Category ID 1)

```
CALL RetrieveOrdersByCategory(1);
```

  - Query 2: Testing for not Existing Category ID (Category ID 28)

```
CALL RetrieveOrdersByCategory(28);
```

- **Result**:

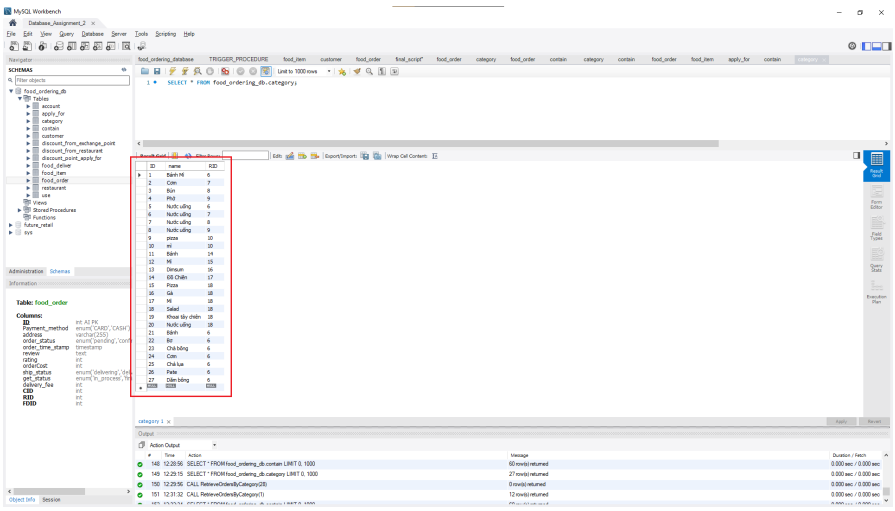


Figure 26: Table of Category before Retrieving

– Result of Query 1:

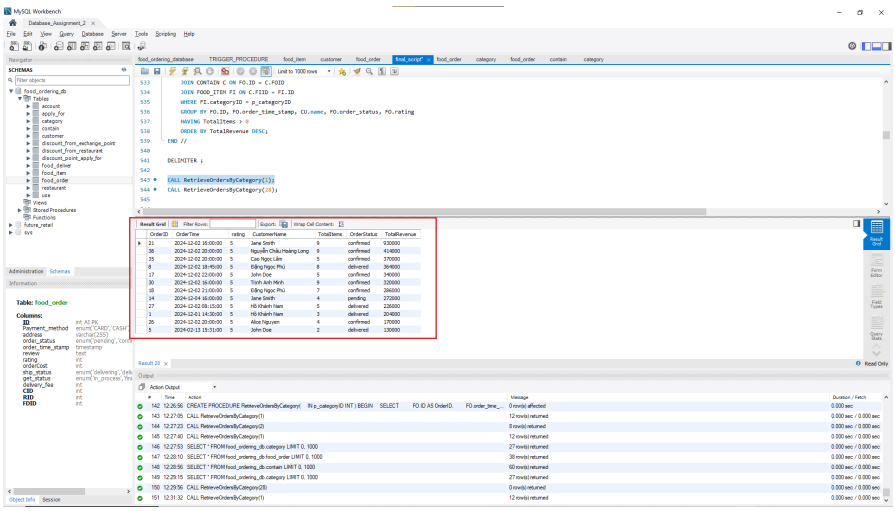


Figure 27: List of Orders Retrieved for Category ID = 1

– Result of Query 2:

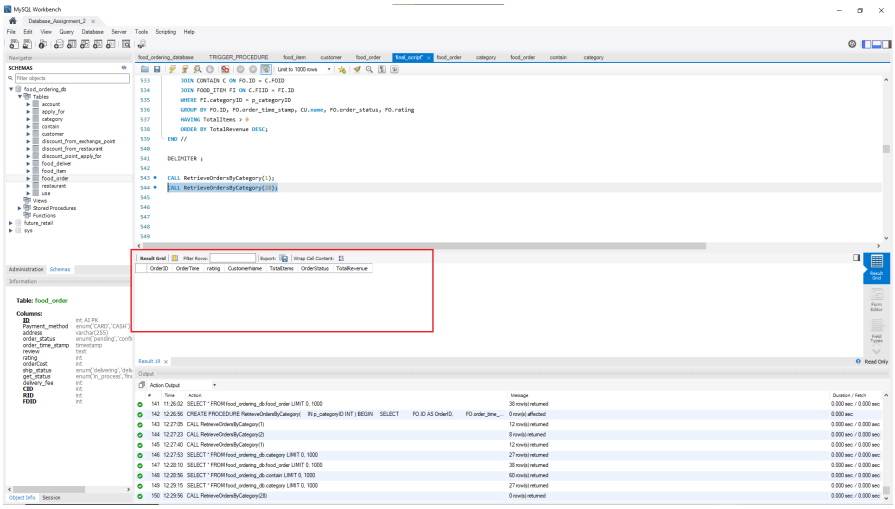


Figure 28: No Orders Retrieved for Category ID = 28

4.4 Writing Functions

Requirements
- Contain IF and/or LOOP statements
- Contain queries, get data from queries to do other operations.
- Have parameters and validate the parameters.

4.4.1 Function 1

- **Description**: The `count_food()` will receive restaurant name and category id and return the number of food items have belong to category id. It receives two parameters:
  1. `restaurant_name`: Represent the restaurant name. It must consist only of alphabetic characters and must not contain any numbers.
  2. `category_id`: Represent the category id. It must be a positive number.
- **Query**:

```
DROP FUNCTION IF EXISTS count_food;
DELIMITER $$
CREATE FUNCTION count_food(
    restaurant_name VARCHAR(255),
    category_id INT
)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE count INT;
    DECLARE restaurant_exists INT;

    IF category_id < 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Category␣
           ID␣must␣be␣positive';
    END IF;

    IF restaurant_name REGEXP '^[\\p{L}␣]+$' THEN
        SELECT COUNT(*) INTO restaurant_exists FROM RESTAURANT
        WHERE name = restaurant_name;

        IF restaurant_exists = 0 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = '
               Restaurant␣does␣not␣exist!';
        ELSE
            SELECT COUNT(*) INTO count FROM FOOD_ITEM fi
            JOIN CATEGORY c ON fi.categoryID = c.ID
            JOIN RESTAURANT r ON c.RID = r.ID
            WHERE categoryID = category_id AND r.name =
               restaurant_name;
        END IF;
    ELSE
```

```
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Wrong␣
            restaurant␣format␣name!';
    END IF;

    RETURN count;
END $$
DELIMITER ;
```

- **Testing queries**:

  In the first testcase, we want to find the number of the restaurant "PizzaHut" with having food belongs to category ID equals 15:

```
SELECT count_food('PizzaHut', 15) AS result;
```

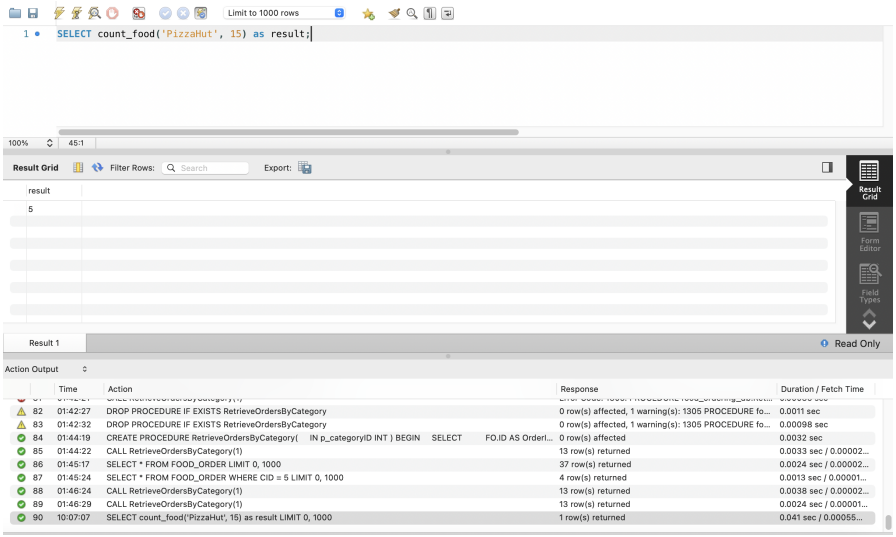


Figure 29: Result 1

In the second ones, we add an unusual case when restaurant name is not valid:

```
SELECT count_food('Pizza Company1', 15) AS result;
```

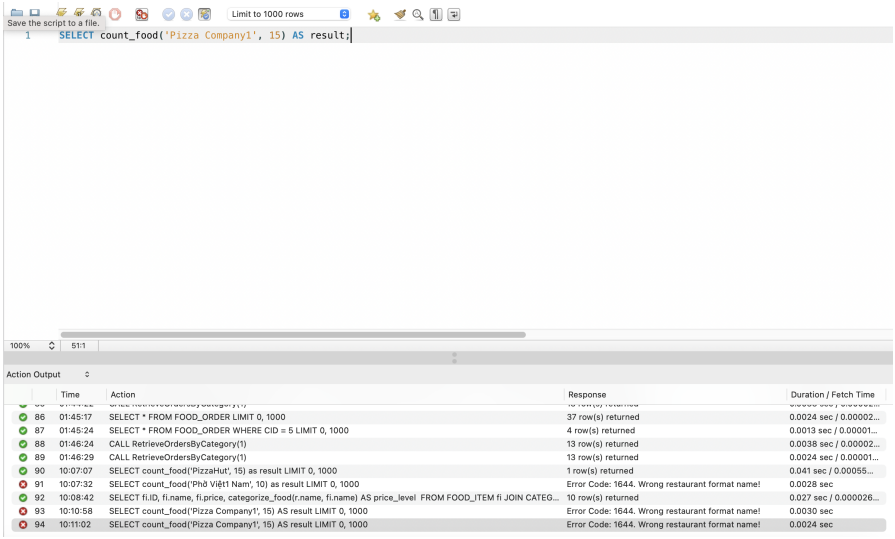


Figure 30: Result 2

As can be observed, MySQL return an error `Error Code: 1644, Wrong restaurant format name!`, which indicates that the validation has succeed.

4.4.2 Function 2

- **Description**: The `categorize _food()` will receive restaurant name and food name in that restaurant and return the price level of that food based on the price of all foods in that restaurant. The price levels will contain three level: "High", "Low", "Medium". It receives two parameters:

  1. `restaurant_name`: Represent the restaurant name. It must consist only of alphabetic characters and must not contain any numbers.
  2. `food_name`: Represent the food name. It must contain one or multiple word and each word may consist of number.

- **Query**:

```
DROP FUNCTION IF EXISTS categorize_food;
DELIMITER $$

CREATE FUNCTION categorize_food(
    restaurant_name VARCHAR(255),
    food_name VARCHAR(255)
)
```

```
RETURNS VARCHAR(15)
DETERMINISTIC
BEGIN
    DECLARE price_level VARCHAR(15);
    DECLARE food_item_price INT;
    DECLARE average_price INT;
    DECLARE min_price INT;
    DECLARE max_price INT;


    IF restaurant_name NOT REGEXP '^[\\p{L}␣]+$' THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Wrong␣
           restaurant␣format␣name!';
    END IF;

    IF food_name NOT REGEXP '^[\\p{L}0-9␣\-\.,\(\)]+$' THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = '
                   Invalid␣food␣name';
        END IF;

    IF (SELECT COUNT(*)
        FROM FOOD_ITEM fi
        JOIN CATEGORY c ON fi.categoryID = c.ID
        JOIN RESTAURANT r ON c.RID = r.ID
        WHERE fi.name = food_name AND r.name = restaurant_name
           ) = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Food␣item␣
           does␣not␣exist␣in␣the␣restaurant!';
    END IF;

    SELECT price INTO food_item_price
    FROM FOOD_ITEM fi
    JOIN CATEGORY c ON fi.categoryID = c.ID
    JOIN RESTAURANT r ON c.RID = r.ID
    WHERE fi.name = food_name AND r.name = restaurant_name;

    SELECT
        AVG(price), MIN(price), MAX(price)
    INTO average_price, min_price, max_price
    FROM FOOD_ITEM fi
    JOIN CATEGORY c ON fi.categoryID = c.ID
    JOIN RESTAURANT r ON c.RID = r.ID
    WHERE r.name = restaurant_name;

    IF food_item_price >= (average_price + max_price) / 2 THEN
        SET price_level = 'High';
    ELSEIF food_item_price <= (average_price + min_price) / 2
       THEN
        SET price_level = 'Low';
```

```
    ELSE
        SET price_level = 'Medium';
    END IF;

    RETURN price_level;
END $$

DELIMITER ;
```

- **Testing query**:

```
SELECT fi.ID, fi.name, fi.price, categorize_food(r.name, fi.
    name) AS price_level
FROM FOOD_ITEM fi
JOIN CATEGORY c ON c.ID = fi.categoryID
JOIN RESTAURANT r ON r.ID = c.RID
WHERE r.name = 'PizzaHut';
```

- **Result**:

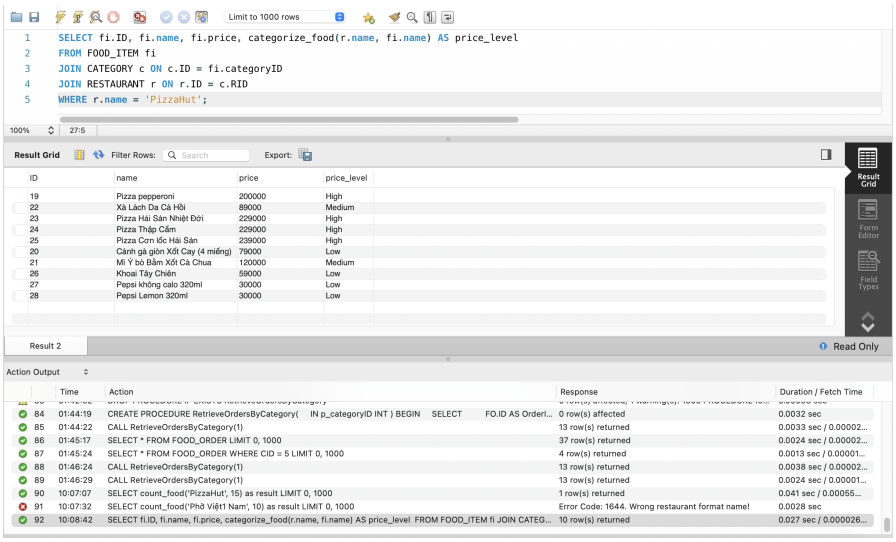


Figure 31: Result

5 Applications Results

For the application implementation, we use **NodeJS**, a runtime environment built for writing JavaScript code on the server-side, and **ExpressJS**, a minimal framework for building web applications on top of NodeJS. For interacting with MySQL, we use `mysql2`, an efficient NodeJS driver that allows us to embed our SQL scripts from above tasks directly into the JavaScript code without the need for ORMs. The source code for our application can be found at the following link: Github link

After running the application, an introduction page will be rendered, showing the basic information about the food ordering system:

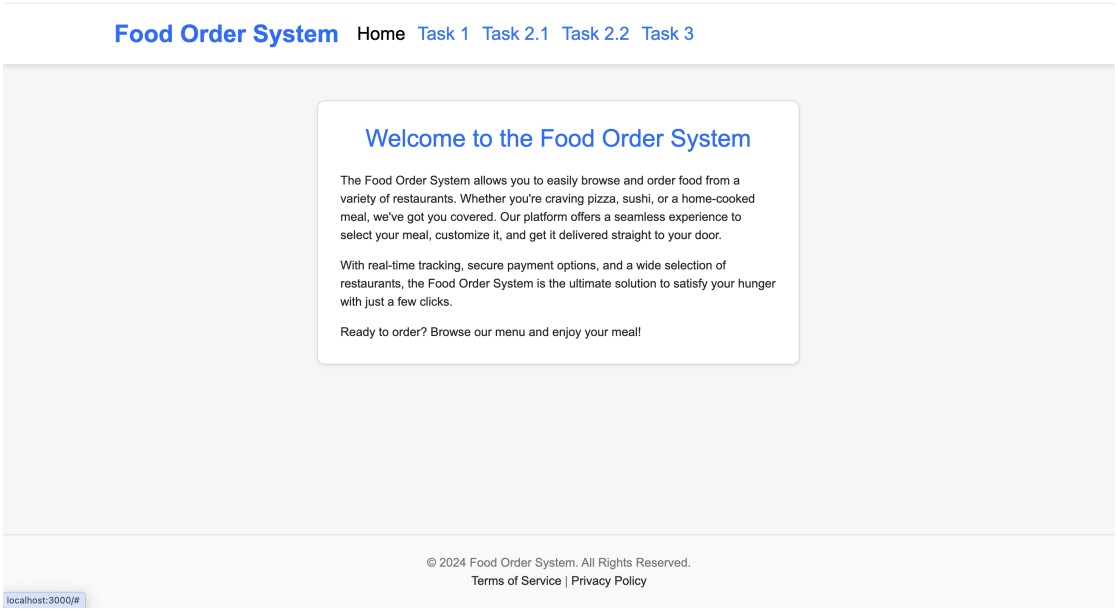


Figure 32: Introduction page

Due to the convenience for instructor to grade the application, we make other 4 tabs in the navigation, represent 3 tasks (task 2 will be splitted into 2.1 and 2.2) for the application implementation in the assignment.

5.1 Task 1

Requirements
- Create an interface to demonstrate inserting, deleting, and updating data.

For the task one, we make an interface for inserting, deleting and updating on food items. We utilize the procedures from task 1.2.1 to implement this page. The list of food items will be rendered when first encounter the page:

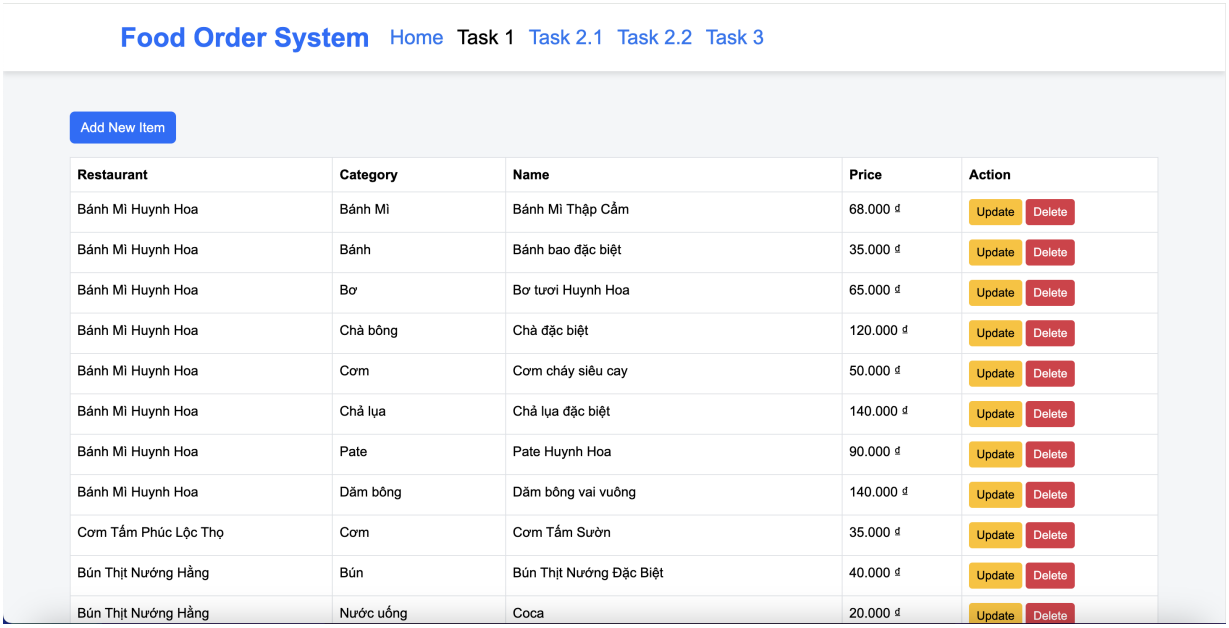


Figure 33: Food Items page

In order to add a new food items, clicking the blue button "Add New Items". An Modal will be opened for the Admin to insert a new food item into the database:

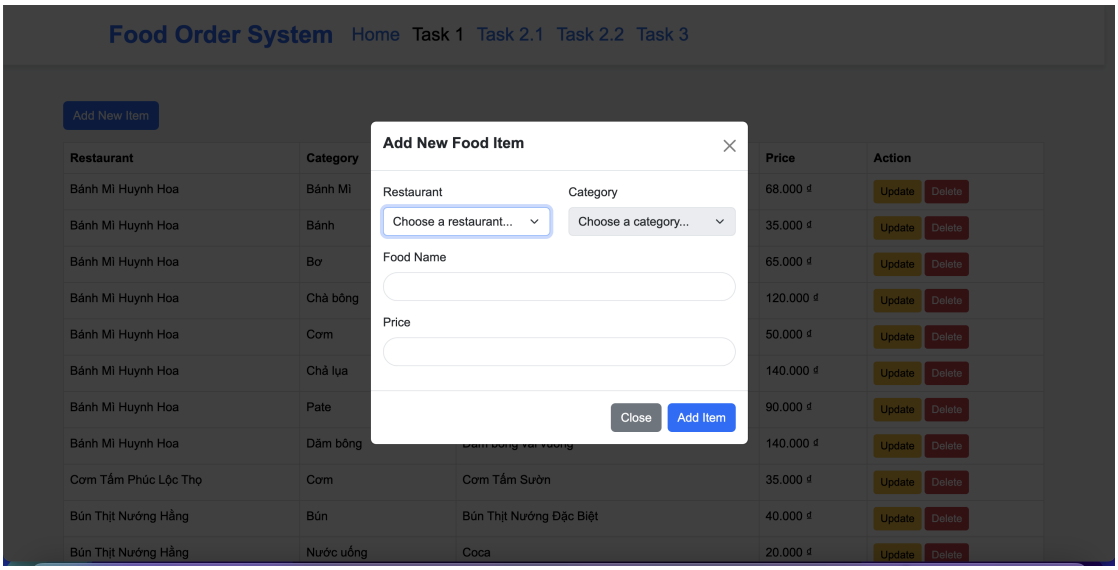


Figure 34: Adding form

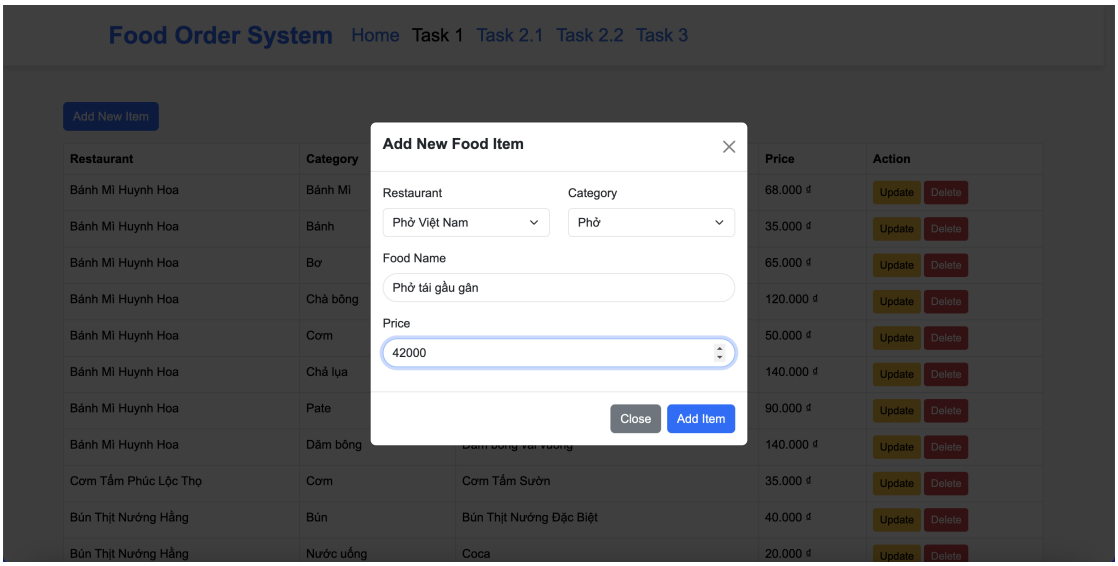
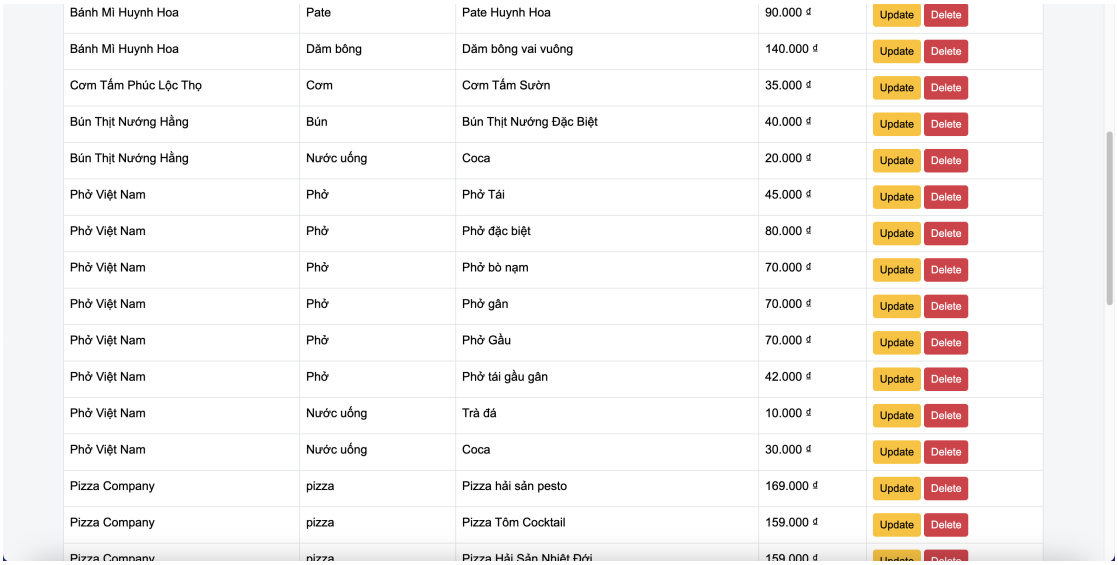


Figure 35: Add a new food item

After submit the form to adding the above food items, the information of that food items has been rendered in the food item list:



Bánh Mì Huynh Hoa	Pate	Pate Huynh Hoa	90.000 ₫	Update Delete
Bánh Mì Huynh Hoa	Dăm bông	Dăm bông vai vuông	140.000 ₫	Update Delete
Cơm Tấm Phúc Lộc Thọ	Cơm	Cơm Tấm Sườn	35.000 ₫	Update Delete
Bún Thịt Nướng Hằng	Bún	Bún Thịt Nướng Đặc Biệt	40.000 ₫	Update Delete
Bún Thịt Nướng Hằng	Nước uống	Coca	20.000 ₫	Update Delete
Phở Việt Nam	Phở	Phở Tái	45.000 ₫	Update Delete
Phở Việt Nam	Phở	Phở đặc biệt	80.000 ₫	Update Delete
Phở Việt Nam	Phở	Phở bò nạm	70.000 ₫	Update Delete
Phở Việt Nam	Phở	Phở gân	70.000 ₫	Update Delete
Phở Việt Nam	Phở	Phở Gầu	70.000 ₫	Update Delete
Phở Việt Nam	Phở	Phở tái gầu gân	42.000 ₫	Update Delete
Phở Việt Nam	Nước uống	Trà đá	10.000 ₫	Update Delete
Phở Việt Nam	Nước uống	Coca	30.000 ₫	Update Delete
Pizza Company	pizza	Pizza hải sản pesto	169.000 ₫	Update Delete
Pizza Company	pizza	Pizza Tôm Cocktail	159.000 ₫	Update Delete
Pizza Company	pizza	Pizza Hải Sản Nhiệt Đới	159.000 ₫	Update Delete

Figure 36: Updated lists

The "Update" button in each food item row allow the user to update the information of that food items:

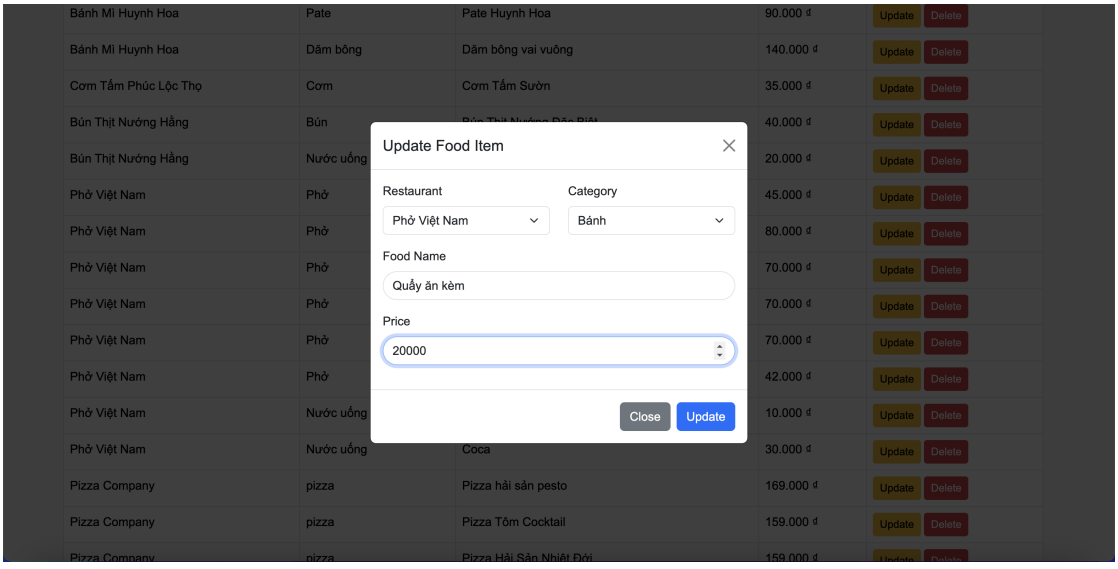


Figure 37: Update form

On the other hand, the "Delete" button allows the user to delete a specific food order:

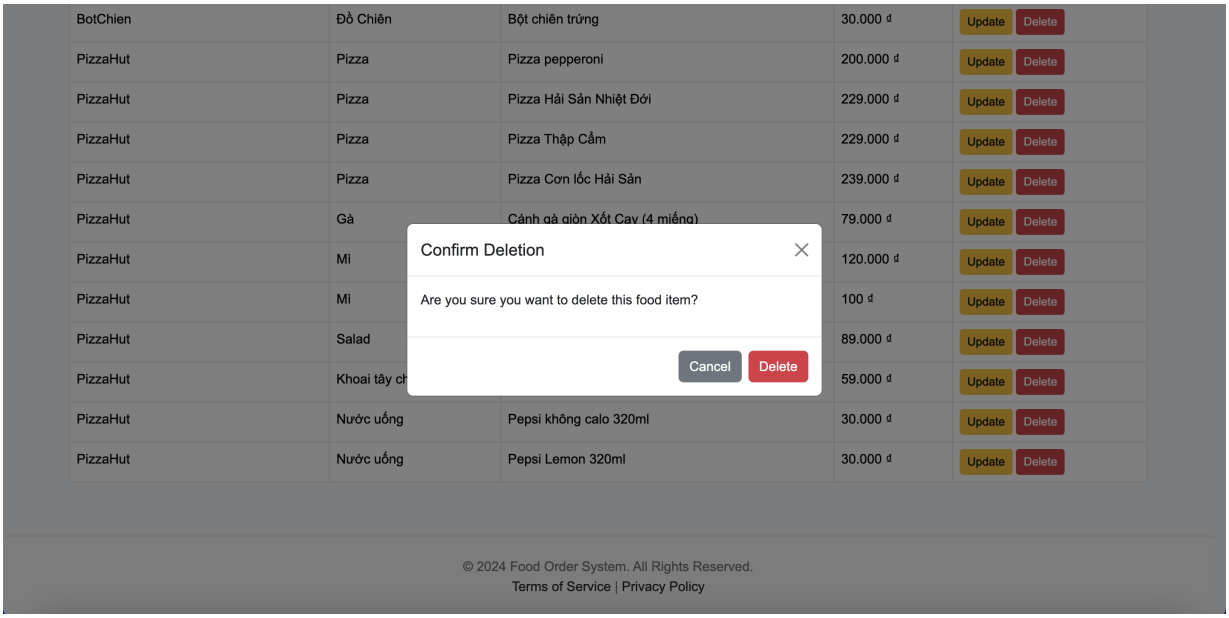


Figure 38: Delete configrm modal

5.2 Task 2

Requirements

- Create an interface to display the outputs of procedures in requirement 1.2.3.
- The interface should allow updating and deleting data from the list, and include some additional features such as search, sorting, data validation, error handling when updating or deleting data, reporting meaningful error messages, ...

For the requirements, we split the page for rendering task 2 UI into 2 pages for 2 procedures in requirement 1.2.3. In the first page, the UI will be splitted into 2 parts, with the first part for the user the enter the customer ID and the second one will render all the orders of that customer that have the discounts, as well as some button serving the filter and sort tasks. The "reset" button in the top right of the table will reset all the filter and sort criteria into initial state, meaning that the table will be rendered by the default result of MySQL script. There would be a delete icon in each row, indicating that the user can delete that order:

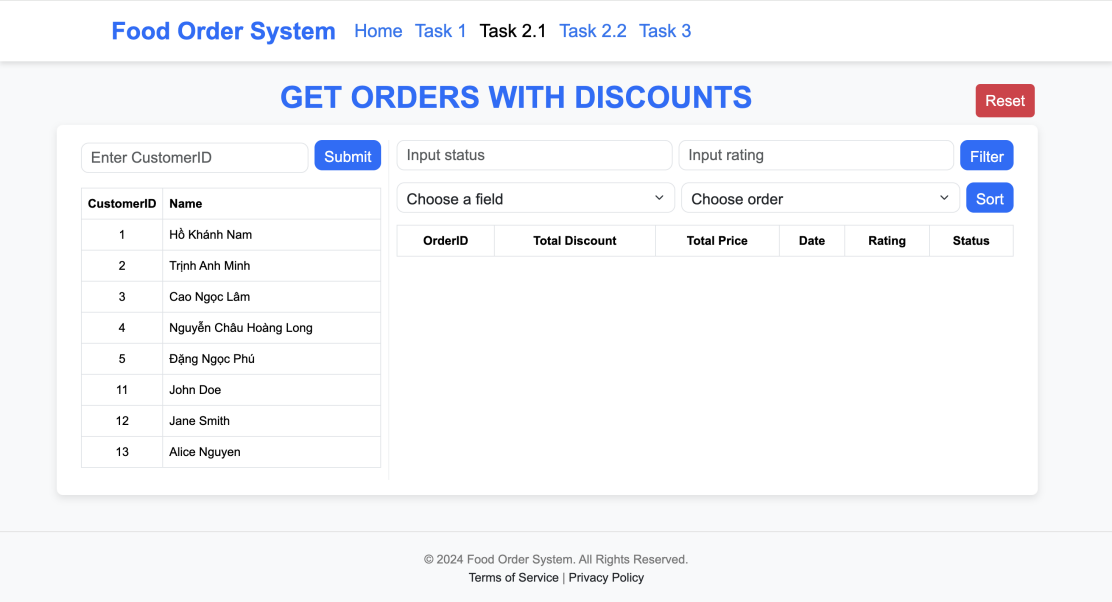


Figure 39: Procedure 1

Typing the customer ID = 1 into the input:

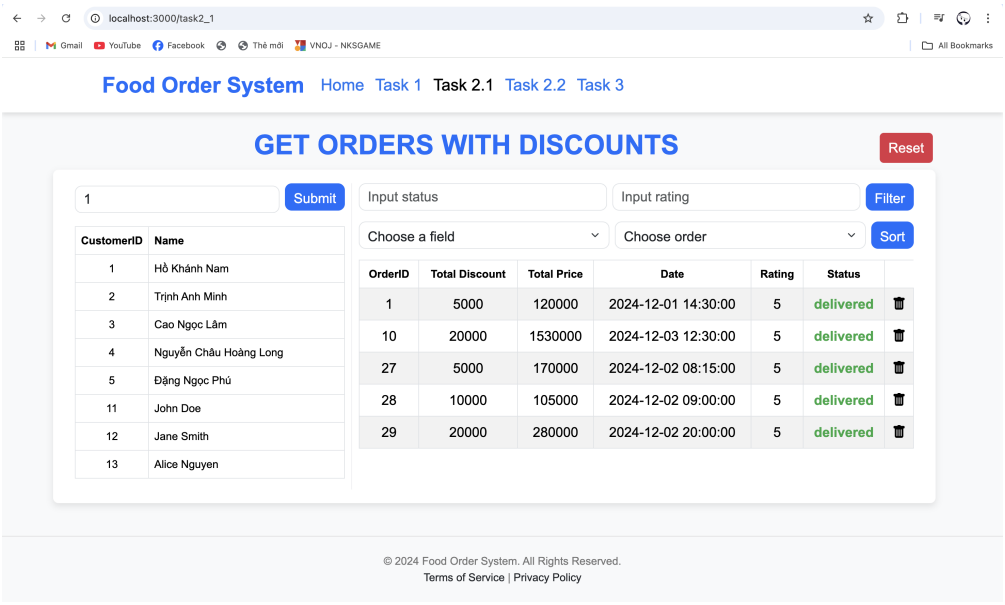


Figure 40: Procedure 1 (input = 1)

Typing the customer ID = 2 into the input:

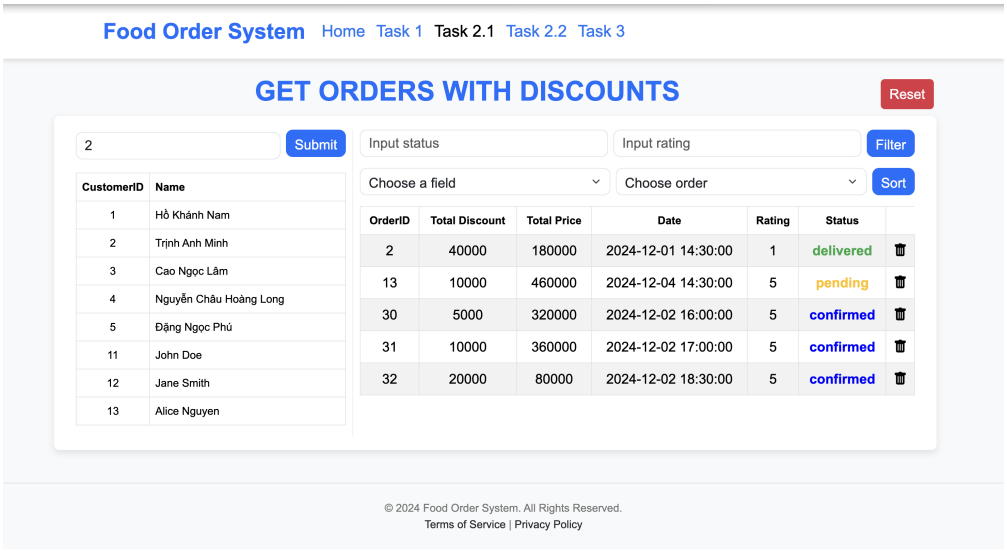


Figure 41: Procedure 1 (input = 2)

Filter the above result with rating = 5 and status = 'confirmed'

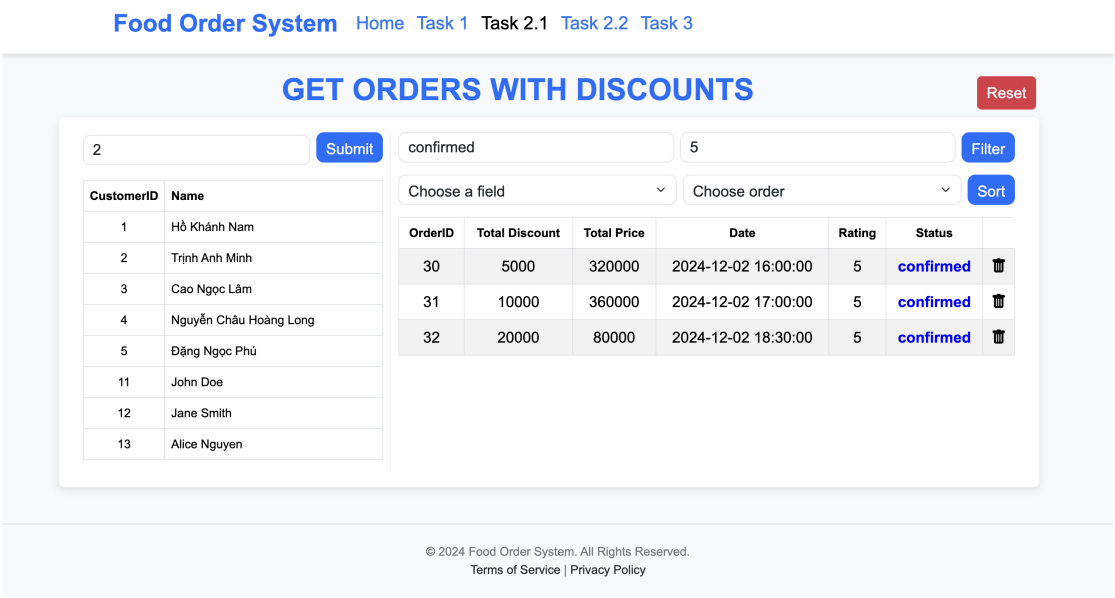


Figure 42: Procedure 1 (input = 2) (filter)

Sort the above result by the descending order of total discount:

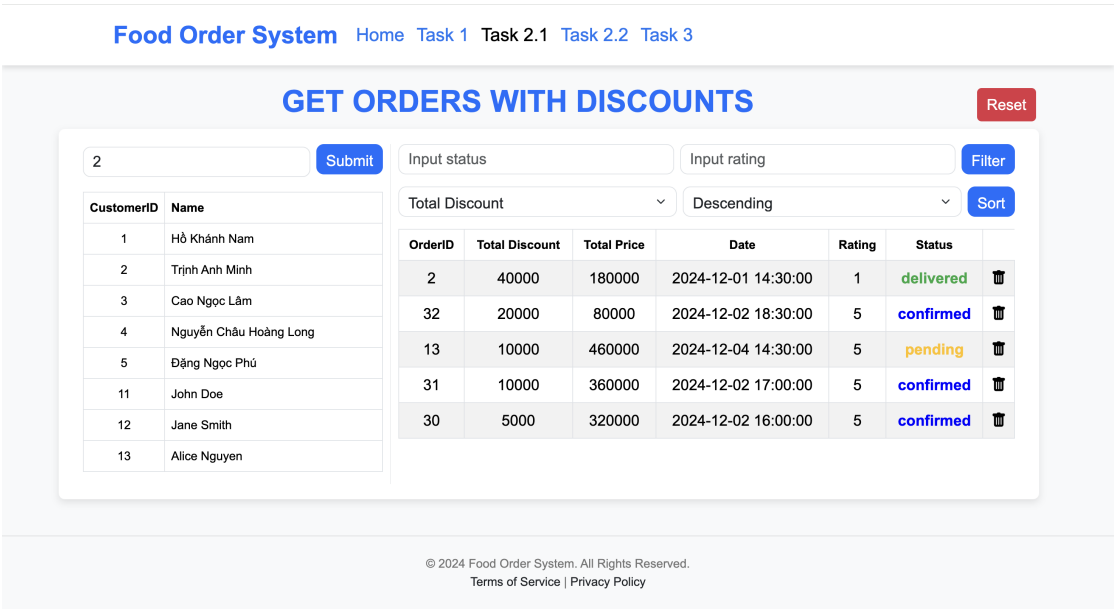


Figure 43: Procedure 1 (input = 2) (sort)

If the button 'delete' of a row is clicked, an modal will be opened to reconfirm again that action:

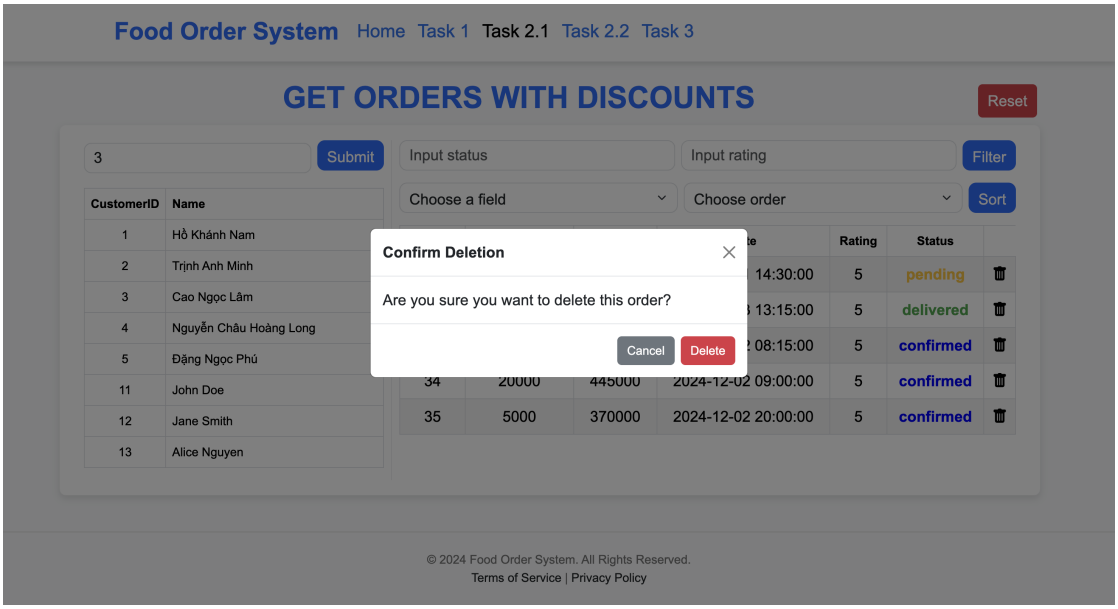


Figure 44: Procedure 1 (delete)

For the remaining procedure page, the functionalities are the same:

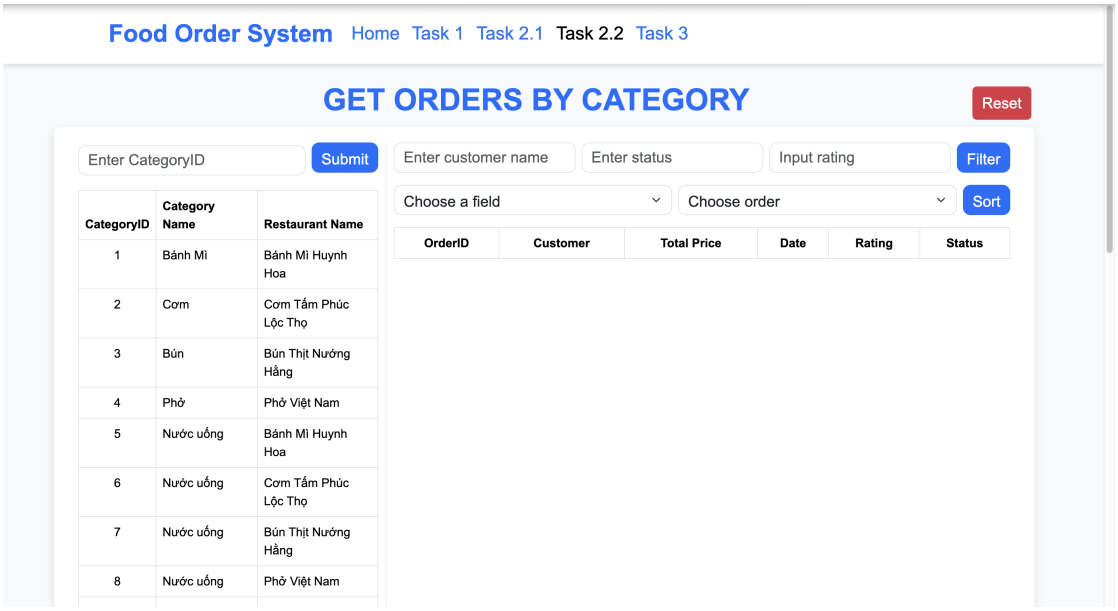


Figure 45: Procedure 2 (First rendering)

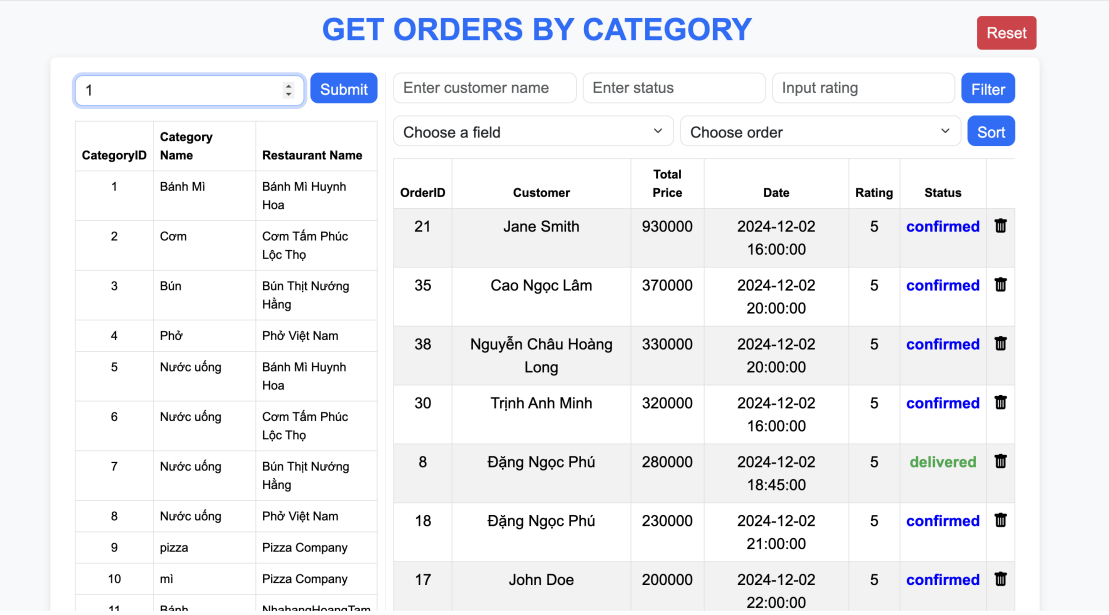


Figure 46: Procedure 2 (input = 1)

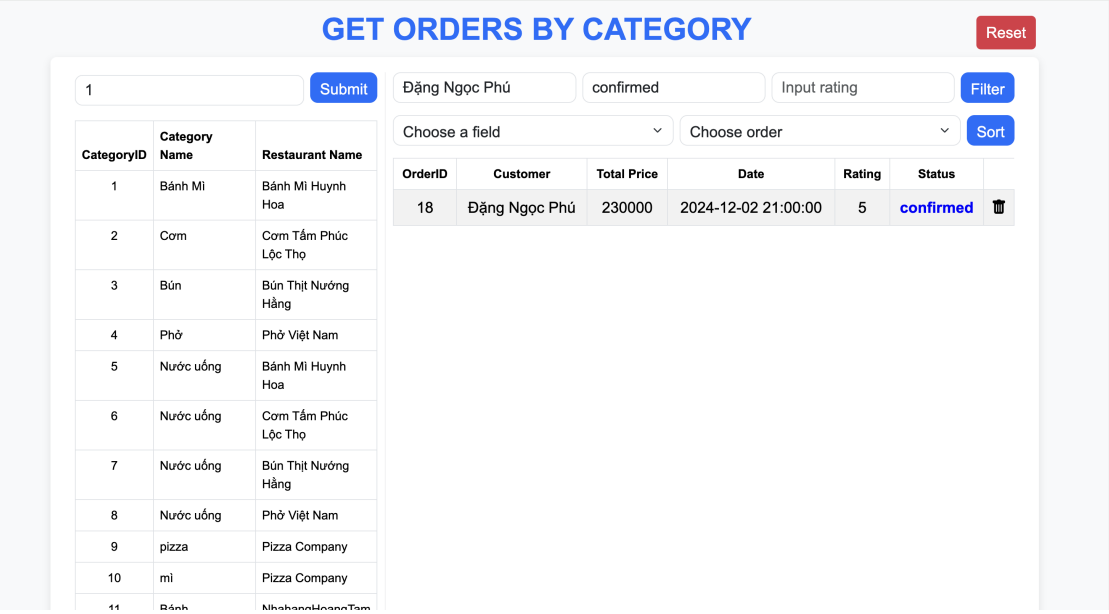


Figure 47: Procedure 2 (input = 1, customer name = 'Đặng Ngọc Phú')

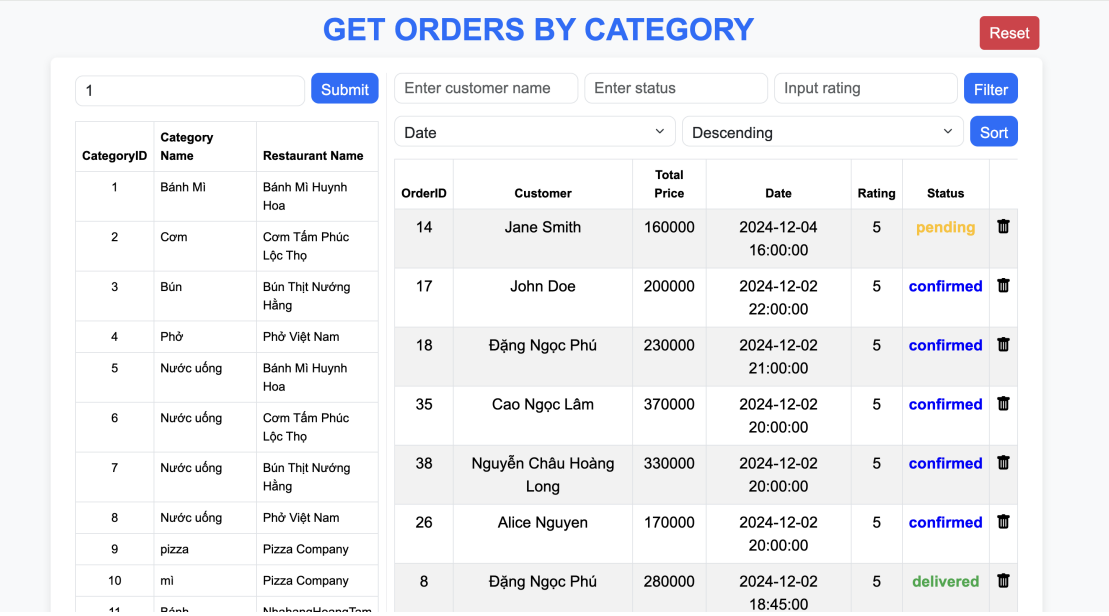


Figure 48: Procedure 2 (input = 1, descending sort by Date)

5.3 Task 3

Requirements

- Create an interface to demonstrate at least one procedure from requirement 1.2.3 or one function from requirement 1.2.4.

In this page, we implement two boxes, each box contain the input textbox as in the functions in 1.3.4. The first box will contain the UI for the input of the Function 1:

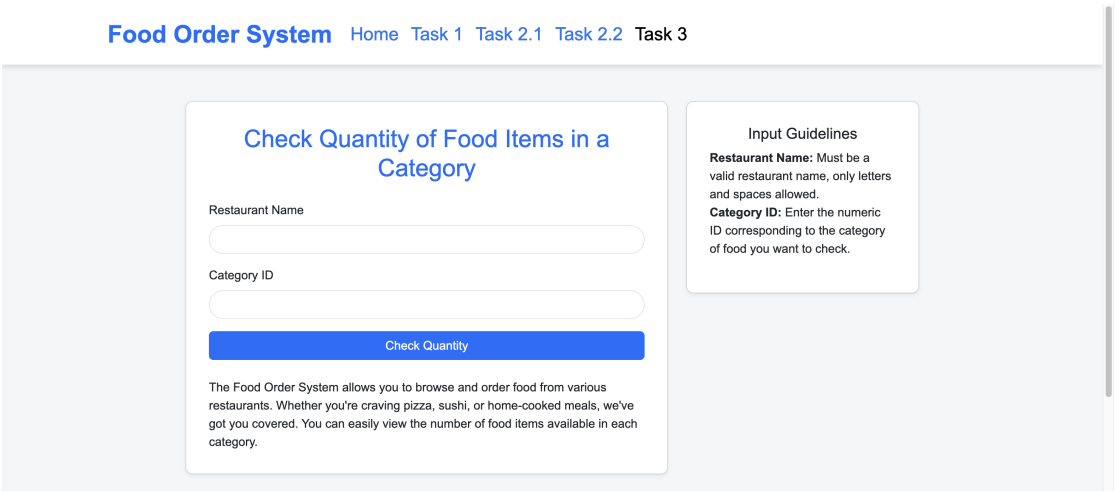


Figure 49: Function 1

Typing the input from the testcases in the above Function section:

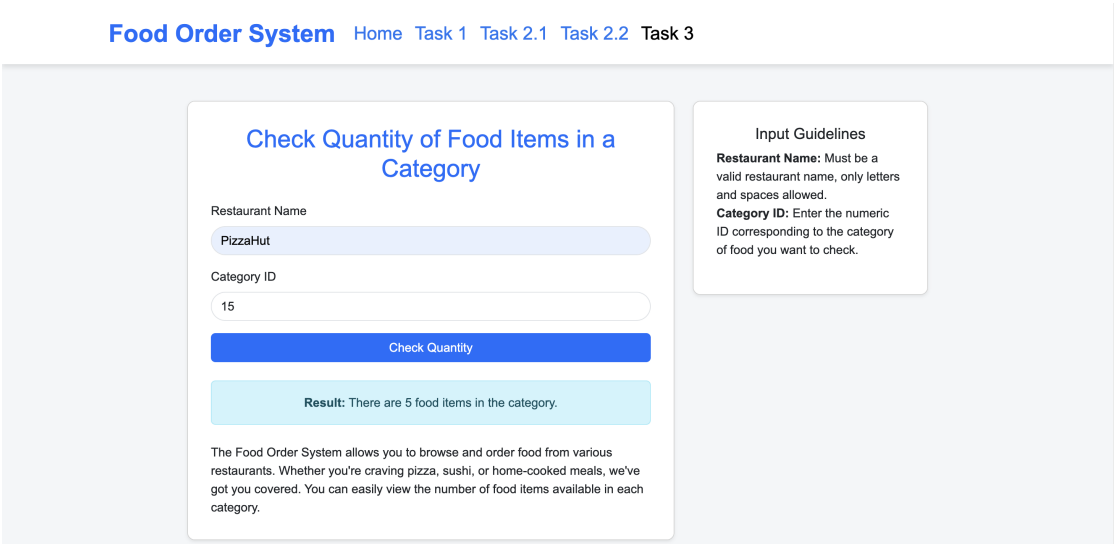


Figure 50: Testcase 1

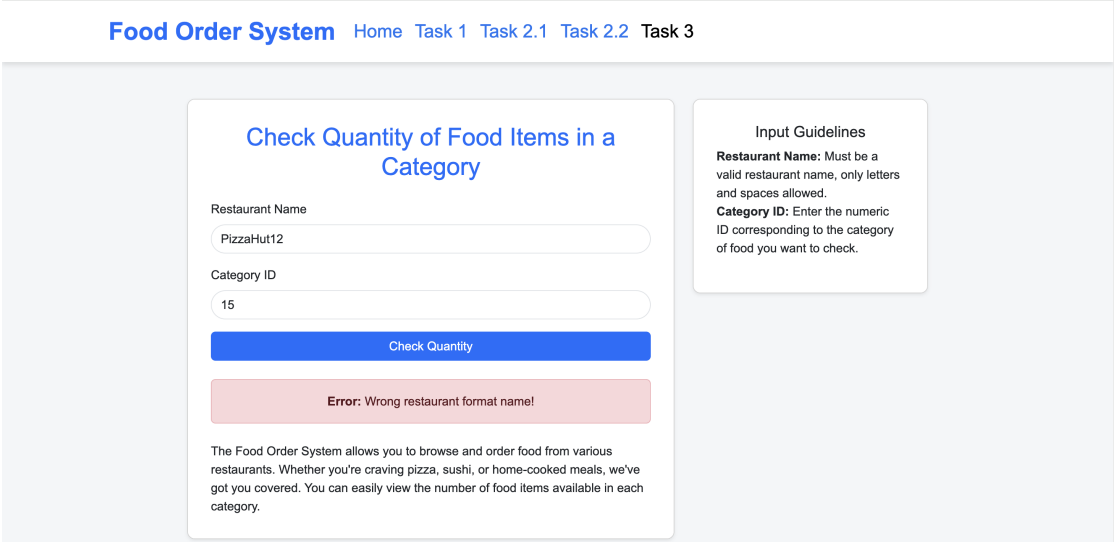


Figure 51: Testcase 2

As can be observed, the result box return expected result as when we run in MySQL Workbench. If the application received throw error from MySQL ("wrong restaurant format name" in the above case), the page will render the same messages error as MySQL.

For the function 2 of the chatbox, we combine the function with a select statement so that user can type the restaurant name and it will render all the food item name, with price and price level (getting from the function 2).

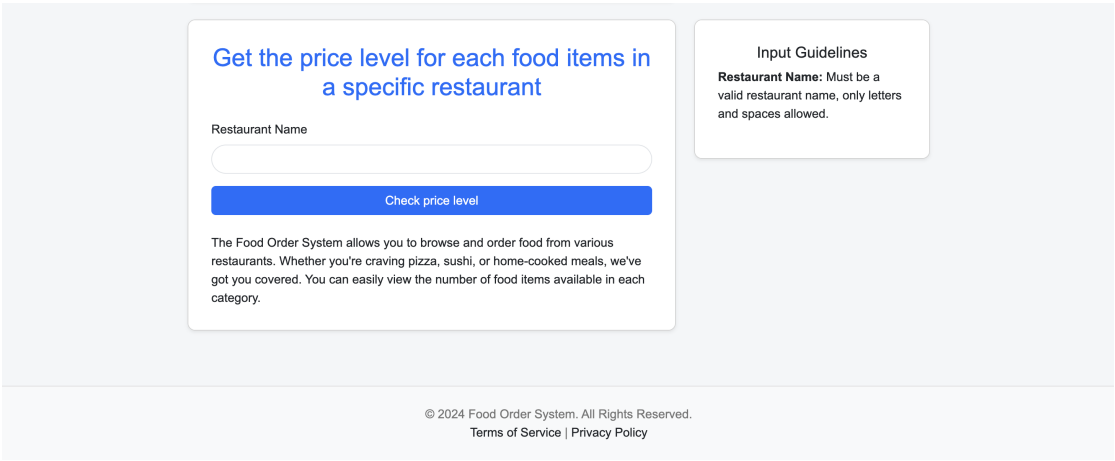


Figure 52: Function 2 Input

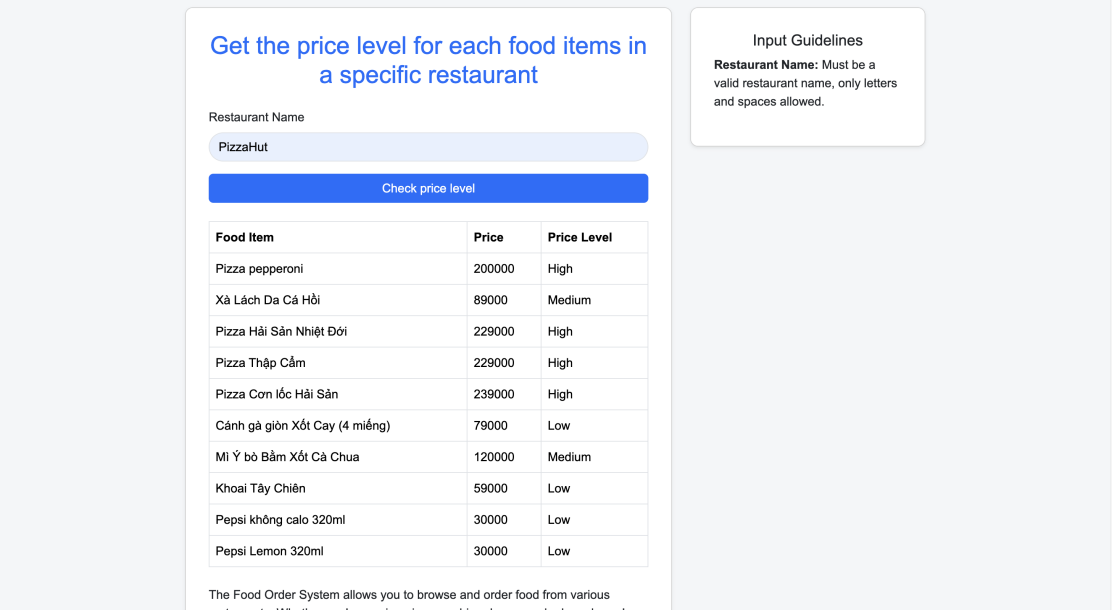


Figure 53: Function 2 Form Result

6 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.

7 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.