BURGER SHOP MANAGEMENT SYSTEM

Submitted By

1. [redacted]
2. Tasnim, Quazi Fariha [redacted]
3. [redacted]
4. [redacted]

SUBJECT: Advance Database Management System

SECTION: B

Submitted To

[redacted]

Department of Computer Science

American International University-Bangladesh

Contents

Burger Shop Management System

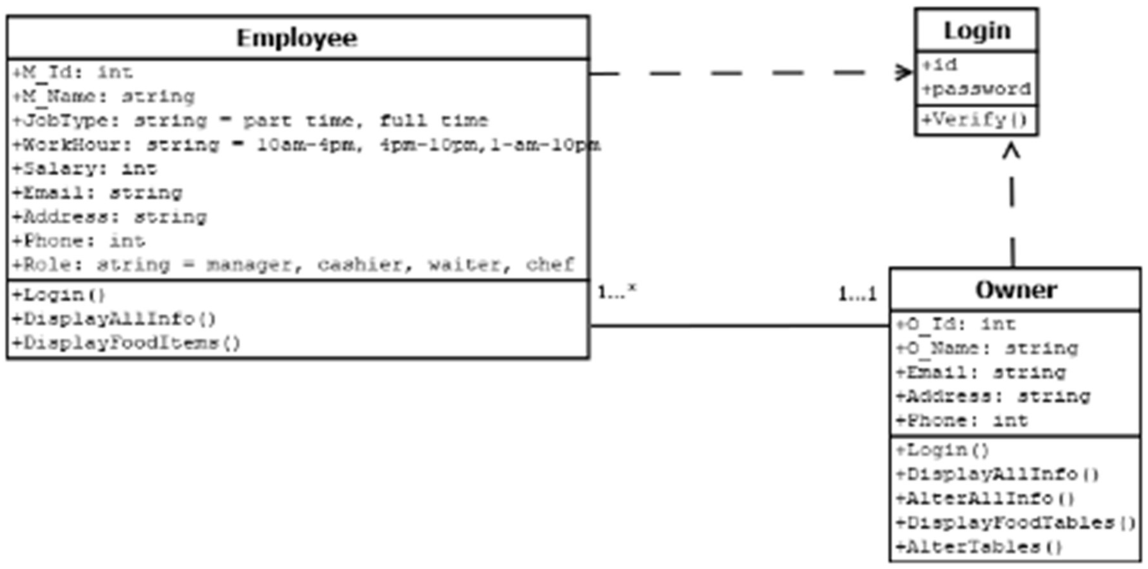
Introduction

Burger shop has become a very popular business in Bangladesh nowadays since fast food has spread its popularity among young adolescents in the developing countries. In order to have a successful Burger shop business there is no alternative of having a Burger Shop Management System. This burger shop management system keeps record of the employees, food items and bills. We are hoping that proper management of data through this system can help to run the Burger Shop more efficiently.

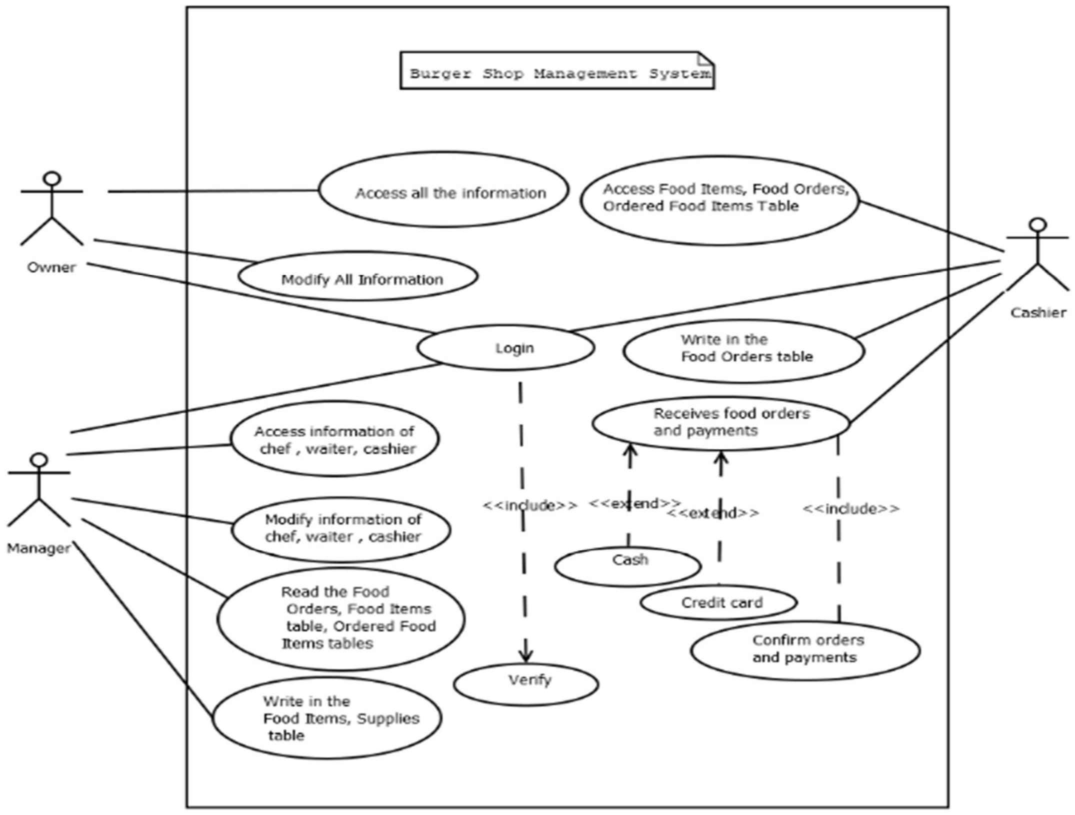
Project Proposal

Due to the popularity of burger among young adolescents burger shop is quite a popular business choice. In order to attain prosperity in burger shop business and to run it efficiently, the data in the regarding the burger shop need to be managed properly. We are proposing a management system for the burger shop. Through this burger shop management system, the owner can store details of burgers and other food items and manage the employees record and also provide the facility to show the details of total purchase and total sales. In this way, this management system can help to improve Burger Shop business and also save time to process food orders and bills.

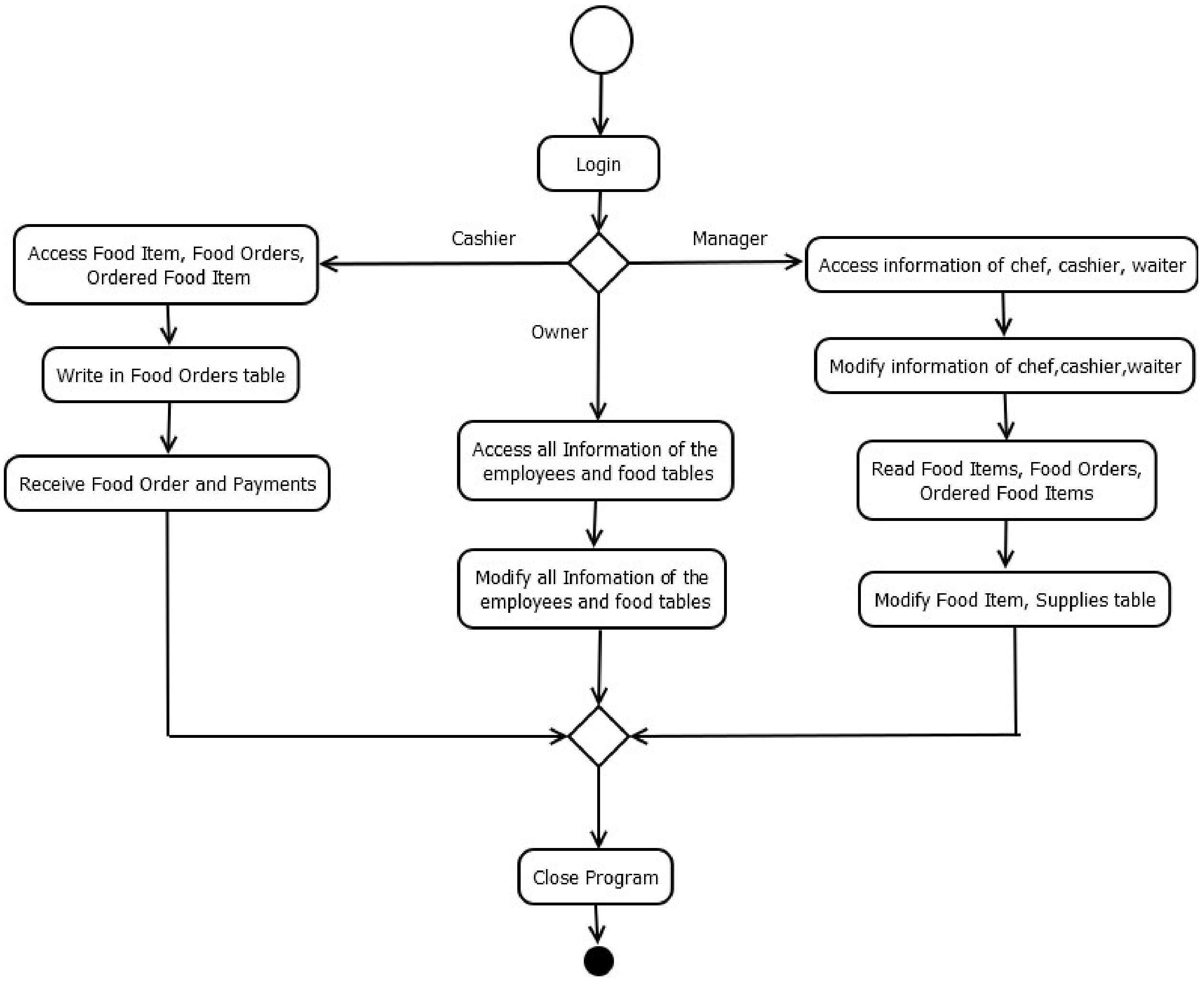
Class Diagram



Use Case Diagram

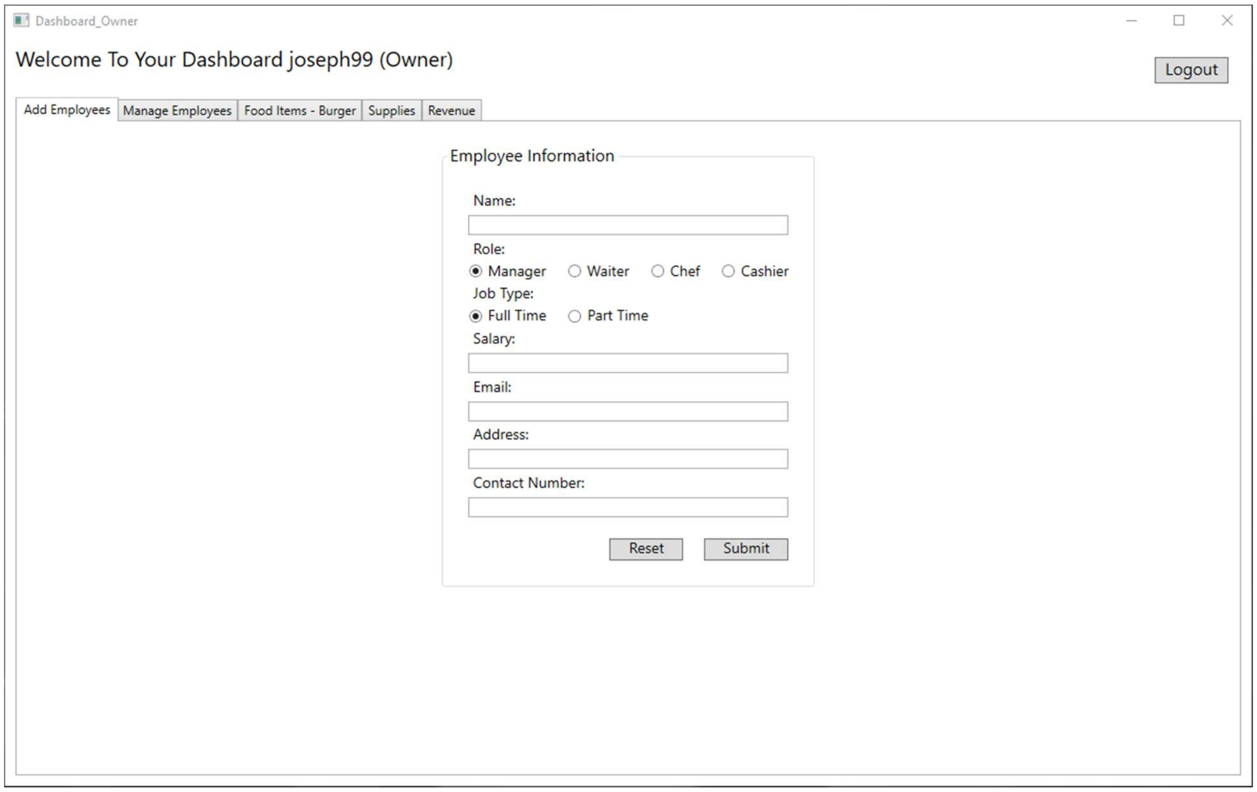

Burger Shop Management System
Owner
Access all the information
Access Food Items, Food Orders, Ordered Food Items Table
Cashier
Modify All Information
Login
Write in the Food Orders table
Receives food orders and payments
Manager
Access information of chef , waiter, cashier
Modify information of chef, waiter , cashier
Read the Food Orders, Food Items table, Ordered Food Items tables
Write in the Food Items, Supplies table
<<include>>
<<extend>>
<<extend>>
<<include>>
Cash
Credit card
Confirm orders and payments
Verify

Activity Diagram


Login
Cashier
Manager
Owner
Access Food Item, Food Orders, Ordered Food Item
Write in Food Orders table
Receive Food Order and Payments
Access all Information of the employees and food tables
Modify all Infomation of the employees and food tables
Access information of chef, cashier, waiter
Modify information of chef,cashier,waiter
Read Food Items, Food Orders, Ordered Food Items
Modify Food Item, Supplies table
Close Program

User Interface

The application for this burger shop management system has been developed using C#. The interfaces of the application are designed using a specific framework of C# which is called Windows Presentation Foundation (WPF). We have completed developing this application. Some screenshots of different windows of the application are shown below:


Dashboard_Owner

Welcome To Your Dashboard joseph99 (Owner)

Logout

Add Employees | Manage Employees | Food Items - Burger | Supplies | Revenue

Employee Information

Name:

Role: [✓ Manager] Waiter Chef Cashier

Job Type: [✓ Full Time] Part Time

Salary:

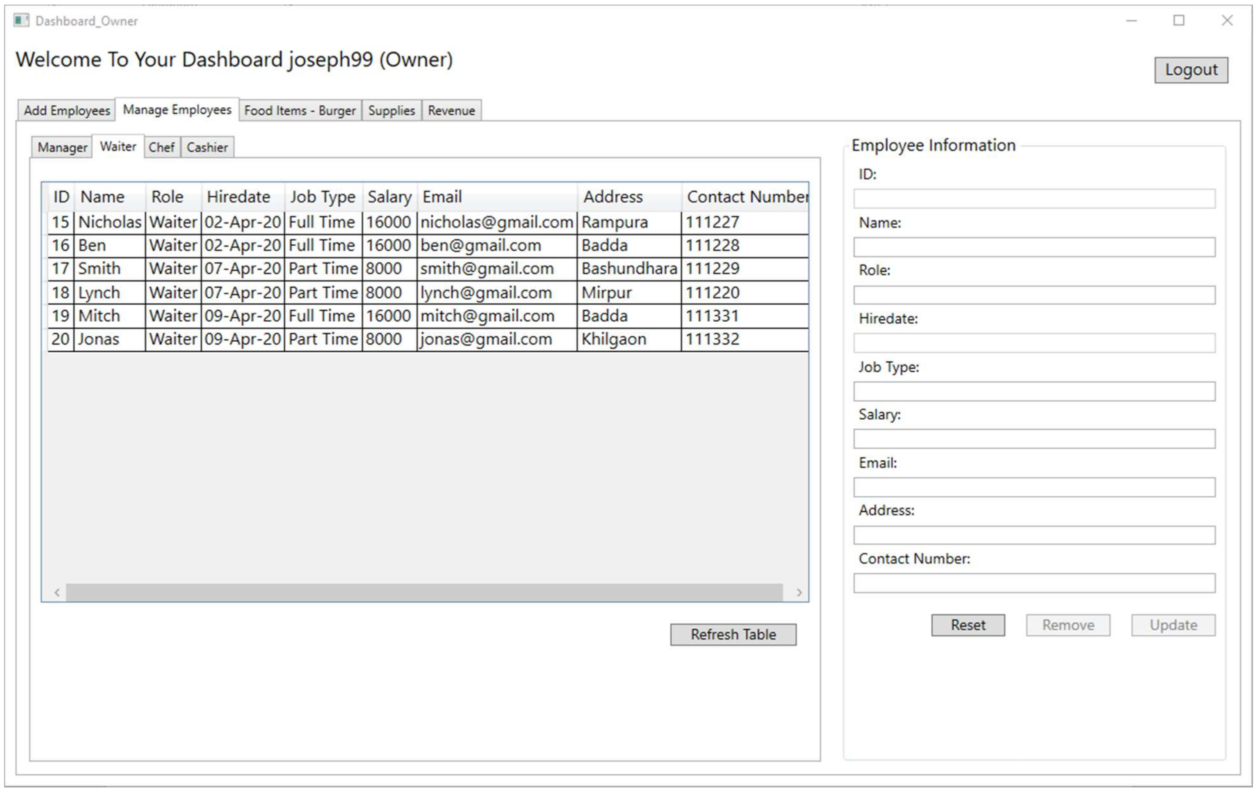
Email:

Address:

Contact Number:

Reset Submit

Description: The owner can add new employees in this window


Dashboard_Owner

Welcome To Your Dashboard joseph99 (Owner)

Logout

Add Employees | Manage Employees | Food Items - Burger | Supplies | Revenue

Manager | Waiter | Chef | Cashier

ID	Name	Role	Hiredate	Job Type	Salary	Email	Address	Contact Number
15	Nicholas	Waiter	02-Apr-20	Full Time	16000	nicholas@gmail.com	Rampura	111227
16	Ben	Waiter	02-Apr-20	Full Time	16000	ben@gmail.com	Badda	111228
17	Smith	Waiter	07-Apr-20	Part Time	8000	smith@gmail.com	Bashundhara	111229
18	Lynch	Waiter	07-Apr-20	Part Time	8000	lynch@gmail.com	Mirpur	111220
19	Mitch	Waiter	09-Apr-20	Full Time	16000	mitch@gmail.com	Badda	111331
20	Jonas	Waiter	09-Apr-20	Part Time	8000	jonas@gmail.com	Khilgaon	111332

Refresh Table

Employee Information

ID:

Name:

Role:

Hiredate:

Job Type:

Salary:

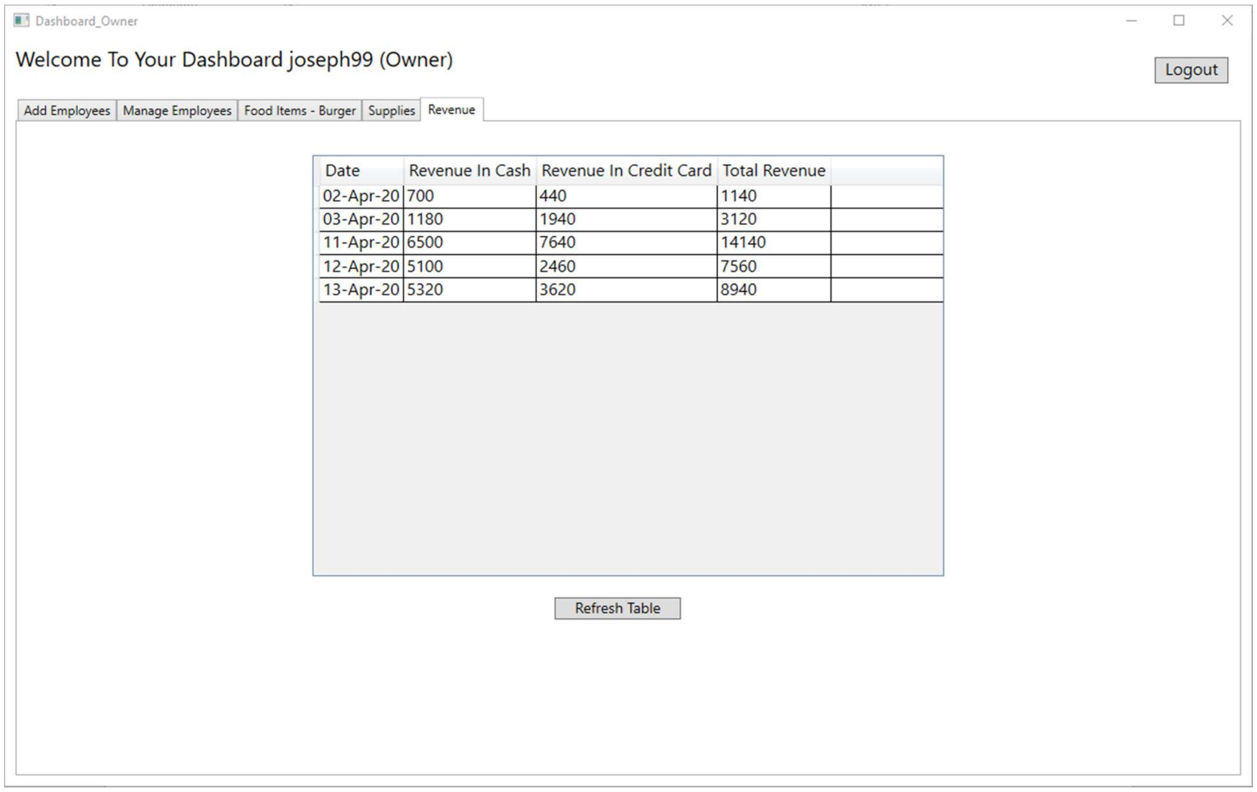
Email:

Address:

Contact Number:

Reset Remove Update

Description: The owner can modify the information of all types of employees in this window

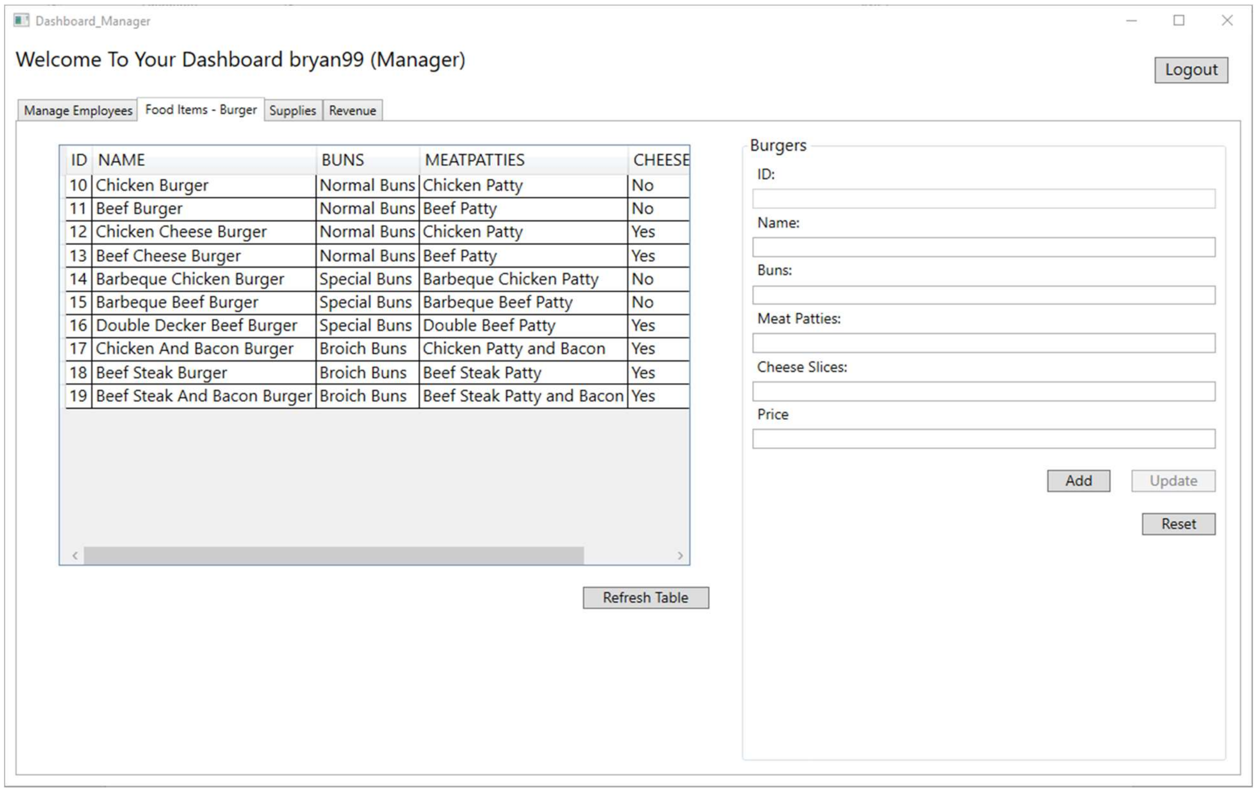


Dashboard_Owner

Welcome To Your Dashboard joseph99 (Owner)

Date	Revenue In Cash	Revenue In Credit Card	Total Revenue
02-Apr-20	700	440	1140
03-Apr-20	1180	1940	3120
11-Apr-20	6500	7640	14140
12-Apr-20	5100	2460	7560
13-Apr-20	5320	3620	8940

Description: The owner can check the total revenue for each date

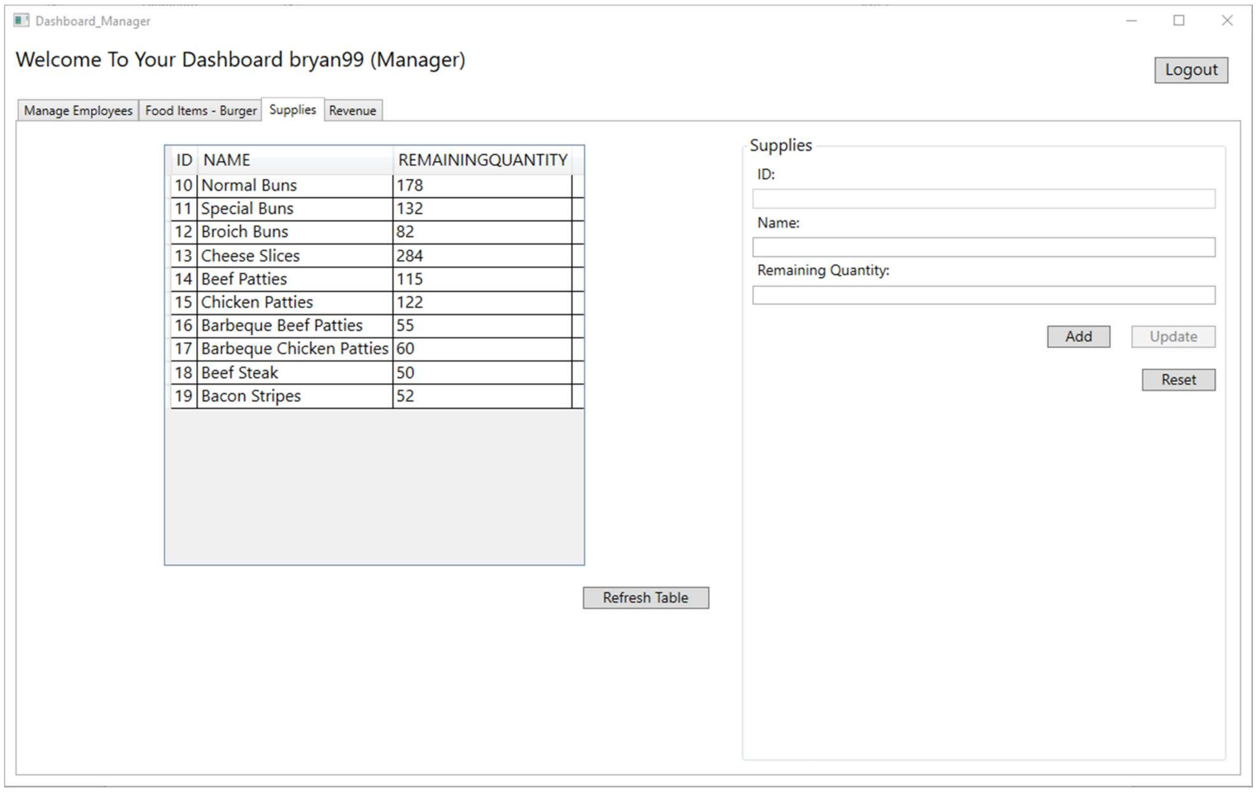


Dashboard_Manager

Welcome To Your Dashboard bryan99 (Manager)

ID	NAME	BUNS	MEATPATTIES	CHEESE
10	Chicken Burger	Normal Buns	Chicken Patty	No
11	Beef Burger	Normal Buns	Beef Patty	No
12	Chicken Cheese Burger	Normal Buns	Chicken Patty	Yes
13	Beef Cheese Burger	Normal Buns	Beef Patty	Yes
14	Barbeque Chicken Burger	Special Buns	Barbeque Chicken Patty	No
15	Barbeque Beef Burger	Special Buns	Barbeque Beef Patty	No
16	Double Decker Beef Burger	Special Buns	Double Beef Patty	Yes
17	Chicken And Bacon Burger	Broich Buns	Chicken Patty and Bacon	Yes
18	Beef Steak Burger	Broich Buns	Beef Steak Patty	Yes
19	Beef Steak And Bacon Burger	Broich Buns	Beef Steak Patty and Bacon	Yes

Description: The manager can manage the food items served in the burger shop

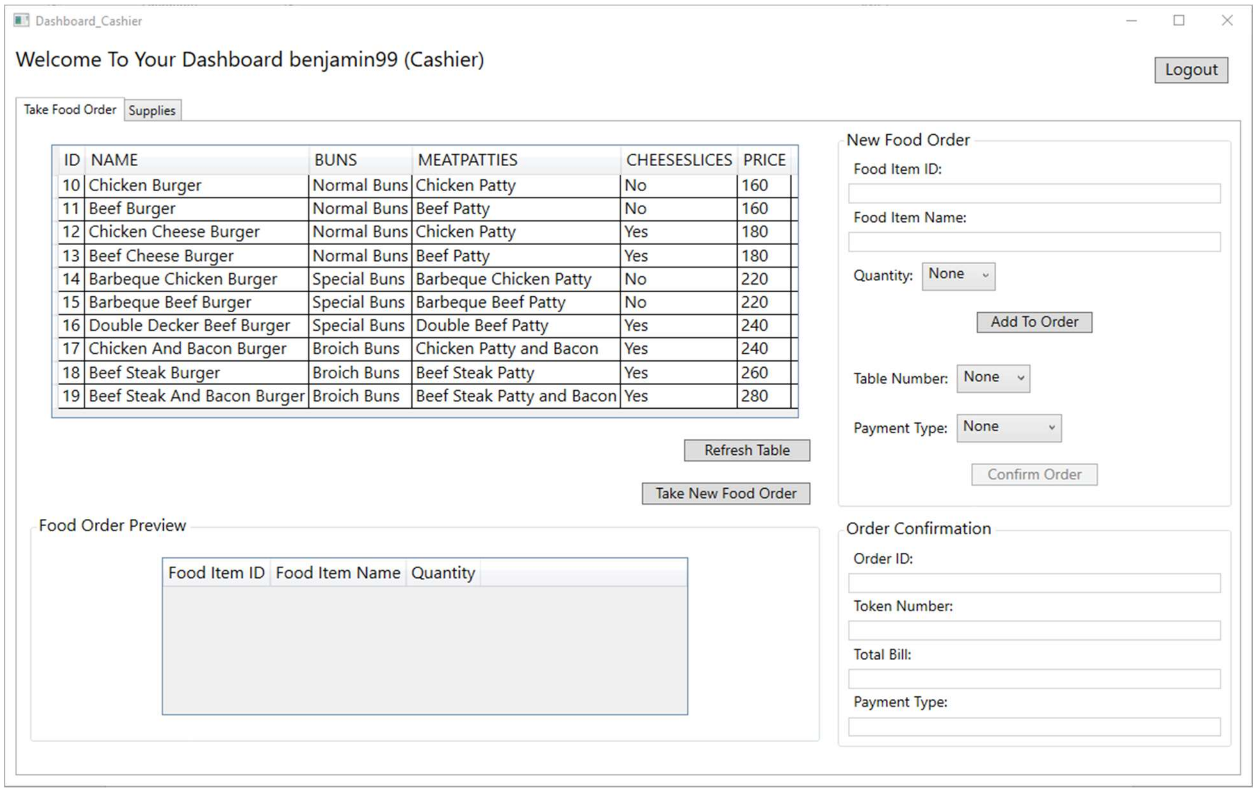


Dashboard_Manager

Welcome To Your Dashboard bryan99 (Manager)

ID	NAME	REMAININGQUANTITY
10	Normal Buns	178
11	Special Buns	132
12	Broich Buns	82
13	Cheese Slices	284
14	Beef Patties	115
15	Chicken Patties	122
16	Barbeque Beef Patties	55
17	Barbeque Chicken Patties	60
18	Beef Steak	50
19	Bacon Stripes	52

Description: The manager can also manage the necessary food supplies

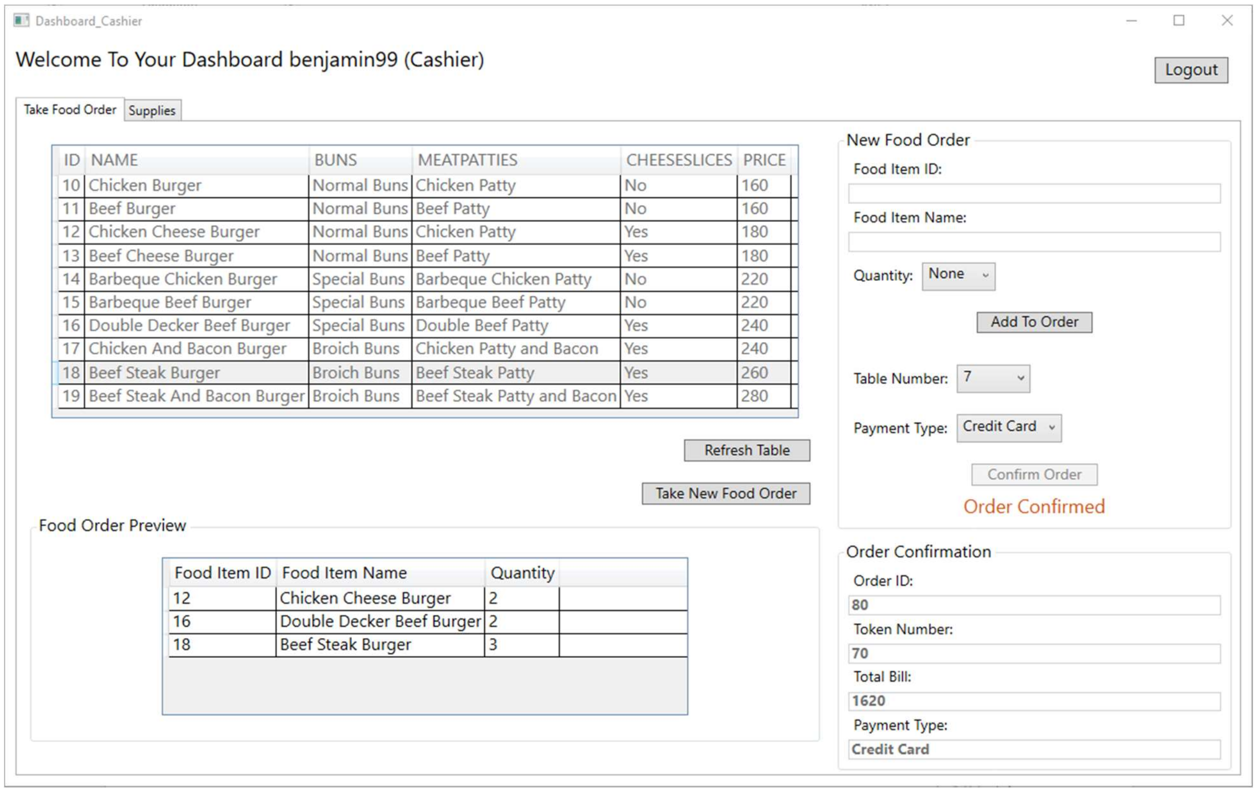


Dashboard_Cashier

Welcome To Your Dashboard benjamin99 (Cashier)

ID	NAME	BUNS	MEATPATTIES	CHEESESLICES	PRICE
10	Chicken Burger	Normal Buns	Chicken Patty	No	160
11	Beef Burger	Normal Buns	Beef Patty	No	160
12	Chicken Cheese Burger	Normal Buns	Chicken Patty	Yes	180
13	Beef Cheese Burger	Normal Buns	Beef Patty	Yes	180
14	Barbeque Chicken Burger	Special Buns	Barbeque Chicken Patty	No	220
15	Barbeque Beef Burger	Special Buns	Barbeque Beef Patty	No	220
16	Double Decker Beef Burger	Special Buns	Double Beef Patty	Yes	240
17	Chicken And Bacon Burger	Broich Buns	Chicken Patty and Bacon	Yes	240
18	Beef Steak Burger	Broich Buns	Beef Steak Patty	Yes	260
19	Beef Steak And Bacon Burger	Broich Buns	Beef Steak Patty and Bacon	Yes	280

Description: The cashier can take food orders from the customer in this window



Dashboard_Cashier

Welcome To Your Dashboard benjamin99 (Cashier)

ID	NAME	BUNS	MEATPATTIES	CHEESESLICES	PRICE
10	Chicken Burger	Normal Buns	Chicken Patty	No	160
11	Beef Burger	Normal Buns	Beef Patty	No	160
12	Chicken Cheese Burger	Normal Buns	Chicken Patty	Yes	180
13	Beef Cheese Burger	Normal Buns	Beef Patty	Yes	180
14	Barbeque Chicken Burger	Special Buns	Barbeque Chicken Patty	No	220
15	Barbeque Beef Burger	Special Buns	Barbeque Beef Patty	No	220
16	Double Decker Beef Burger	Special Buns	Double Beef Patty	Yes	240
17	Chicken And Bacon Burger	Broich Buns	Chicken Patty and Bacon	Yes	240
18	Beef Steak Burger	Broich Buns	Beef Steak Patty	Yes	260
19	Beef Steak And Bacon Burger	Broich Buns	Beef Steak Patty and Bacon	Yes	280

Food Order Preview

Food Item ID	Food Item Name	Quantity
12	Chicken Cheese Burger	2
16	Double Decker Beef Burger	2
18	Beef Steak Burger	3

New Food Order

Food Item ID:

Food Item Name:

Quantity: None

Table Number: 7

Payment Type: Credit Card

Order Confirmed

Order Confirmation

Order ID: 80

Token Number: 70

Total Bill: 1620

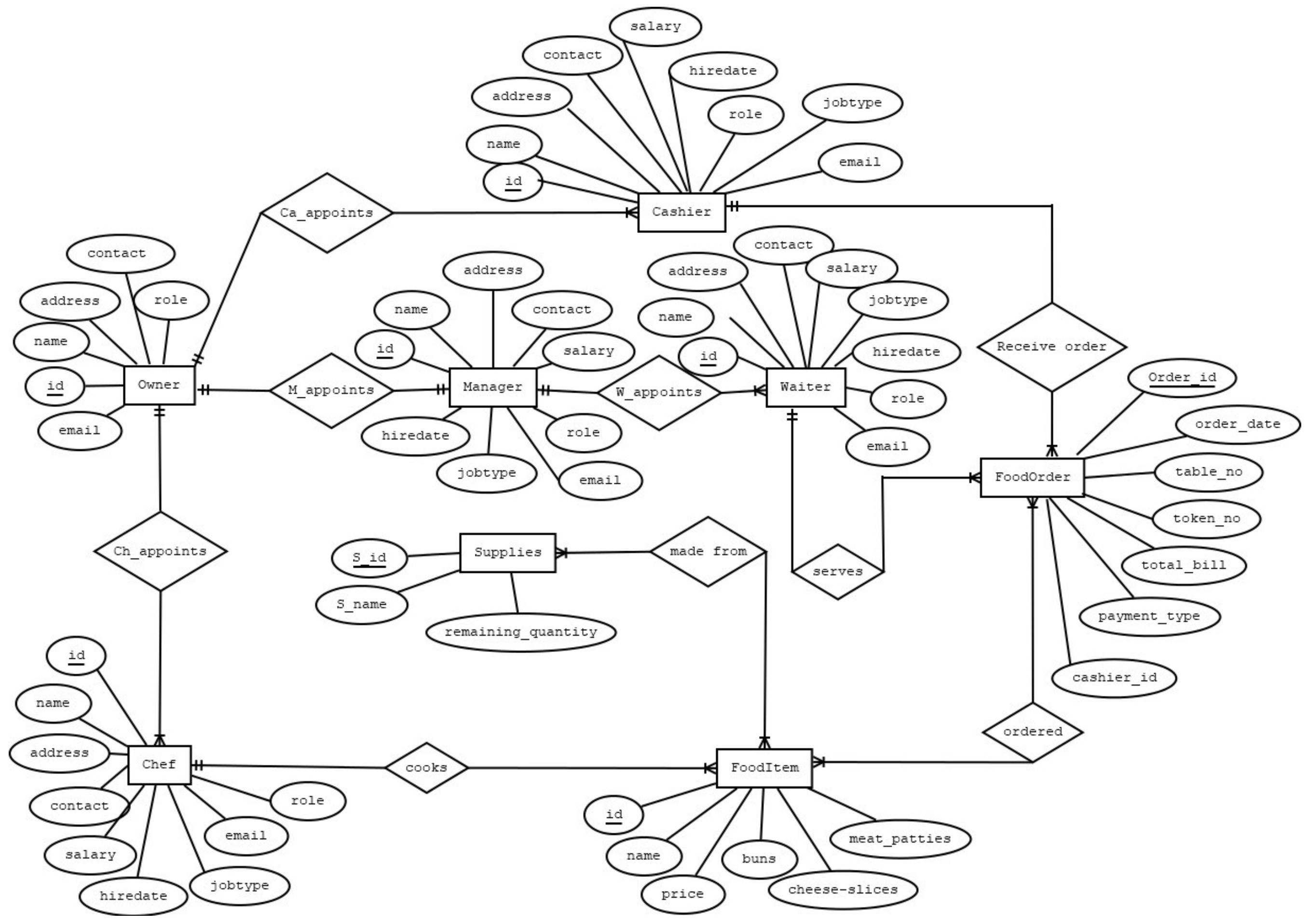
Payment Type: Credit Card

Description: The food order preview shows the food items ordered by the customer. After the order is confirmed, the token number, total bill and the payment type is displayed in the order confirmation.

Scenario Description

This burger shop has an owner and a group of employees with different roles. The roles are manager, waiter, chef and cashier. The management software for this burger shop holds information about the roles of the employees along with their unique id, name, job type (part time, full time), salary, email, address and contact number. The main food item in this burger shop are burgers where information about different types of burgers that are served here are stored in the management software. The different types of burgers served here have a unique id, name, types of buns/bread, types of meat patties, number of cheese slices and their price. Specifically, the users of this software system are the owner, manager and cashier. The owner has the highest privilege. He has access to all the information and can modify any information. The manager has less privilege than the owner. He has access to some limited information and also can modify them. The manager only has access to information about the waiters, the chefs and the cashiers. He can also manage the food items. The cashier has the least privilege in the management software. The payment method in this burger shop is pay first and only the cashier takes food orders from the customers. The management software records some specific information for every food order that is taken by the cashier. These are order id, order date, table number, token number, total payment/bill, payment type (cash, credit card) and the unique id of the cashier who took the order from the customer. When a customer orders some food items, the cashier selects those food items and the prices of those individual food items get added up to make the total payment/bill for that specific order. Supplies are needed to make the food items that are served in this burger shop. Information about the food supplies are also stored in the management software. The food supplies have their unique id, name (buns/breads, cheese slices etc.) and the remaining quantity. Every time the customers of this burger shop order some food items, the supplies that are needed to make those food items are also consumed, that is, the quantity of the supplies gets deducted. If any of the supplies to make a food item is fully consumed, that is, the quantity becomes zero then the owner or the manager has the privilege to update the quantity of the supplies so that more food orders can be taken.

ER Diagram



Normalization

Receive Order (Cashier, FoodOrder):

UNF:
Receive Order (ca_id, name, role, hiredate, job_type, salary, email, address, contact_number, order_id, date, table_number, token_number, total_bill, payment_type)

1NF:
There is no multivalued attribute
1. (ca_id, name, role, hiredate, job_type, salary, email, address, contact_number, order_id, date, table_number, token_number, total_bill, payment_type)

2NF:
1. (ca_id, name, role, hiredate, job_type, salary, email, address, contact_number)
2. (order_id, date, table_number, token_number, total_bill, payment_type)

3NF:
There is no transitive dependency
1. (ca_id, name, role, hiredate, job_type, salary, email, address, contact_number)
2. (order_id, date, table_number, token_number, total_bill, payment_type)

Table Creation (One to Many):
1. (ca_id, name, role, hiredate, job_type, salary, email, address, contact_number)
2. (order_id, date, table_number, token_number, total_bill, payment_type, **ca_id**)

Ordered (FoodOrder, FoodItem):

UNF:
Ordered (order_id, date, table_number, token_number, total_bill, payment_type, food_id, name, buns, meat_patties, cheese_slices, price)

1NF:
There is no multivalued attribute
1. (order_id, date, table_number, token_number, total_bill, payment_type, food_id, name, buns, meat_patties, cheese_slices, price)

2NF:
1. (order_id, date, table_number, token_number, total_bill, payment_type)
2. (food_id, name, buns, meat_patties, cheese_slices, price)

3NF:
There is no transitive dependency
1. (order_id, date, table_number, token_number, total_bill, payment_type)
2. (food_id, name, buns, meat_patties, cheese_slices, price)

Table Creation (Many to Many):
1. (order_id, date, table_number, token_number, total_bill, payment_type)
2. (food_id, name, buns, meat_patties, cheese_slices, price)
3. (id, **order_id**, **food_id**, quantity)

Made From (FoodItem, Supplies):

UNF:
Made From (food_id, name, buns, meat_patties, cheese_slices, price, supply_id, name, remaining_quantity)

1NF:
There is no multivalued attribute
1. (food_id, name, buns, meat_patties, cheese_slices, price, supply_id, name, remaining_quantity)

2NF:
1. (food_id, name, buns, meat_patties, cheese_slices, price)
2. (supply_id, name, remaining_quantity)

3NF:
There is no transitive dependency
1. (food_id, name, buns, meat_patties, cheese_slices, price)
2. (supply_id, name, remaining_quantity)

Table Creation (Many to Many):
1. (food_id, name, buns, meat_patties, cheese_slices, price)
2. (supply_id, name, remaining_quantity)
3. (id, **food_id**, **supply_id**, quantity_deduction)

Serves (Waiter, FoodOrder):

UNF:
Serves (w_id, name, role, hiredate, job_type, salary, email, address, contact_number, order_id, date, table_number, token_number, total_bill, payment_type)

1NF:
There is no multivalued attribute
1. (w_id, name, role, hiredate, job_type, salary, email, address, contact_number, order_id, date, table_number, token_number, total_bill, payment_type)

2NF:
1. (w_id, name, role, hiredate, job_type, salary, email, address, contact_number)
2. (order_id, date, table_number, token_number, total_bill, payment_type)

3NF:
1. (w_id, name, role, hiredate, job_type, salary, email, address, contact_number)
2. (order_id, date, table_number, token_number, total_bill, payment_type)

Table Creation (One to Many):
1. (w_id, name, role, hiredate, job_type, salary, email, address, contact_number)
2. (order_id, date, table_number, token_number, total_bill, payment_type, **w_id**)

Cooks (Chef, FoodItem):

UNF:
Cooks (ch_id, name, role, hiredate, job_type, salary, email, address, contact_number, food_id, name, buns, meat_patties, cheese_slices, price)

1NF:
There is no multivalued attribute
1. (ch_id, name, role, hiredate, job_type, salary, email, address, contact_number, food_id, name, buns, meat_patties, cheese_slices, price)

2NF:
1. (ch_id, name, role, hiredate, job_type, salary, email, address, contact_number)
2. (food_id, name, buns, meat_patties, cheese_slices, price)

3NF:

There is no transitive dependency

1. (ch_id, name, role, hiredate, job_type, salary, email, address, contact_number)
2. (food_id, name, buns, meat_patties, cheese_slices, price)

Table Creation (One to Many):

1. (ch_id, name, role, hiredate, job_type, salary, email, address, contact_number)
2. (food_id, name, buns, meat_patties, cheese_slices, price, **ch_id**)

Ca-appoints(Owner,Cashier):

UNF:

Ca-appoints (o_id, o_name, address, email, phone, role, ca_id, ca_name, address, phone, email, salary, hiredate, role, jobtype).

1NF:

Phone is multivalued attribute

1.(o_id,o_name,address,email,role,ca_id,ca_name,address,email,salary,hiredate,role,

jobtype).

2NF:

1. (o_id,o_name,address,email,role)

2. (ca_id,ca_name,address,email,salary,hiredate,role,jobtype)

3NF:

There is no transitive dependency

1. (o_id,o_name,address,email,role)

2. (ca_id,ca_name,address,email,salary,hiredate,role,jobtype)

Table Creation(One to Many):

1. (o_id,o_name,address,email,role)

2. (ca_id,ca_name,address ,email,salary,hiredate,role,jobtype,**o_id**)

M_appoints(owner,manager)

UNF:

M_appoints(o_id,o_name,address,email,phone,role,M_id,M_name,address,email,phone,salary,hiredate,

jobtype,role)

1NF:

Phone is multivalued attribute.

1. (o_id,o_name,address,email,role,M_id,M_name,address,email,salary,hiredate,jobtype,role)

2NF:

1.(o_id,o_name,address,email,role)

2.(M_id,M_name,address,email,salary,hiredate,jobtype,role)

3NF:

There is no transitive dependency

1.(o_id,o_name,address,email,role)

2. (M_id,M_name,address,email,salary,hiredate,jobtype,role)

Table Creation: (One to One)

1.(o_id,o_name,address,email,role,**M_id**)

2. (M_id,M_name,address,email,salary,hiredate,jobtype,role)

Ch_appoints (owner,chef)

UNF:

Ch_appoints(o_id,o_name,address,email,phone,role,Ch_id,Ch_name,address,email,phone,salary,

hiredate,jobtype,role)

1NF:

Phone is multivalued attribute.

1. (o_id,o_name,address,email,role,Ch_id,Ch_name,address,email,salary,hiredate,jobtype,role)

2NF:

1.(o_id,o_name,address,email,role)

2.(Ch_id,Ch_name,address,email,salary,hiredate,jobtype,role)

3NF:

There is no transitive dependency

1. (o_id,o_name,address,email,role)

2.(Ch_id,Ch_name,address,email,salary,hiredate,jobtype,role)

Table Creation: (One to Many)

1. (o_id,o_name,address,email,role)

2.(Ch_id,Ch_name,address,email,salary,hiredate,jobtype,role,**o_id**)

W_appoints (manager,waiter)

UNF:

W_appoints(M_id,M_name,address,email,phone,salary,hiredate,jobtype,role,W_id,W_name,address,

phone,email,salary,hiredate,jobtype,role)

1NF:

Phone is multivalued attribute.

1.(M_id,M_name,address,email,salary,hiredate,jobtype,role,W_id,W_name,address, email,salary,hiredate,jobtype,role)

2NF:

1.(M_id,M_name,address,email,salary,hiredate,jobtype,role)

2.(W_id,W_name,address, email,salary,hiredate,jobtype,role)

3NF:

There is no transitive dependency

1. (M_id,M_name,address,email,salary,hiredate,jobtype,role)

2.(W_id,W_name,address, email,salary,hiredate,jobtype,role)

Table Creation: (One to Many)

1. (M_id,M_name,address,email,salary,hiredate,jobtype,role)

2.(W_id,W_name,address, email,salary,hiredate,jobtype,role,**M_id**)

Final Tables:

1. (order_id, date, table_number, token_number, total_bill, payment_type, **ca_id**)

2. (<u>food_id</u>, name, buns, meat_patties, cheese_slices, price)

3. (<u>id</u>, **order_id**, **food_id**, quantity)

4. (<u>supply_id</u>, name, remaining_quantity)

5. (<u>id</u>, **food_id**, **supply_id**, quantity_deduction)

6. (<u>owner_id</u>, name, role, email, address, contact_number)

7. (<u>emp_id</u>, name, role, hiredate, job_type, salary, email, address, contact_number)

Schema Diagram

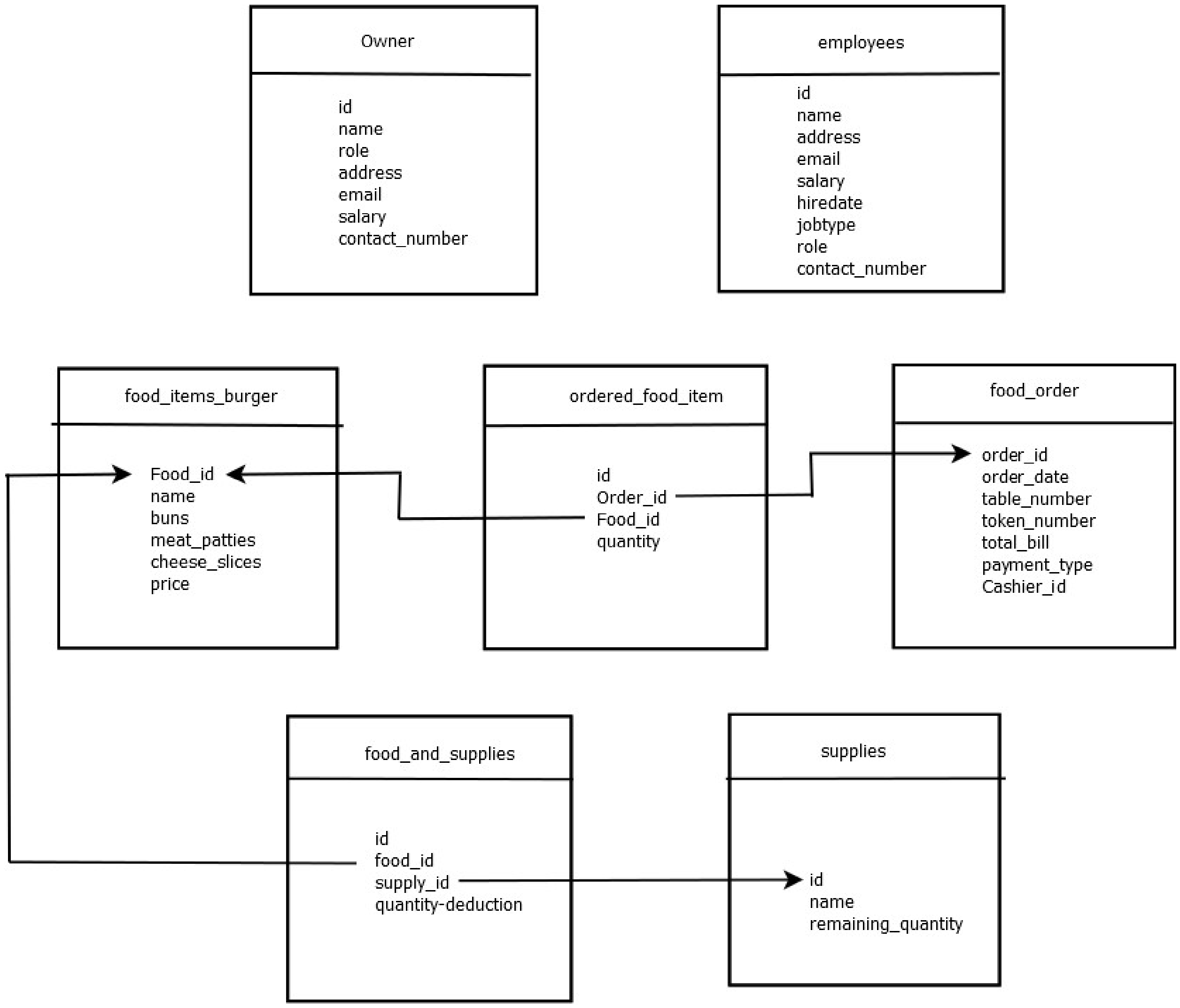


Table Creation

Creating a User with the Necessary Privileges:

All the necessary tables, views, sequences, procedures, functions, triggers etc. will be created with this user.

```
create user burgershop identified by burgershop;

create role custompermissions;

grant create session, create table, create view, create sequence, create
synonym to custompermissions;

grant custompermissions to burgershop;

grant create procedure to burgershop;

grant create trigger to burgershop;

grant unlimited tablespace to burgershop;
```

Creating the Tables with the Necessary Alterations:

```
1. create table owner(

    id number(10) primary key,

    name varchar2(50),

    role varchar2(50),

    email varchar2(50),

    address varchar2(50),

    contact_number number(20)

);

alter table owner add constraint unique_owner_email unique(email);

alter table owner add constraint unique_owner_contact_number
unique(contact_number);

2. create table employees(

    id number(10) primary key,

    name varchar2(50),
    role varchar2(50),

    hiredate date,

    job_type varchar2(50),

    salary number(10),

    email varchar2(50),

    address varchar2(50),

    contact_number number(20)

);

alter table employees add constraint unique_employees_email unique(email);

alter table employees add constraint unique_employees_contact_num
unique(contact_number);
```

3. `create table supplies(`

```
    id number(10) primary key,

    name varchar2(50),

    remaining_quantity number(10)

);
```

4. `create table food_items_burger(`

```
    id number(10) primary key,

    name varchar2(50),

    buns varchar2(50),

    meat_patties varchar2(50),

    cheese_slices varchar2(50),

    price number(10)

);
```

5. `create table food_order(`

```
    id number(10) primary key,

    order_date date,

    table_number number(10),

    token_number number(10),
    total_bill number(10),

    payment_type varchar2(50),

    cashier_id number(10)

);

alter table food_order add constraint fk_cashier_id foreign key (cashier_id)
references employees (id);

6. create table ordered_food_items(

    id number(10) primary key,

    food_order_id number(10),

    food_items_id number(10),

    quantity number(10)

);

alter table ordered_food_items add constraint fk_foodorderid foreign key
(food_order_id) references food_order (id);

alter table ordered_food_items add constraint fk_fooditemsid foreign key
(food_items_id) references food_items_burger (id);

7. create table food_and_supplies(

    id number(10) primary key,

    food_items_id number(10),

    supply_id number(10),

    quantity_deduction number(10)

);

alter table food_and_supplies add constraint fk_fooditemsid2 foreign key
(food_items_id) references food_items_burger (id);

alter table food_and_supplies add constraint fk_supplyid foreign key
(supply_id) references supplies (id);
```

Creating the Sequences Necessary for the Tables above:

```
1. create sequence employees_seq

      start with 10
```

```
  increment by 1
  nocache
  nocycle;
```

2. create sequence supplies_seq

```
 start with 10
 increment by 1
 nocache
 nocycle;
```

3. create sequence food_items_seq

```
 start with 10
 increment by 1
 nocache
 nocycle;
```

4. create sequence food_order_seq

```
 start with 10
 increment by 1
 nocache
 nocycle;
```

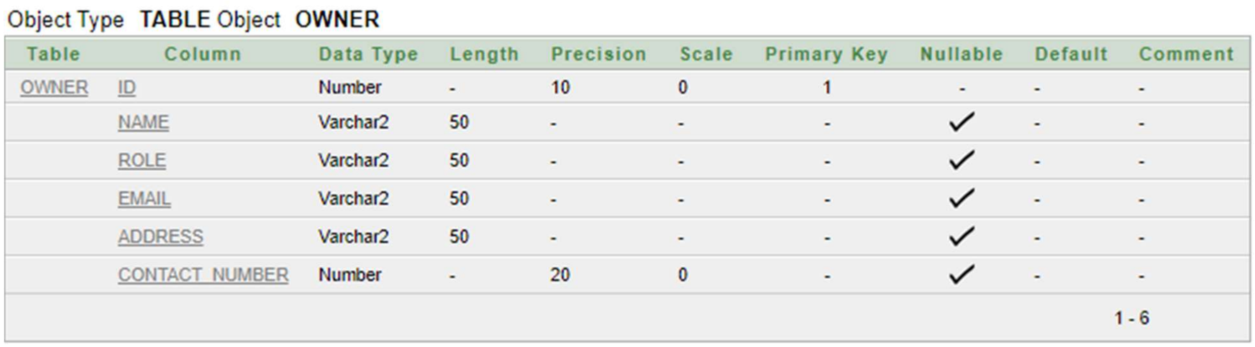
5. create sequence token_seq

```
 start with 10
 increment by 1
 nocache
 nocycle;
```

6. create sequence general_seq

```
 start with 10
 increment by 1
 nocache
 nocycle;
```

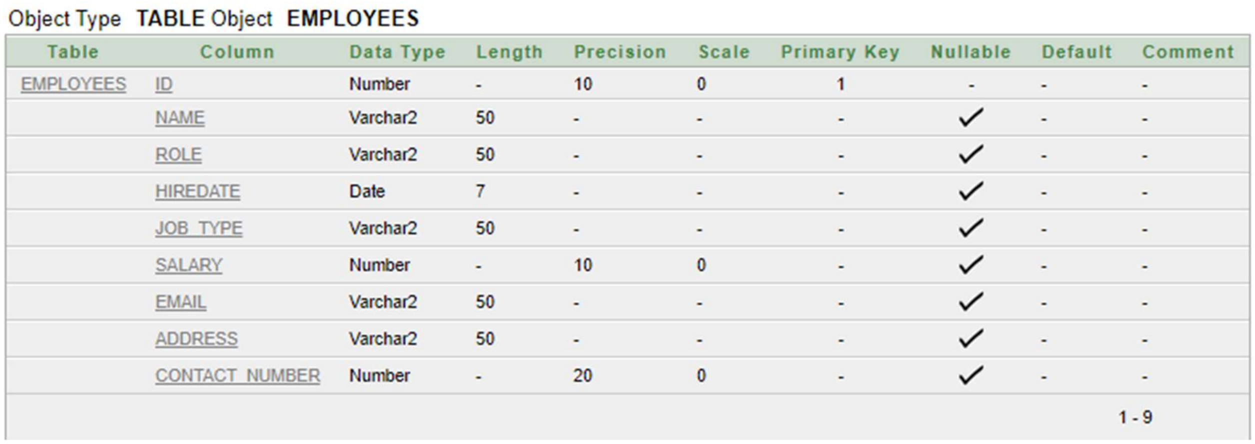
Screenshots of the Created Tables:



Object Type TABLE Object OWNER

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
OWNER	ID	Number	-	10	0	1	-	-	-
	NAME	Varchar2	50	-	-	-	✓	-	-
	ROLE	Varchar2	50	-	-	-	✓	-	-
	EMAIL	Varchar2	50	-	-	-	✓	-	-
	ADDRESS	Varchar2	50	-	-	-	✓	-	-
	CONTACT_NUMBER	Number	-	20	0	-	✓	-	-

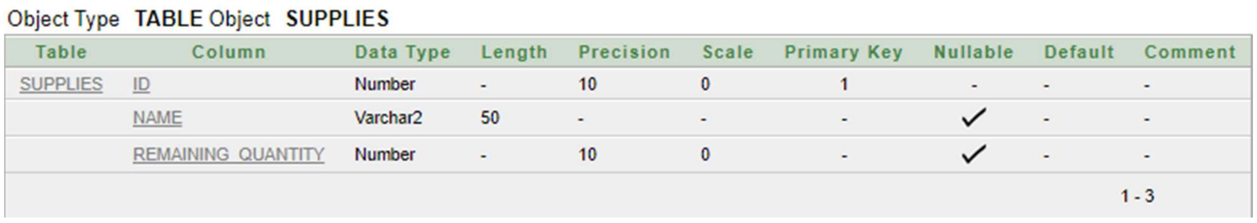
1 - 6



Object Type TABLE Object EMPLOYEES

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPLOYEES	ID	Number	-	10	0	1	-	-	-
	NAME	Varchar2	50	-	-	-	✓	-	-
	ROLE	Varchar2	50	-	-	-	✓	-	-
	HIREDATE	Date	7	-	-	-	✓	-	-
	JOB_TYPE	Varchar2	50	-	-	-	✓	-	-
	SALARY	Number	-	10	0	-	✓	-	-
	EMAIL	Varchar2	50	-	-	-	✓	-	-
	ADDRESS	Varchar2	50	-	-	-	✓	-	-
	CONTACT_NUMBER	Number	-	20	0	-	✓	-	-

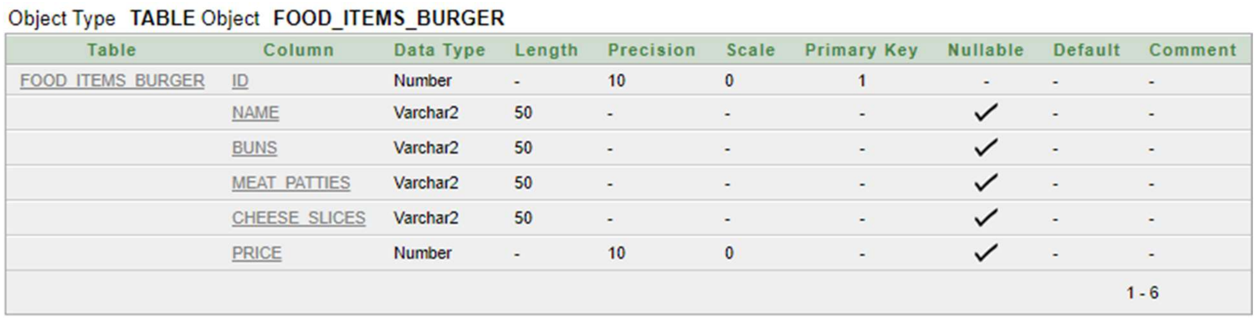
1 - 9



Object Type TABLE Object SUPPLIES

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
SUPPLIES	ID	Number	-	10	0	1	-	-	-
	NAME	Varchar2	50	-	-	-	✓	-	-
	REMAINING_QUANTITY	Number	-	10	0	-	✓	-	-

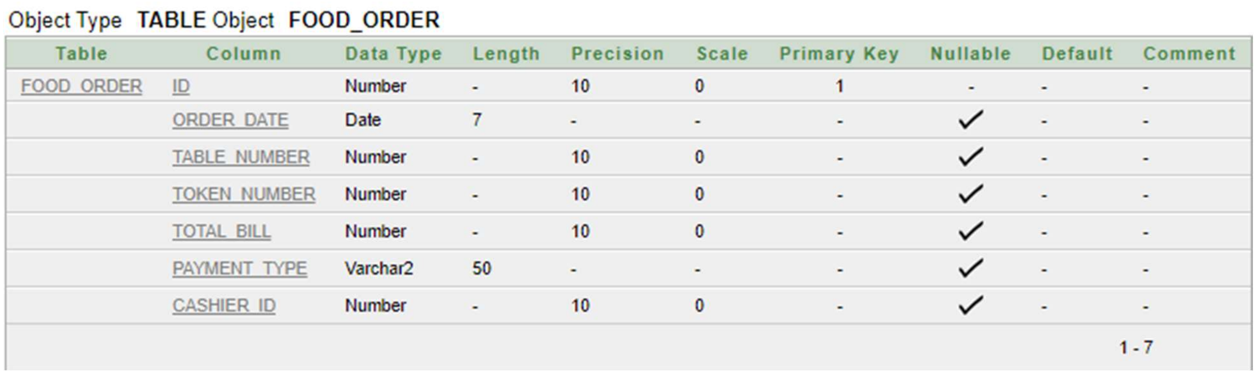
1 - 3



Object Type TABLE Object FOOD_ITEMS_BURGER

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
FOOD_ITEMS_BURGER	ID	Number	-	10	0	1	-	-	-
	NAME	Varchar2	50	-	-	-	✓	-	-
	BUNS	Varchar2	50	-	-	-	✓	-	-
	MEAT_PATTIES	Varchar2	50	-	-	-	✓	-	-
	CHEESE_SLICES	Varchar2	50	-	-	-	✓	-	-
	PRICE	Number	-	10	0	-	✓	-	-

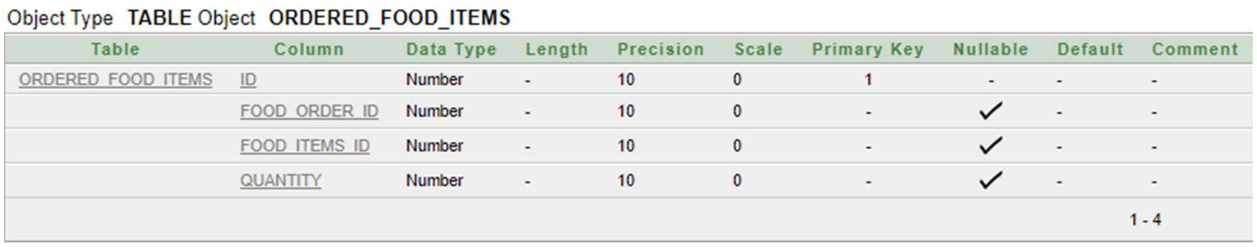
1 - 6



Object Type TABLE Object FOOD_ORDER

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
FOOD_ORDER	ID	Number	-	10	0	1	-	-	-
	ORDER_DATE	Date	7	-	-	-	✓	-	-
	TABLE_NUMBER	Number	-	10	0	-	✓	-	-
	TOKEN_NUMBER	Number	-	10	0	-	✓	-	-
	TOTAL_BILL	Number	-	10	0	-	✓	-	-
	PAYMENT_TYPE	Varchar2	50	-	-	-	✓	-	-
	CASHIER_ID	Number	-	10	0	-	✓	-	-

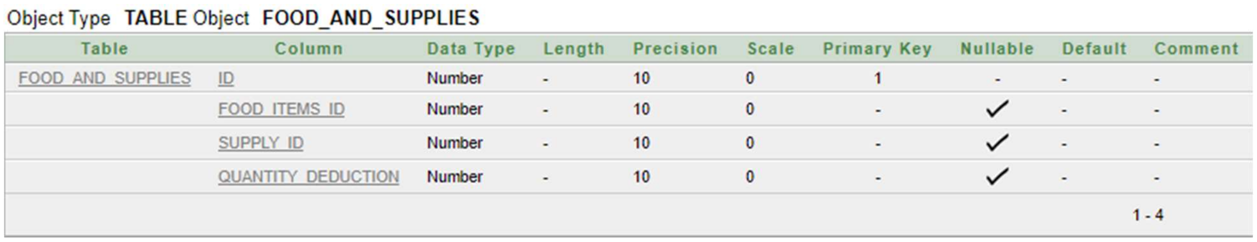
1 - 7



Object Type TABLE Object ORDERED_FOOD_ITEMS

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
ORDERED_FOOD_ITEMS	ID	Number	-	10	0	1	-	-	-
	FOOD_ORDER_ID	Number	-	10	0	-	✓	-	-
	FOOD_ITEMS_ID	Number	-	10	0	-	✓	-	-
	QUANTITY	Number	-	10	0	-	✓	-	-

1 - 4



Object Type TABLE Object FOOD_AND_SUPPLIES

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
FOOD_AND_SUPPLIES	ID	Number	-	10	0	1	-	-	-
	FOOD_ITEMS_ID	Number	-	10	0	-	✓	-	-
	SUPPLY_ID	Number	-	10	0	-	✓	-	-
	QUANTITY_DEDUCTION	Number	-	10	0	-	✓	-	-

1 - 4

Data Insertion:

1. Owner Table: There is only one row in this table.

```
insert into owner values (99, 'Joseph', 'Owner', 'joseph@gmail.com',
'Shantinagar', 111111);
```

2. Employees Table:

```
insert into employees values (employees_seq.nextval, 'Bryan', 'Manager', '02-
Apr-20', 'Full Time', 35000, 'bryan@gmail.com', 'Basabo', 111222);
```

```
insert into employees values (employees_seq.nextval, 'Jimmy', 'Cashier', '02-
Apr-20', 'Full Time', 20000, 'jimmy@gmail.com', 'Khilgaon', 111223);
```

```
insert into employees values (employees_seq.nextval, 'Chris', 'Cashier', '07-
Apr-20', 'Part Time', 10000, 'chris@gmail.com', 'Uttara', 111224);

insert into employees values (employees_seq.nextval, 'John', 'Chef', '02-Apr-
20', 'Full Time', 24000, 'john@gmail.com', 'Gulshan', 111225);

insert into employees values (employees_seq.nextval, 'Kevin', 'Chef', '07-
Apr-20', 'Part Time', 12000, 'kevin@gmail.com', 'Banani', 111226);

insert into employees values (employees_seq.nextval, 'Nicholas', 'Waiter',
'02-Apr-20', 'Full Time', 16000, 'nicholas@gmail.com', 'Rampura', 111227);

insert into employees values (employees_seq.nextval, 'Ben', 'Waiter', '02-
Apr-20', 'Full Time', 16000, 'ben@gmail.com', 'Badda', 111228);

insert into employees values (employees_seq.nextval, 'Smith', 'Waiter', '07-
Apr-20', 'Part Time', 8000, 'smith@gmail.com', 'Bashundhara', 111229);

insert into employees values (employees_seq.nextval, 'Lynch', 'Waiter', '07-
Apr-20', 'Part Time', 8000, 'lynch@gmail.com', 'Mirpur', 111220);
```

3. Supplies Table:

```
insert into supplies values (supplies_seq.nextval, 'Normal Buns', 250);

insert into supplies values (supplies_seq.nextval, 'Special Buns', 180);

insert into supplies values (supplies_seq.nextval, 'Broich Buns', 120);

insert into supplies values (supplies_seq.nextval, 'Cheese Slices', 400);

insert into supplies values (supplies_seq.nextval, 'Beef Patties', 150);

insert into supplies values (supplies_seq.nextval, 'Chicken Patties', 150);

insert into supplies values (supplies_seq.nextval, 'Barbeque Beef Patties',
60);

insert into supplies values (supplies_seq.nextval, 'Barbeque Chicken
Patties', 60);

insert into supplies values (supplies_seq.nextval, 'Beef Steak', 40);

insert into supplies values (supplies_seq.nextval, 'Bacon Stripes', 30);
```

4. Food_Items_Burger Table:

```
insert into food_items_burger values (food_items_seq.nextval, 'Chicken
Burger', 'Normal Buns', 'Chicken Patty', 'No', 160);

insert into food_items_burger values (food_items_seq.nextval, 'Beef Burger',
'Normal Buns', 'Beef Patty', 'No', 160);

insert into food_items_burger values (food_items_seq.nextval, 'Chicken Cheese
Burger', 'Normal Buns', 'Chicken Patty', 'Yes', 180);

insert into food_items_burger values (food_items_seq.nextval, 'Beef Cheese
Burger', 'Normal Buns', 'Beef Patty', 'Yes', 180);

insert into food_items_burger values (food_items_seq.nextval, 'Barbeque
Chicken Burger', 'Special Buns', 'Barbeque Chicken Patty', 'No', 220);

insert into food_items_burger values (food_items_seq.nextval, 'Barbeque Beef
Burger', 'Special Buns', 'Barbeque Beef Patty', 'No', 220);

insert into food_items_burger values (food_items_seq.nextval, 'Double Decker
Beef Burger', 'Special Buns', 'Double Beef Patty', 'Yes', 240);

insert into food_items_burger values (food_items_seq.nextval, 'Chicken And
Bacon Burger', 'Broich Buns', 'Chicken Patty and Bacon', 'Yes', 240);

insert into food_items_burger values (food_items_seq.nextval, 'Beef Steak
Burger', 'Broich Buns', 'Beef Steak Patty', 'Yes', 260);
```

5. Food_Order Table:

```
insert into food_order values (food_order_seq.nextval, '02-Apr-20', 5,
token_seq.nextval, 320, 'Cash', 12);

insert into food_order values (food_order_seq.nextval, '02-Apr-20', 2,
token_seq.nextval, 380, 'Credit Card', 12);

insert into food_order values (food_order_seq.nextval, '03-Apr-20', 1,
token_seq.nextval, 360, 'Cash', 12);

insert into food_order values (food_order_seq.nextval, '03-Apr-20', 3,
token_seq.nextval, 680, 'Cash', 13);

insert into food_order values (food_order_seq.nextval, '03-Apr-20', 5,
token_seq.nextval, 1260, 'Credit Card', 13);
```

6. Ordered_Food_Items Table:

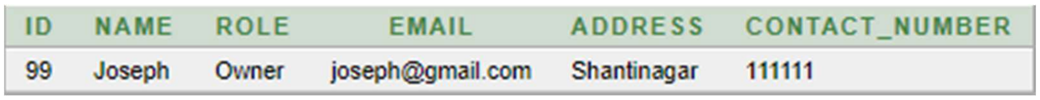
```
insert into ordered_food_items values (general_seq.nextval, 10, 10, 2);

insert into ordered_food_items values (general_seq.nextval, 11, 11, 1);

insert into ordered_food_items values (general_seq.nextval, 11, 14, 1);

insert into ordered_food_items values (general_seq.nextval, 12, 13, 2);

insert into ordered_food_items values (general_seq.nextval, 13, 14, 2);

insert into ordered_food_items values (general_seq.nextval, 13, 16, 1);

insert into ordered_food_items values (general_seq.nextval, 14, 17, 2);

insert into ordered_food_items values (general_seq.nextval, 14, 18, 3);
```

7. Food_And_Supplies table:

```
insert into food_and_supplies values (general_seq.nextval, 16, 12, 2);

insert into food_and_supplies values (general_seq.nextval, 16, 15, 2);

insert into food_and_supplies values (general_seq.nextval, 16, 14, 3);

insert into food_and_supplies values (general_seq.nextval, 17, 13, 2);

insert into food_and_supplies values (general_seq.nextval, 17, 16, 1);

insert into food_and_supplies values (general_seq.nextval, 17, 20, 3);

insert into food_and_supplies values (general_seq.nextval, 17, 14, 2);
```

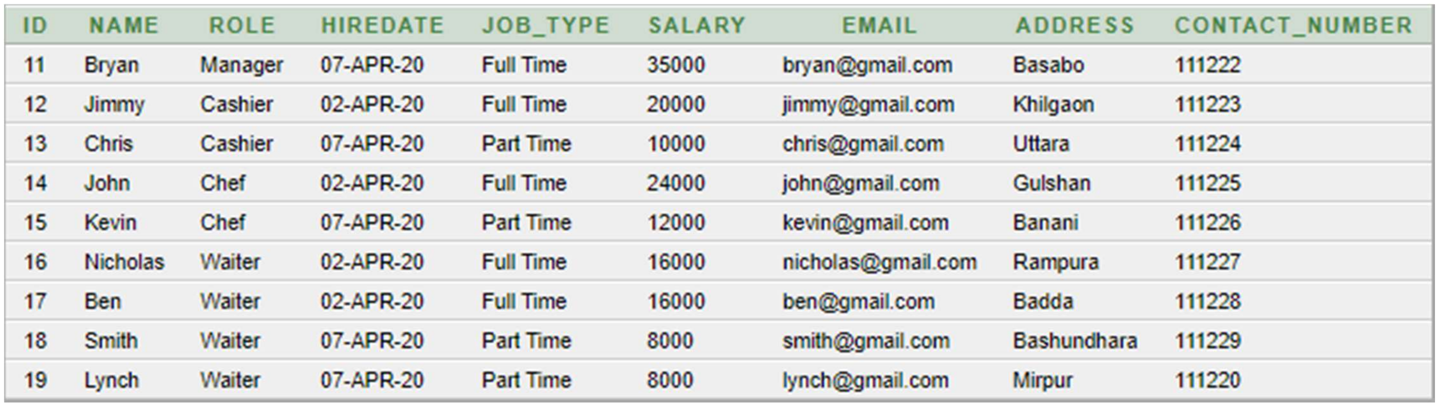
Screenshots of the Tables after Data Insertion:

1. Owner Table:



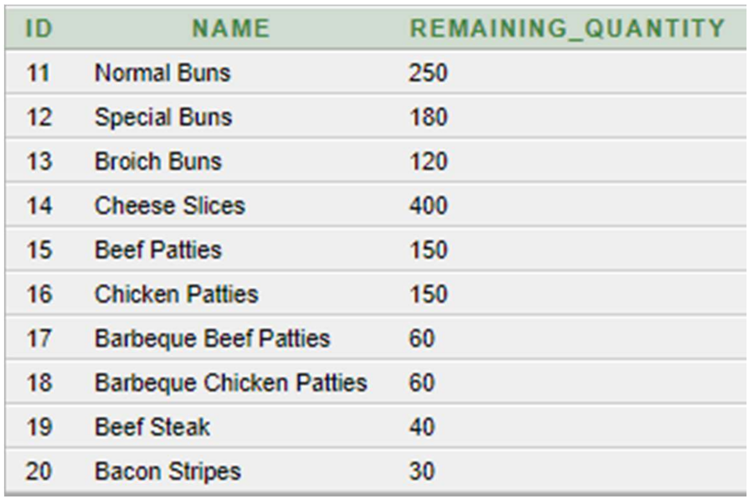
ID	NAME	ROLE	EMAIL	ADDRESS	CONTACT_NUMBER
99	Joseph	Owner	joseph@gmail.com	Shantinagar	111111

2. Employees Table:



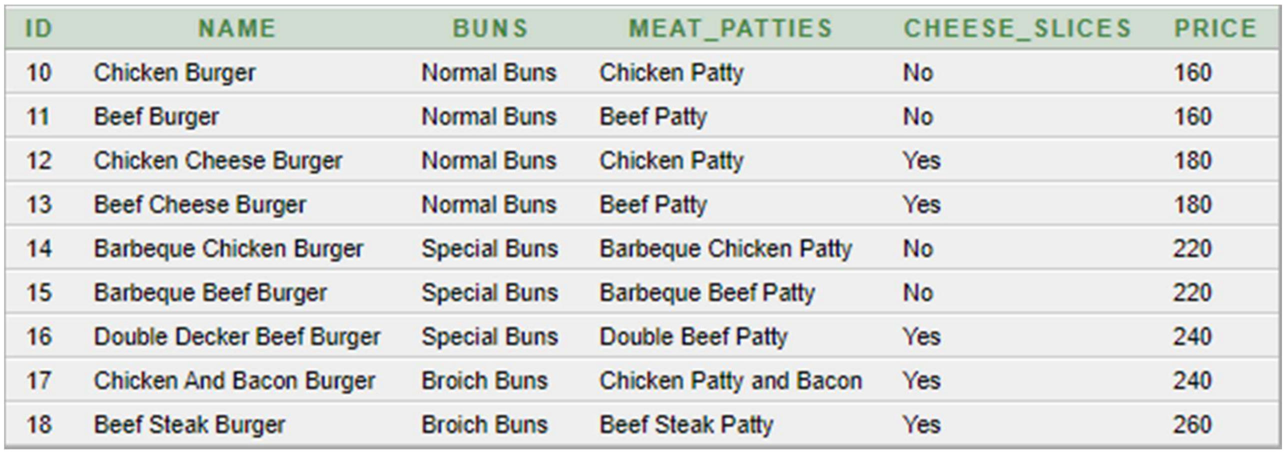
ID	NAME	ROLE	HIREDATE	JOB_TYPE	SALARY	EMAIL	ADDRESS	CONTACT_NUMBER
11	Bryan	Manager	07-APR-20	Full Time	35000	bryan@gmail.com	Basabo	111222
12	Jimmy	Cashier	02-APR-20	Full Time	20000	jimmy@gmail.com	Khilgaon	111223
13	Chris	Cashier	07-APR-20	Part Time	10000	chris@gmail.com	Uttara	111224
14	John	Chef	02-APR-20	Full Time	24000	john@gmail.com	Gulshan	111225
15	Kevin	Chef	07-APR-20	Part Time	12000	kevin@gmail.com	Banani	111226
16	Nicholas	Waiter	02-APR-20	Full Time	16000	nicholas@gmail.com	Rampura	111227
17	Ben	Waiter	02-APR-20	Full Time	16000	ben@gmail.com	Badda	111228
18	Smith	Waiter	07-APR-20	Part Time	8000	smith@gmail.com	Bashundhara	111229
19	Lynch	Waiter	07-APR-20	Part Time	8000	lynch@gmail.com	Mirpur	111220

3. Supplies Table:



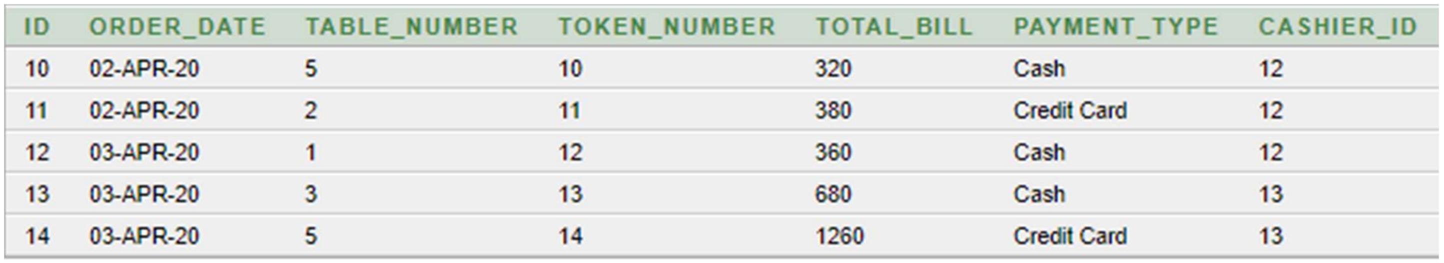
ID	NAME	REMAINING_QUANTITY
11	Normal Buns	250
12	Special Buns	180
13	Broich Buns	120
14	Cheese Slices	400
15	Beef Patties	150
16	Chicken Patties	150
17	Barbeque Beef Patties	60
18	Barbeque Chicken Patties	60
19	Beef Steak	40
20	Bacon Stripes	30

4. Food_Items_Burger Table:



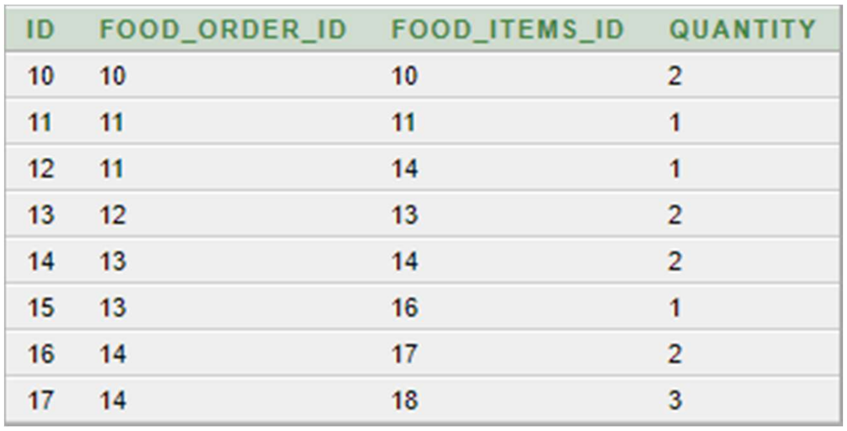
ID	NAME	BUNS	MEAT_PATTIES	CHEESE_SLICES	PRICE
10	Chicken Burger	Normal Buns	Chicken Patty	No	160
11	Beef Burger	Normal Buns	Beef Patty	No	160
12	Chicken Cheese Burger	Normal Buns	Chicken Patty	Yes	180
13	Beef Cheese Burger	Normal Buns	Beef Patty	Yes	180
14	Barbeque Chicken Burger	Special Buns	Barbeque Chicken Patty	No	220
15	Barbeque Beef Burger	Special Buns	Barbeque Beef Patty	No	220
16	Double Decker Beef Burger	Special Buns	Double Beef Patty	Yes	240
17	Chicken And Bacon Burger	Broich Buns	Chicken Patty and Bacon	Yes	240
18	Beef Steak Burger	Broich Buns	Beef Steak Patty	Yes	260

5. Food_Order Table:



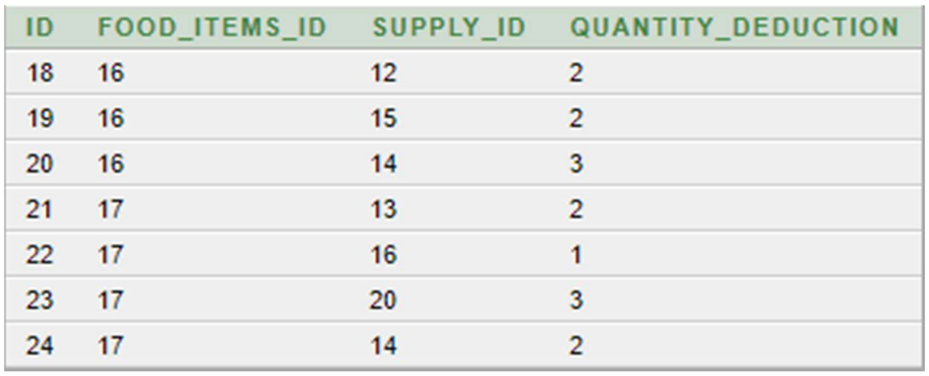
ID	ORDER_DATE	TABLE_NUMBER	TOKEN_NUMBER	TOTAL_BILL	PAYMENT_TYPE	CASHIER_ID
10	02-APR-20	5	10	320	Cash	12
11	02-APR-20	2	11	380	Credit Card	12
12	03-APR-20	1	12	360	Cash	12
13	03-APR-20	3	13	680	Cash	13
14	03-APR-20	5	14	1260	Credit Card	13

6. Ordered_Food_Items Table:



ID	FOOD_ORDER_ID	FOOD_ITEMS_ID	QUANTITY
10	10	10	2
11	11	11	1
12	11	14	1
13	12	13	2
14	13	14	2
15	13	16	1
16	14	17	2
17	14	18	3

7. Food_And_Supplies Table:



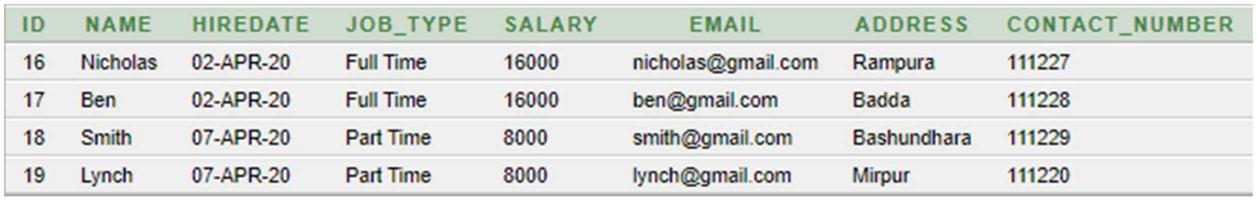
ID	FOOD_ITEMS_ID	SUPPLY_ID	QUANTITY_DEDUCTION
18	16	12	2
19	16	15	2
20	16	14	3
21	17	13	2
22	17	16	1
23	17	20	3
24	17	14	2

SQL

Views:

1. Create a view that contains all the details about the waiters of the burger shop.

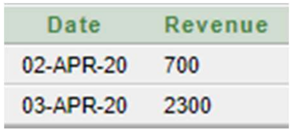
Ans. `create view waiters as select id, name, hiredate, job_type, salary, email, address, contact_number from employees where role='Waiter';`



ID	NAME	HIREDATE	JOB_TYPE	SALARY	EMAIL	ADDRESS	CONTACT_NUMBER
16	Nicholas	02-APR-20	Full Time	16000	nicholas@gmail.com	Rampura	111227
17	Ben	02-APR-20	Full Time	16000	ben@gmail.com	Badda	111228
18	Smith	07-APR-20	Part Time	8000	smith@gmail.com	Bashundhara	111229
19	Lynch	07-APR-20	Part Time	8000	lynch@gmail.com	Mirpur	111220

2. Create a view that shows the total revenue earned in each date in the burger shop.

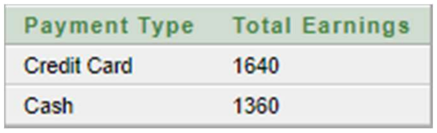
Ans. `create view revenue_per_day as select order_date as "Date", sum(total_bill) as "Revenue" from food_order group by order_date;`



Date	Revenue
02-APR-20	700
03-APR-20	2300

3. Create a view that shows the total amount of customer payments received through each of the payment types in the burger shop.

Ans. `create view earnings as select payment_type as "Payment Type", sum(total_bill) as "Total Earnings" from food_order group by payment_type;`



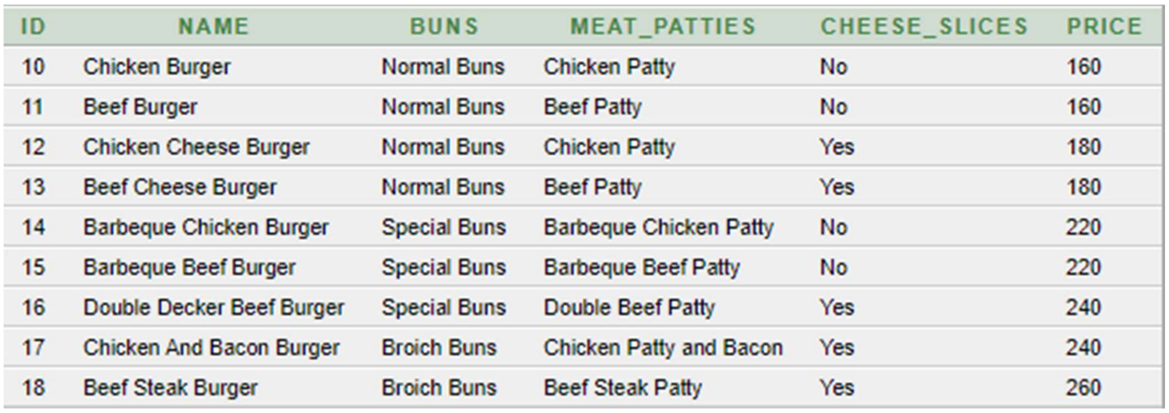
Payment Type	Total Earnings
Credit Card	1640
Cash	1360

Synonyms:

1. create a synonym for the table food_items_burger.

Ans. `create synonym burgers for food_items_burger;`

```
select * from burgers;
```

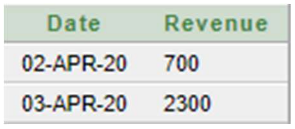


ID	NAME	BUNS	MEAT_PATTIES	CHEESE_SLICES	PRICE
10	Chicken Burger	Normal Buns	Chicken Patty	No	160
11	Beef Burger	Normal Buns	Beef Patty	No	160
12	Chicken Cheese Burger	Normal Buns	Chicken Patty	Yes	180
13	Beef Cheese Burger	Normal Buns	Beef Patty	Yes	180
14	Barbeque Chicken Burger	Special Buns	Barbeque Chicken Patty	No	220
15	Barbeque Beef Burger	Special Buns	Barbeque Beef Patty	No	220
16	Double Decker Beef Burger	Special Buns	Double Beef Patty	Yes	240
17	Chicken And Bacon Burger	Broich Buns	Chicken Patty and Bacon	Yes	240
18	Beef Steak Burger	Broich Buns	Beef Steak Patty	Yes	260

2. create a synonym for the view revenue_per_day.

Ans. `create synonym revenue for revenue_per_day;`

```
select * from revenue;
```

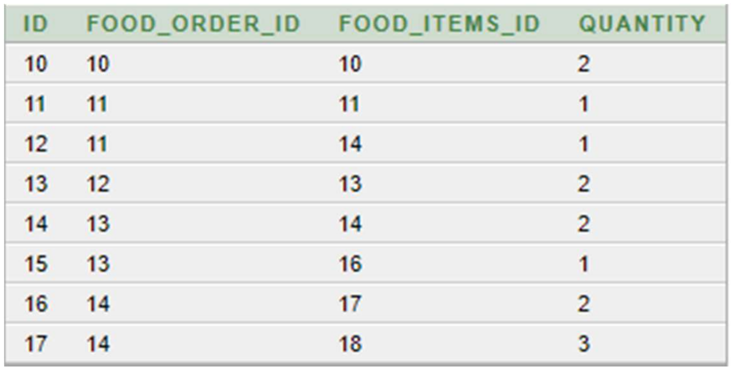


Date	Revenue
02-APR-20	700
03-APR-20	2300

3. create a synonym for the table ordered_food_items.

Ans. `create synonym order_list for ordered_food_items;`

```
select * from order_list;
```

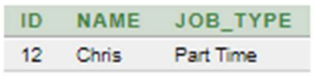


ID	FOOD_ORDER_ID	FOOD_ITEMS_ID	QUANTITY
10	10	10	2
11	11	11	1
12	11	14	1
13	12	13	2
14	13	14	2
15	13	16	1
16	14	17	2
17	14	18	3

Single Row Functions:

1. Display the employee number, name, and job-type for employee Chris.

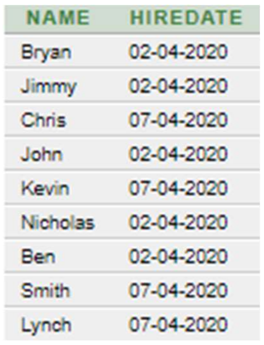
Ans. `Select id, name, job_type  from employees where name = 'chris';`



ID	NAME	JOB_TYPE
12	Chris	Part Time

2. Display the hiredate of the employees in formate "DD-MM-YYYY" along with their name.

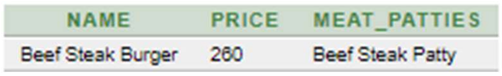
Ans. `Select name, TO_CHAR(hiredate, 'DD-MM-YYYY') hiredate from employees;`



NAME	HIREDATE
Bryan	02-04-2020
Jimmy	02-04-2020
Chris	07-04-2020
John	02-04-2020
Kevin	07-04-2020
Nicholas	02-04-2020
Ben	02-04-2020
Smith	07-04-2020
Lynch	07-04-2020

3. Display the name and price of the food items where beef steak bun is being used.

Ans.
```
select name, price, meat_patties from food_items_burger where
meat_patties='Beef Steak Patty';
```

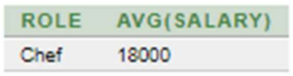


NAME	PRICE	MEAT_PATTIES
Beef Steak Burger	260	Beef Steak Patty

Group Functions:

1. Display the average salary of chef;

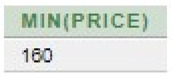
Ans.
```
Select role, avg(salary) from employees where role='Chef' group by
role;
```



ROLE	AVG(SALARY)
Chef	18000

2. Display the min priced burger available in the shop.

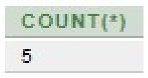
Ans. `select min(price) from food_items_burger;`



MIN(PRICE)
160

3. Display the amount of burgers where cheese slices are used.

Ans. `select  count(*) from food_items_burger where cheese_slices='Yes';`

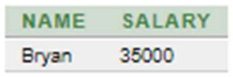


COUNT(*)
5

Sub-Query:

1. Display the employee names that earn a salary that is higher than the salary of all Chef.

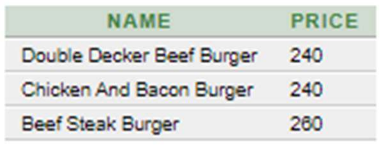
Ans.
```
select name, salary from employees where salary > ALL (select salary
from employees where role='Chef');
```



NAME	SALARY
Bryan	35000

2. Display the burger name that has the higher price than any other burger that does not have cheese.

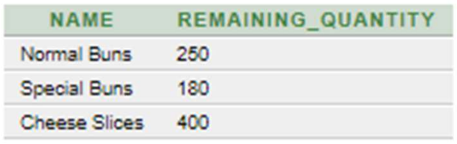
Ans.
```
select name, price from food_items_burger where price > ALL(select
price from food_items_burger where cheese_slices='No');
```



NAME	PRICE
Double Decker Beef Burger	240
Chicken And Bacon Burger	240
Beef Steak Burger	260

3. Display the items from supplies that are left more than 150 pieces.

Ans.
```
select name, remaining_quantity from supplies where remaining_quantity
> ALL (select remaining_quantity from supplies where
remaining_quantity='150');
```

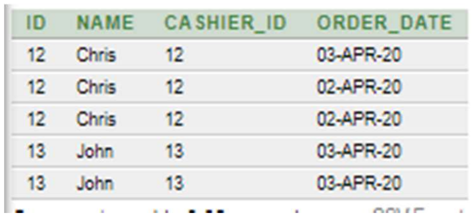


NAME	REMAINING_QUANTITY
Normal Buns	250
Special Buns	180
Cheese Slices	400

Joining:

1. Display the order dates of the orders taken by different Cashier .

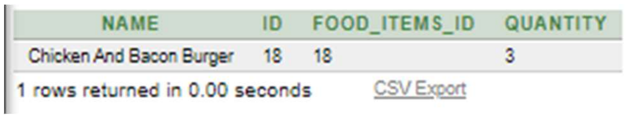
Ans. `select e.id, e.name, o.cashier_id, o.order_date from employees e, food_order o where e.id=o.cashier_id;`



ID	NAME	CASHIER_ID	ORDER_DATE
12	Chris	12	03-APR-20
12	Chris	12	02-APR-20
12	Chris	12	02-APR-20
13	John	13	03-APR-20
13	John	13	03-APR-20

2. Display the name of the burger that has been ordered more than twice in an order.

Ans. `select b.name, b.id, oi.food_items_id, oi. quantity from food_items_burger b, ordered_food_items oi where b.id=oi.food_items_id AND oi.quantity>'2';`

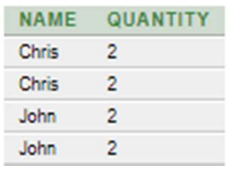


NAME	ID	FOOD_ITEMS_ID	QUANTITY
Chicken And Bacon Burger	18	18	3

1 rows returned in 0.00 seconds CSV Export

3. Display the name of the Cashiers who received orders containing 2 items.

Ans. `select  e.name, oi.quantity from  employees e,food_order o, ordered_food_items oi where e.id=o.cashier_id AND o.id=oi.food_order_id AND oi.quantity='2';`



NAME	QUANTITY
Chris	2
Chris	2
John	2
John	2

PL/SQL

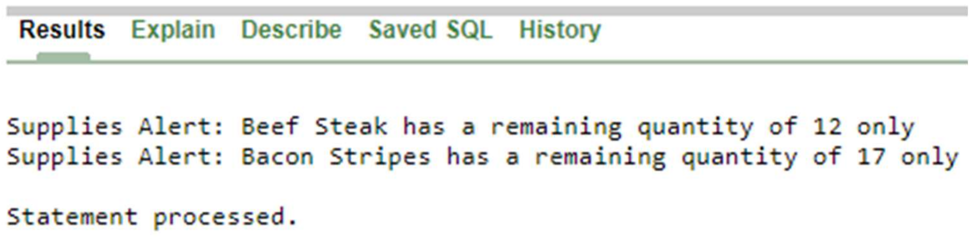
Procedure:

1. Create a procedure that displays the supplies that have remaining quantity less than 20.

Ans.

```
create or replace procedure check_supplies_count
as
    cursor supplies_cursor is select * from supplies;
    supplies_record supplies%rowtype;
    var_index number(10);
    var_size number(10);
begin
    var_index:=0;
    select count(*) into var_size from supplies;
    open supplies_cursor;
    while (var_index < var_size) loop
        fetch supplies_cursor into supplies_record;
        if(supplies_record.remaining_quantity < 20) then
            dbms_output.put_line('Supplies Alert: ' || supplies_record.name
|| ' has a remaining quantity of ' || supplies_record.remaining_quantity || '
only');
        end if;
        var_index:=var_index + 1;
    end loop;
    close supplies_cursor;
end;
```

```
begin

    check_supplies_count();

end;
```

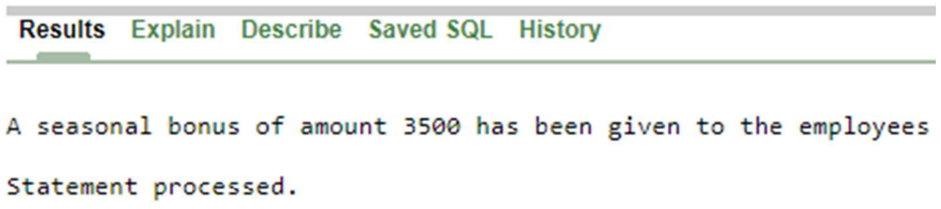


```
Supplies Alert: Beef Steak has a remaining quantity of 12 only
Supplies Alert: Bacon Stripes has a remaining quantity of 17 only

Statement processed.
```

2. Create a procedure though which a seasonal bonus can added to the salary of all the employees.

Ans.

```
create or replace procedure add_seasonal_bonus(par_bonus_amount in number)

as

begin

    update employees set salary=salary + par_bonus_amount;

    dbms_output.put_line('A seasonal bonus of amount ' || par_bonus_amount ||
' has been given to the employees');

end;


declare

    var_bonus_amount number(10);

begin

    var_bonus_amount:=3500;

    add_seasonal_bonus(var_bonus_amount);

end;
```



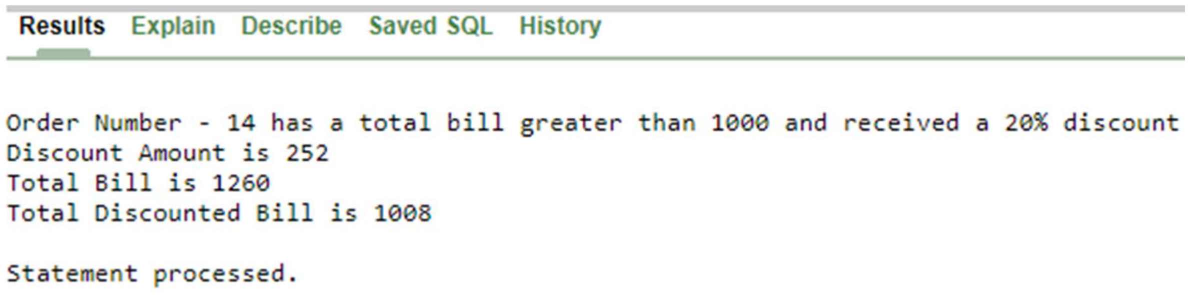
```
A seasonal bonus of amount 3500 has been given to the employees

Statement processed.
```

3. Create a procedure through which a discount (20%) can be added to the food orders which have a total bill of more than 1000.

Ans.

```
create or replace procedure customer_discount(par_order_id in number)

as

    var_total_bill number(10);

    var_discount_percentage number(10, 2);

    var_discount_amount number(10);

    var_discounted_bill number(10);

begin

    var_discount_percentage:=0.2;

    select total_bill into var_total_bill from food_order where
id=par_order_id;

    if(var_total_bill > 1000) then

        var_discount_amount:=var_total_bill * var_discount_percentage;

        var_discounted_bill:=var_total_bill - var_discount_amount;

        dbms_output.put_line('Order Number - ' || par_order_id || ' has a
total bill greater than 1000 and received a 20% discount');

        dbms_output.put_line('Discount Amount is ' || var_discount_amount);

        dbms_output.put_line('Total Bill is ' || var_total_bill);

        dbms_output.put_line('Total Discounted Bill is ' ||
var_discounted_bill);

    else

        dbms_output.put_line('Order Number - ' || par_order_id || ' has no
discount since the total bill is less than 1000');

    end if;

end;


declare
```

```
    var_order_id number(10);
begin
    var_order_id:=14;
    customer_discount(var_order_id);
end;
```



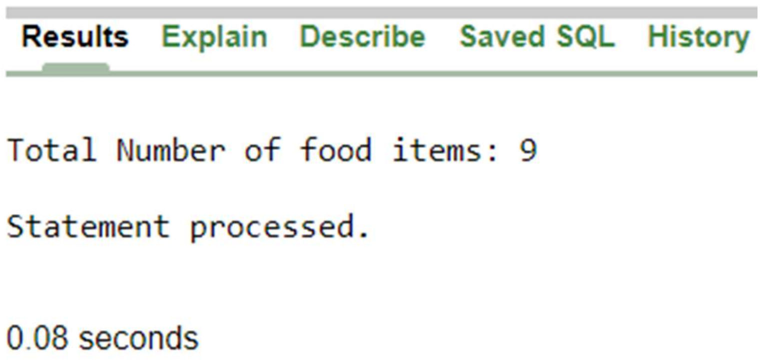
```
Order Number - 14 has a total bill greater than 1000 and received a 20% discount
Discount Amount is 252
Total Bill is 1260
Total Discounted Bill is 1008

Statement processed.
```

Function:

1. Create a function that returns the total number of food items stored inside the burger shop.

Ans:
```
CREATE OR REPLACE FUNCTION totalfooditems
RETURN NUMBER IS
total number(12):=0;
BEGIN
   SELECT count(*) into total FROM Food_Items_Burger;
   RETURN total;
END;
/
DECLARE
   c number(12);
BEGIN
   c := totalfooditems();
```

```
    dbms_output.put_line('Total Number of food items: ' || c);
END;
```

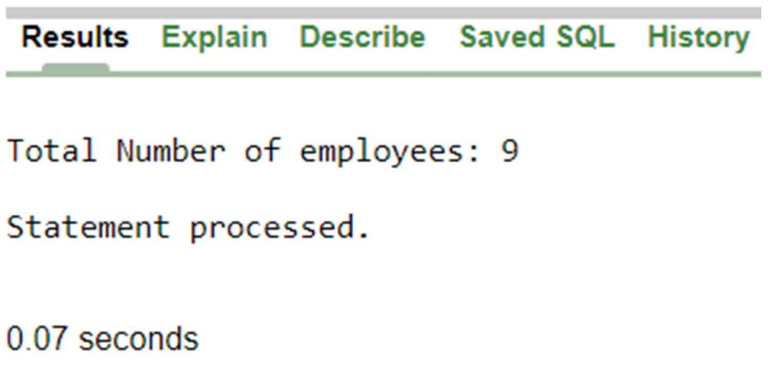


```
Total Number of food items: 9

Statement processed.

0.08 seconds
```

2. Create a function that returns the total number of employees stored inside the burger shop.

Ans: `CREATE OR REPLACE FUNCTION totalemployees`

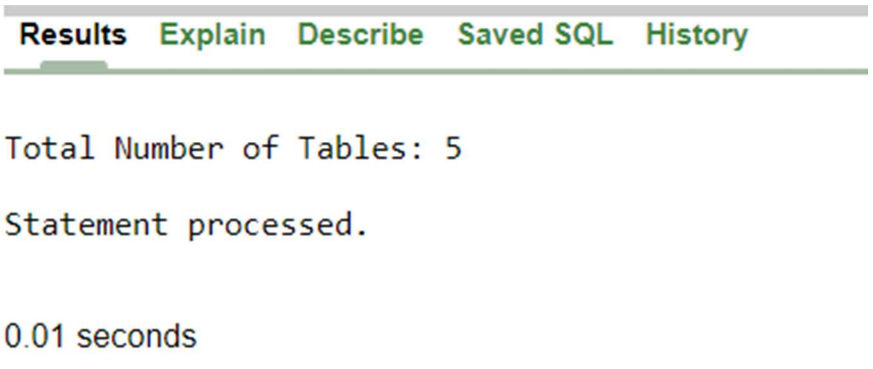
```
RETURN NUMBER IS
total number(12):=0;
BEGIN
    SELECT count(*) into total FROM Employees;
    RETURN total;
END;
/
DECLARE
    c number(12);
BEGIN
    c := totalemployees();
    dbms_output.put_line('Total Number of employees: ' || c);
END;
```



```
Total Number of employees: 9

Statement processed.

0.07 seconds
```

3. Create a function that returns the total number of tables stored inside the burger shop.

Ans:
```
CREATE OR REPLACE FUNCTION totalTable
RETURN NUMBER IS
total number(12):=0;
BEGIN
   SELECT count(*) into total FROM Food_Order;
   RETURN total;
END;
/
DECLARE
   c number(12);
BEGIN
   c := totalTable();
   dbms_output.put_line('Total Number of Tables: ' || c);
END;
```



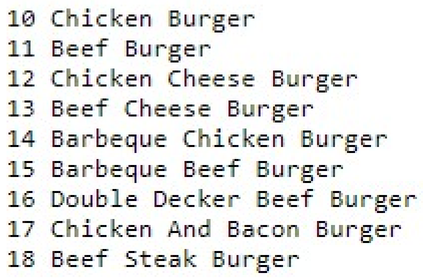
```
Total Number of Tables: 5

Statement processed.


0.01 seconds
```

Cursor:

1. Create a cursor that can output the id and name of all the burgers item that are sorted in Food_Items_Burgers table.

Ans:

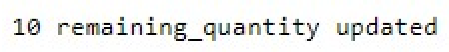
```
declare
      b_id food_items_burger.id%type;
 b_name food_items_burger.name%type;
 i number(2);
 cursor f_food_items_burger is
 select id, name from food_items_burger;
 begin
 open f_food_items_burger;
 FOR i in 1.. 9 LOOP
 fetch  f_food_items_burger into b_id, b_name;
 dbms_output.put_line(b_id||' '||b_name);
 end loop;
 close f_food_items_burger;
 end;
```



```
10 Chicken Burger
11 Beef Burger
12 Chicken Cheese Burger
13 Beef Cheese Burger
14 Barbeque Chicken Burger
15 Barbeque Beef Burger
16 Double Decker Beef Burger
17 Chicken And Bacon Burger
18 Beef Steak Burger
```

2. Create an implicit cursor that can display the total number of rows updated in the supplies table.

Ans:
```
Declare
        total_rows number(2);
        begin
        update supplies
        set remaining_quantity = remaining_quantity + 5;
      if sql%notfound then
      dbms_output.put_line('remaining_quantity not updated');
        elseif sql%found then
        total_rows := sql%rowcount;
     dbms_output.put_line( total_rows || ' remaining_quantity updated
');
         end if;
end;
```

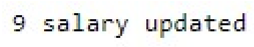


```
10 remaining_quantity updated
```

3. Create a cursor that can display the total number of rows updated in the Employees table.

Ans:
```
Declare
total_rows number(2);
begin
update employees
set salary = salary + 500;
if sql%notfound then
dbms_output.put_line('no salary updated');
 elseif sql%found then
total_rows := sql%rowcount;
dbms_output.put_line( total_rows || ' salary updated ');
```

```
 end if;
end;
```

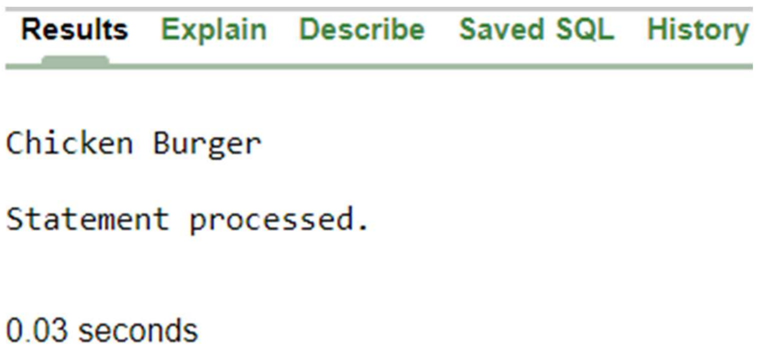


```
9 salary updated
```

Record:

1. Create a record that can output the name of the food whose id is 10.

Ans:
```
declare
food_rec Food_Items_Burger%rowtype;
begin
select * into food_rec from Food_Items_Burger
where ID='10';
dbms_output.put_line(food_rec.NAME);
end;
```

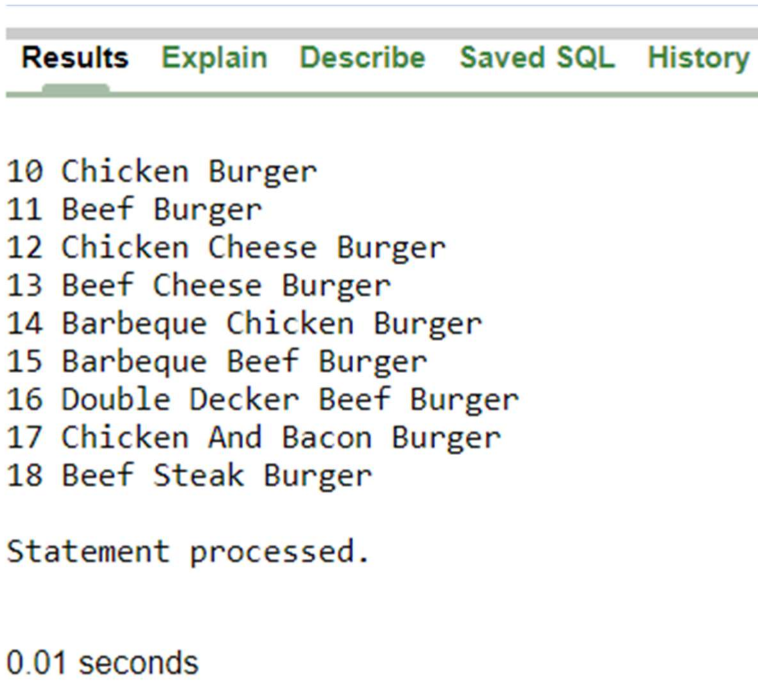


```
Chicken Burger

Statement processed.

0.03 seconds
```

2. Create a record that can output the name of all the foods and id inside the burger shop.

Ans:
```
declare
food_rec Food_Items_Burger%rowtype;
begin
for food_rec
in(select * from Food_Items_Burger)
```

```
loop

dbms_output.put_line(food_rec.ID||' '||food_rec.NAME);

end loop;

end;
```



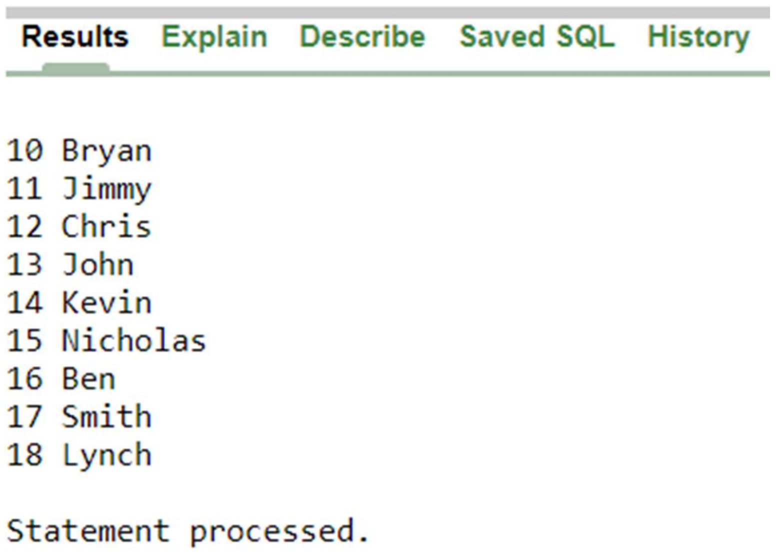
```
10 Chicken Burger
11 Beef Burger
12 Chicken Cheese Burger
13 Beef Cheese Burger
14 Barbeque Chicken Burger
15 Barbeque Beef Burger
16 Double Decker Beef Burger
17 Chicken And Bacon Burger
18 Beef Steak Burger

Statement processed.


0.01 seconds
```

3. Create a record that can output the name of all the employees and their id inside the burger shop .

Ans: declare

```
employees_rec Employees%rowtype;

begin

for employees_rec

in(select * from Employees)

loop

dbms_output.put_line(employees_rec.ID||' '||employees_rec.NAME);
```

```
end loop;

end;
```



```
10 Bryan
11 Jimmy
12 Chris
13 John
14 Kevin
15 Nicholas
16 Ben
17 Smith
18 Lynch

Statement processed.
```

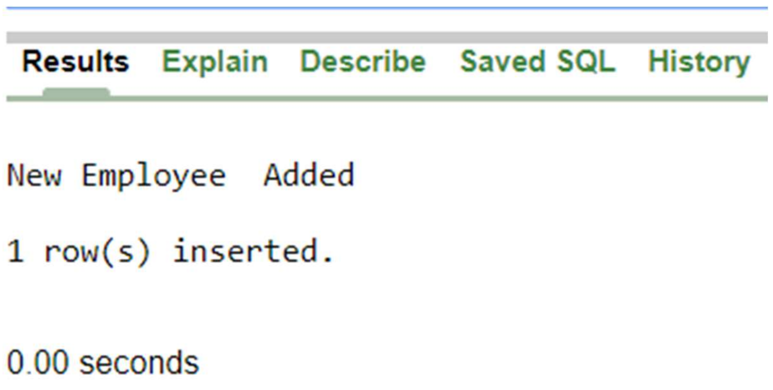
Trigger:

1. Create a trigger in such a way that whenever a new row is inserted into the employee table an output ‘New Employee Added’ is generated.

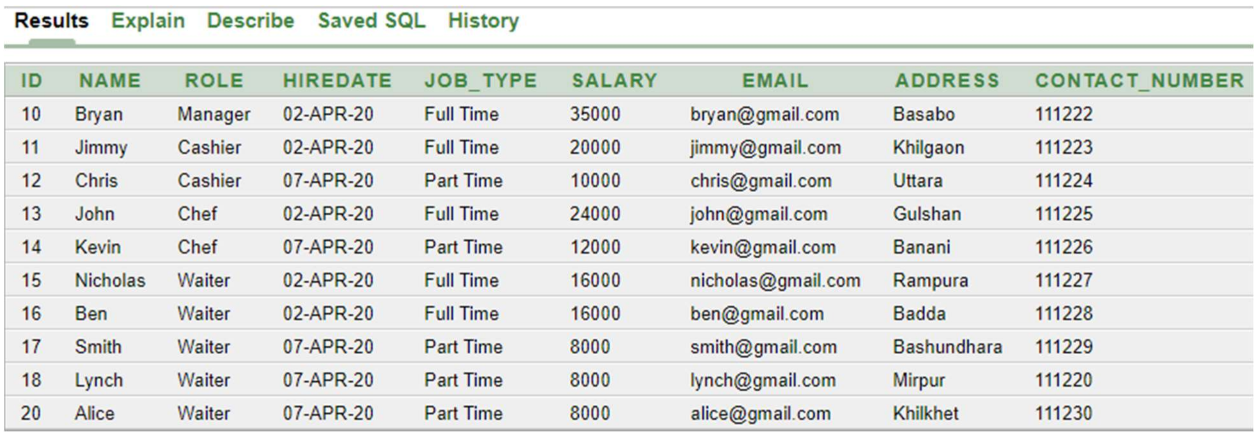
Ans:

```
CREATE OR REPLACE TRIGGER employee_added

after INSERT ON Employees

FOR EACH ROW

BEGIN

   dbms_output.put_line('New Employee  Added');

END;

/

select * from Employees;
```

```
insert into Employees values ('20','Alice','Waiter','07-Apr-20', 'Part
Time', 8000, 'alice@gmail.com', 'Khilkhet', 111230);

rollback
```



```
New Employee  Added

1 row(s) inserted.


0.00 seconds
```



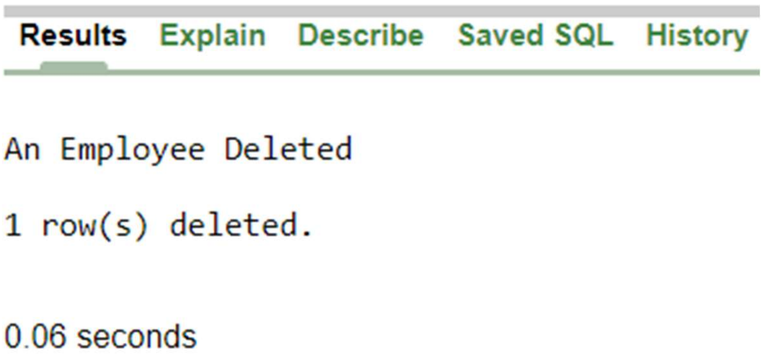
ID	NAME	ROLE	HIREDATE	JOB_TYPE	SALARY	EMAIL	ADDRESS	CONTACT_NUMBER
10	Bryan	Manager	02-APR-20	Full Time	35000	bryan@gmail.com	Basabo	111222
11	Jimmy	Cashier	02-APR-20	Full Time	20000	jimmy@gmail.com	Khilgaon	111223
12	Chris	Cashier	07-APR-20	Part Time	10000	chris@gmail.com	Uttara	111224
13	John	Chef	02-APR-20	Full Time	24000	john@gmail.com	Gulshan	111225
14	Kevin	Chef	07-APR-20	Part Time	12000	kevin@gmail.com	Banani	111226
15	Nicholas	Waiter	02-APR-20	Full Time	16000	nicholas@gmail.com	Rampura	111227
16	Ben	Waiter	02-APR-20	Full Time	16000	ben@gmail.com	Badda	111228
17	Smith	Waiter	07-APR-20	Part Time	8000	smith@gmail.com	Bashundhara	111229
18	Lynch	Waiter	07-APR-20	Part Time	8000	lynch@gmail.com	Mirpur	111220
20	Alice	Waiter	07-APR-20	Part Time	8000	alice@gmail.com	Khilkhet	111230

2. Create a trigger in such a way that whenever a row is deleted from the employee table an output ‘An Employee Deleted’ is generated.

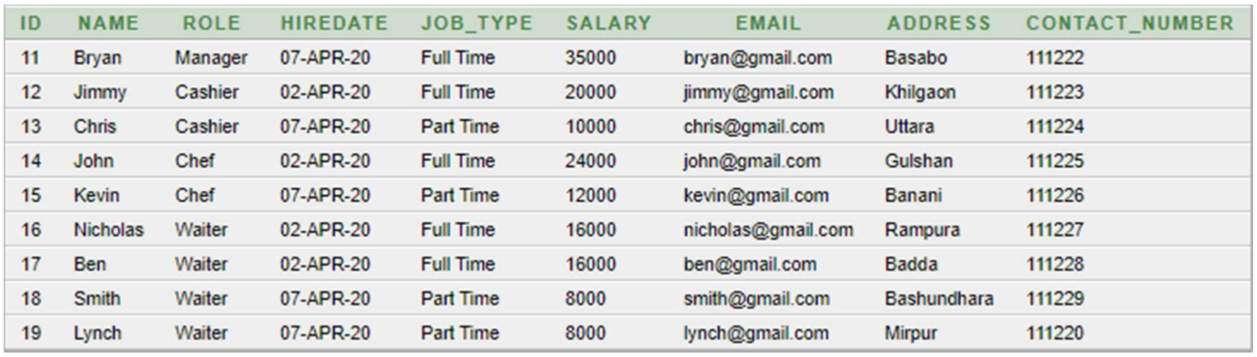
Ans:

```
CREATE OR REPLACE TRIGGER employee_deleted
after DELETE ON Employees
FOR EACH ROW
BEGIN
   dbms_output.put_line('An Employee Deleted ');
END;
/
select * from Employees;
DELETE FROM Employees
```

```
WHERE ID = 20;
```



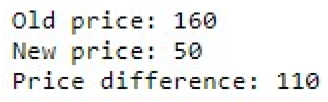
```
An Employee Deleted

1 row(s) deleted.

0.06 seconds
```



ID	NAME	ROLE	HIREDATE	JOB_TYPE	SALARY	EMAIL	ADDRESS	CONTACT_NUMBER
11	Bryan	Manager	07-APR-20	Full Time	35000	bryan@gmail.com	Basabo	111222
12	Jimmy	Cashier	02-APR-20	Full Time	20000	jimmy@gmail.com	Khilgaon	111223
13	Chris	Cashier	07-APR-20	Part Time	10000	chris@gmail.com	Uttara	111224
14	John	Chef	02-APR-20	Full Time	24000	john@gmail.com	Gulshan	111225
15	Kevin	Chef	07-APR-20	Part Time	12000	kevin@gmail.com	Banani	111226
16	Nicholas	Waiter	02-APR-20	Full Time	16000	nicholas@gmail.com	Rampura	111227
17	Ben	Waiter	02-APR-20	Full Time	16000	ben@gmail.com	Badda	111228
18	Smith	Waiter	07-APR-20	Part Time	8000	smith@gmail.com	Bashundhara	111229
19	Lynch	Waiter	07-APR-20	Part Time	8000	lynch@gmail.com	Mirpur	111220

3. Create a trigger which can display the old price, new price as well as price difference of any burger in the food_items_burger table.

Ans.
```
create or replace trigger display_price_changes
before delete or insert or update on food_items_burger
for each row
when (new.id > 0)
declare
   price_diff number;
begin
   price_diff := :old.price  - :new.price;
   dbms_output.put_line('Old price: ' || :Old.price);
   dbms_output.put_line('New price: ' || :NEW.price);
   dbms_output.put_line('Price difference: ' || price_diff);
end;
update food_items_burger set price='50' where id=12;
```



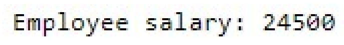
```
Old price: 160
New price: 50
Price difference: 110
```

Package:

1. Create a package that contains a procedure which can display the salary of any employee whose name is passed as its parameter.

Ans: `create or replace package emp_pack as`

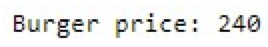
```
   procedure display_empsalary(e__name employees.name%type);
    end emp_pack;
   create or replace package body emp_pack as
  procedure display_empsalary(e__name employees.name%type) is
   e_salary employees.salary%TYPE;
begin
      select salary into e_salary
      from employees
      where  name=e__name;
      dbms_output.put_line('Employee salary: '|| e_salary);
   END display_empsalary;
END emp_pack;
begin
emp_pack.display_empsalary('John');
end;
```



```
Employee salary: 24500
```

2. Create a package that contains a procedure which can display the price of any burger whose name is passed as its parameter.

Ans:
```
create or replace package burger_pack as
  procedure display_price(b__name food_items_burger.name%type);
   end burger_pack;
  create or replace package body burger_pack as
  procedure display_price(b__name food_items_burger.name%type) is
  b_price food_items_burger.price%TYPE;
 begin
     select price into b_price
     from   food_items_burger
     where name=b__name;
     dbms_output.put_line('Burger price: '|| b_price);
  end display_price;
end burger_pack;
begin
burger_pack.display_price('Chicken And Bacon Burger');
end;
```

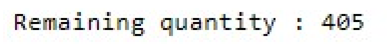


```
Burger price: 240
```

3. Create a package that contains a procedure which can display the remaining quantity of any supplies whose name is passed as its parameter.

Ans:
```
create or replace package supplies_pack as
procedure display_remaining_quantity(s__name supplies.name%type);
end supplies_pack;
create or replace package body supplies_pack as
 procedure display_remaining_quantity(s__name supplies.name%type) is
 s_remaining_quantity supplies.remaining_quantity%TYPE;
 begin
  select remaining_quantity into s_remaining_quantity
    from supplies
    where  name=s__name;
```

```
     dbms_output.put_line('Remaining quantity : '|| s_remaining_quantity);
   end display_remaining_quantity;
end supplies_pack;
begin
supplies_pack.display_remaining_quantity('Cheese Slices');
end;
```



```
Remaining quantity : 405
```

Conclusion

Our project is about a Burger Shop Management System. We think our project will be beneficial for maintaining a Burger Shop. The project is now complete but still there are scopes for adding some extra features. Here we have used oracle database. It is necessary to mention that while developing the application, it was necessary to make some changes in the database tables. For implementation purpose, some attributes of some tables had to be tuned and changed. We have included the database tables in the document which we have implemented in our application. We have also included the data insertion queries for demonstrating how the database tables of the application store the necessary information. This application can be used to take food orders from the customers in a very easy way. It is expected that the application will be able to serve the purpose of maintaining a burger shop in an efficient manner.