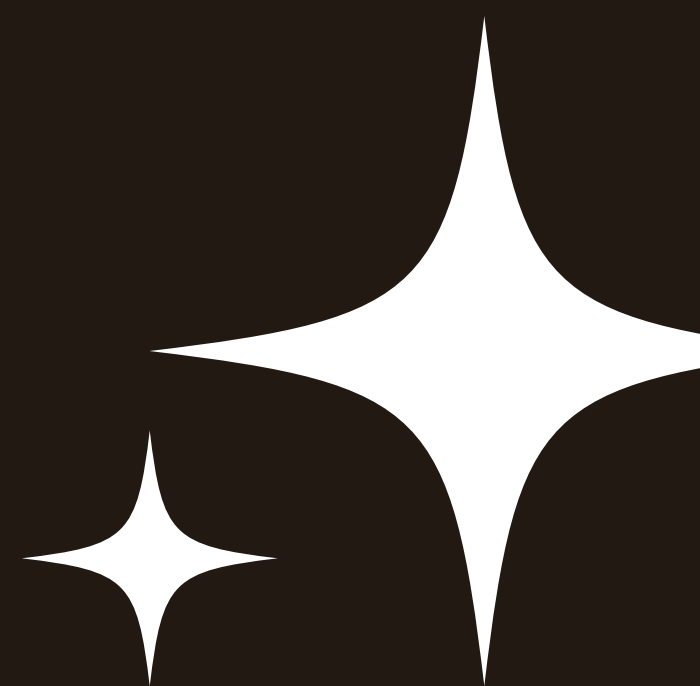


Cloud Burger Restaurant



Introduction

Our restaurant is a fast food restaurant specialized in making delicious burgers. We have two branches and we also have a home delivery service, so that our customers can save their time. They just open our website or application and choose what they want from our delicious types of burgers as they can choose their favorite side item and so on. The best thing is that you can choose the exact time for delivery, and you will receive food as quickly as possible and at the exact time. But unfortunately, we have some hidden limitations such as, very difficult access to information, separation & isolation of data. So, we will develop our restaurant to the best version we can through our written objectives and using the requirements.

background

In the past our restaurant had an all paper base system, which was great until we became more popular and busier. The process of executing orders became difficult in many respects, such as the difficulty of the waiter communicating with the chef to implement the orders, and the error rate is high, in addition to managing the restaurant and manipulating paper reports is so hard. As it was difficult to keep track of our budget, supplies, and customer problems. what was happening in our restaurant based on paper system, lead to a bad customer experience.

Problem

Our problem in our restaurant was using a paper base system (manual method). It was occupying a very large place, tiring, complex and inaccurate, full of errors and repeated data, and it was also difficult to access data when needed. Also, there is isolation & separation of data and data dependence.

Objectives

Aims to reduce errors, quick access to information and prevent data duplication. And one of the most important goals of this project is to create subprograms to make maintenance easier, improve data security and integrity, and improve performance. Such as, Create function to return list of customer bills by accepting customerID(SSN) , Create function to return list of all orders that has been delivered for each driver by taking LicenseID to compare them, Create procedure to update notes to supplier in 'supply' table in case of any new notes or changes, Create procedure to delete customerID(SSN) from 'making' table in case order is cancelled. Also, Merchant should receive an automatic message when the supply amount in 'supply' table become less than half (using triggers), And so on.

Requirement

Since our restaurant decided to convert the system(database) from paper to electronic, we decided to work using language (PL/SQL) (and create an account in Oracle APEX), and we also needed staff devices (iPads to take order from customers and send it automatically to chef) and staff who know how to deal with our program. So, we created courses for our employees to have sufficient experience to deal with our new database. And we signed contracts for the latest equipment for our restaurant. All this for a better customer experience.

Distribution of duties

Customer: should be able to browse through the menu and look at various food options available in the restaurant along with the price for each item

The customer table contains customerId, custmerName, phone and address

Chef: oversees the purchase of food goods, interacts with customers, and ensures that they enjoy the food.

The chef table contains id ,name, specialty and salary

Waiter: Serves customers in cafes and restaurants

The waiter table contains the name and id

Supplier: should be able to view the inventory items so as to supply the lacking ingredients to the chef

The supplier table contains (supplierName, type , price and amount)

Bill: A document describing the products, quantities, and costs of the services the seller will give to the buyer

The bill table contains billNumber , products , quantities, cost and total

Summary

To sum up, our restaurant was using a paper base system to access, enter and view data. It was great until we became more famous, crowded, and open our second branch. then we lost control of the paper system. So, we decided to develop our restaurant to grow functionally, financially, technically and to keep track of budget, employees, and customer satisfaction in both of our branches. Also, to make our work environment and our employees satisfied and that will reflect on how they treat customers and attract customers and gain more money and increase income.

list of entities (tables) that have been identified to solve the identified problem:

Bill table: Contains order information such as order name, quantity, order date, and price . We use this table to document customers orders.

Customer table: Contains important customer information such as name , address and the purpose of this table is to know customer information.

Floor table: Contains information about the floor in general, the purpose of this table is to know the number of tables on every floor.

Delivery table: Contains driver information, name , license ID and ride type. The purpose of this table is to keep an eye on drivers.

Merchant table: Contains seller information and phone number to contact them.

Supply table: The purpose of this table is to know the amount and price of each type and if there are any notes from the seller.

Chef table: The purpose of this table is implementing the customer's request through the bill sent to chef.

Waiter table: taking orders from clients, giving them to the kitchen staff, and serving food to clients. This table's function is to serve as a conduit for client communications with the kitchen staff.

Meal table: Determine the meal and the favorite side item and determine calories for each meal.

Order table: This table tells about the Order ID, gives details about when order confirmed along with the status of the order.

Takeaway table: Contains the address, order number and delivery number. The purpose of this table is that the customer can order his food and we'll take it to him wherever he wants.

Cooking table: It specify the chef who cooks the meal for me, and I specify the meal.

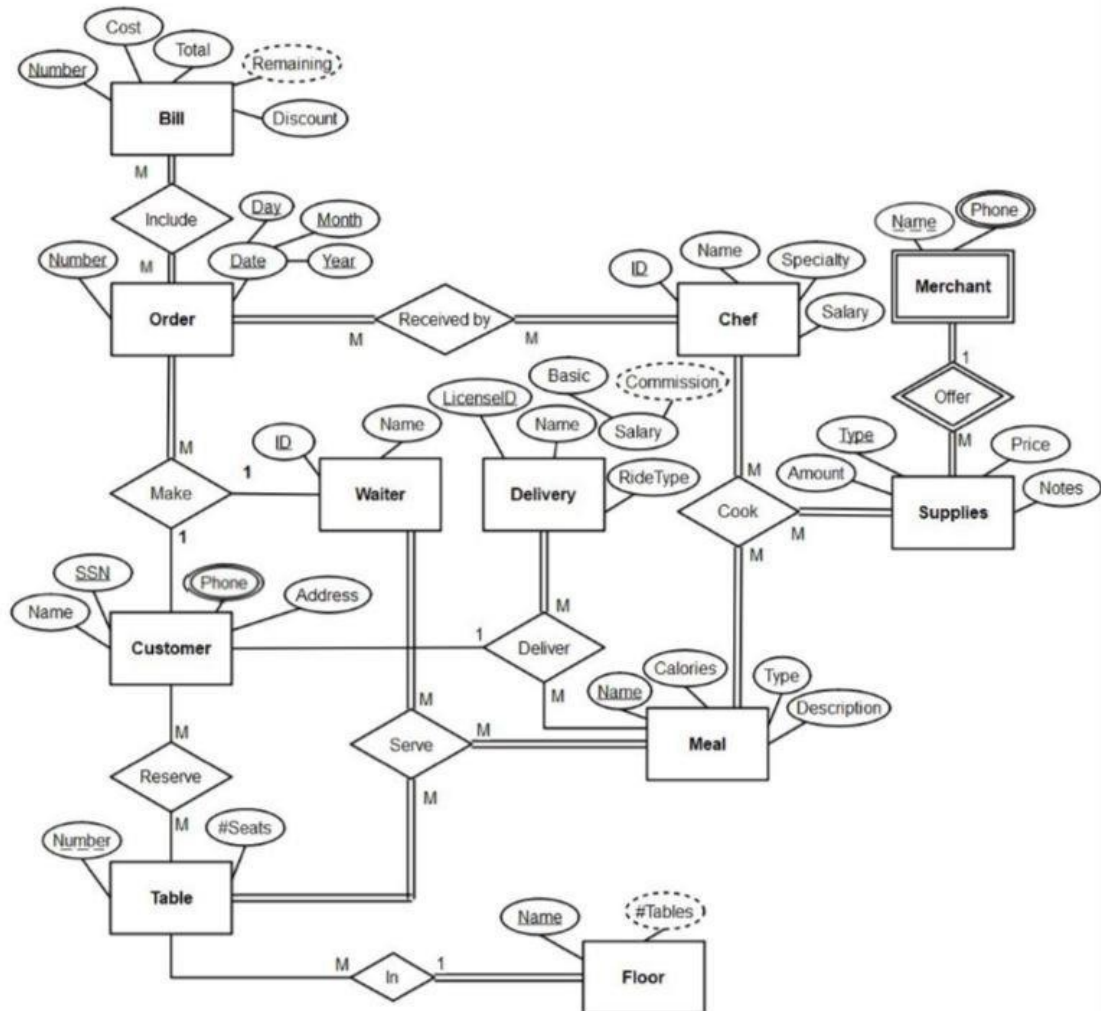
Making table: This table in a restaurant database is used to store information about orders placed by customers. It contains the customer's Ssn, the order number, the day, month and year of the order, and the waiter ID who took the order. It allows easy retrieval of data related to orders placed by customers.

Receiving table: This table in a restaurant database is used to store information about the orders that have been received by the restaurant. It stores the order number, the day, month and year of the order, and the chef's ID who received it.. This ensures that all data in this table is valid and up-to-date with other tables in the database.

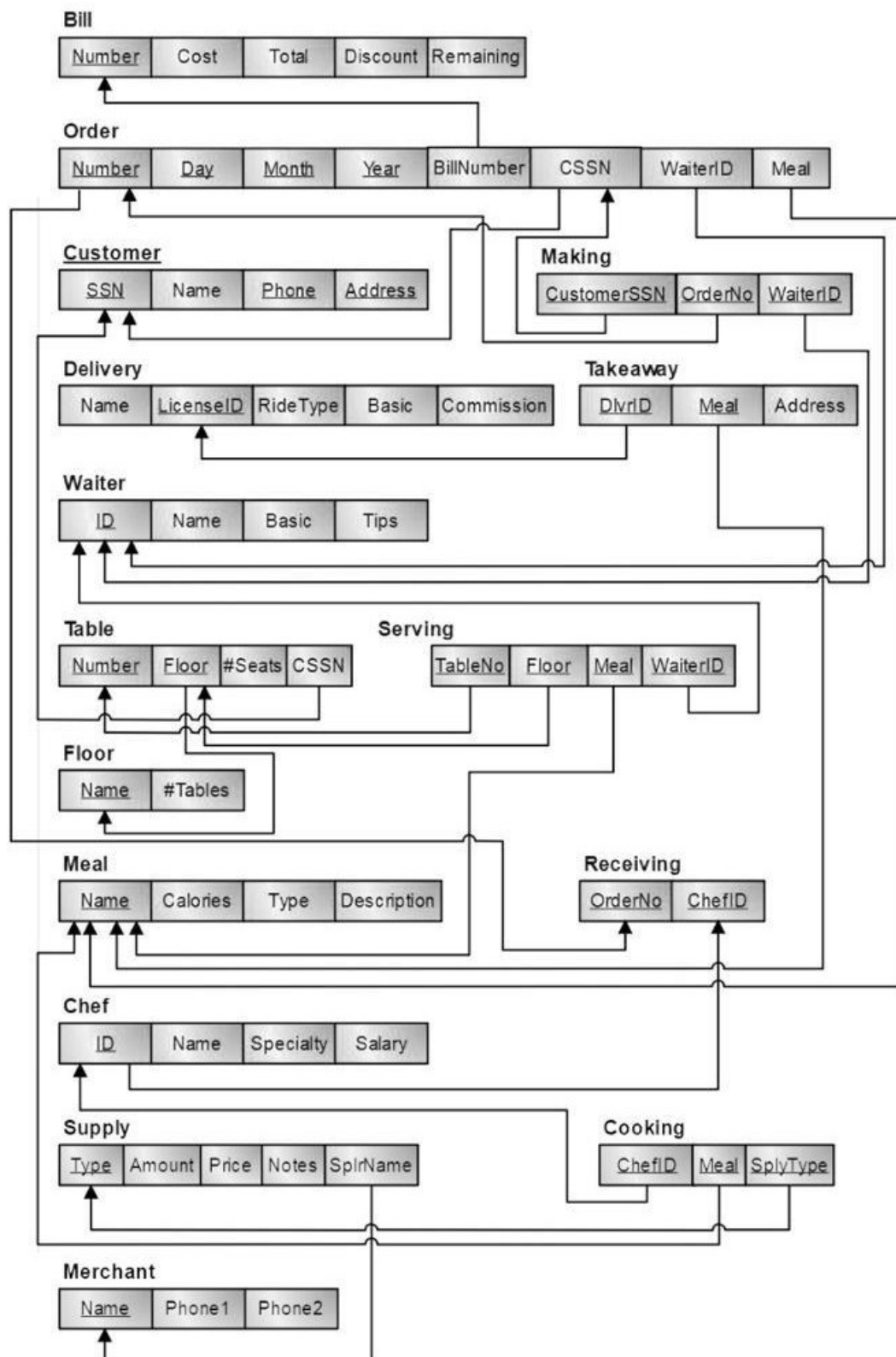
Serving table: Used to store information about the meals that are being served at each table. It stores the table number, floor, meal name, and waiter ID for each meal. This table helps to keep track of which meals are being served at which tables and by which waiters.

Tables table: It generally contains all the necessary information for a table, the table number, the number of chairs it has and its location (floor).

The conceptual data model (ERM)



The Logical data model (Database Schema)



Code :

Bill:

create table Bill

(BillNumber number(8) primary key not null,

BillCost float not null,

Total float not null,

Discount float,

Remaining float not null)

```
create table Bill
(BillNumber number(8) primary key not null,
BillCost float not null,
Total float not null,
Discount float,
Remaining float not null)
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Table created.

0.16 seconds

Customer:

create table Customer

(SSN number(8) PRIMARY KEY NOT NULL,

Name varchar(50) NOT NULL,

Phone varchar(10) unique,

Address varchar(100))

```
create table Customer
(SSN number(8) PRIMARY KEY NOT NULL,
Name varchar(50) NOT NULL,
Phone varchar(10) unique,
Address varchar(100))
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Table created.

0.16 seconds

Floor:

create table Floor
(Name varchar(25) PRIMARY KEY NOT NULL,
NoTables int not null)

```
create table Floor
(Name varchar(25) PRIMARY KEY NOT NULL,
NoTables int not null)
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Table created.

0.13 seconds

Tables1:

create table Tables1
(TNumber Number(3) NOT NULL,
Floor varchar(25) NOT NULL,
NoSeats number(2),
CSSN number(8),
primary key (TNumber),
CONSTRAINT FK_CSSN FOREIGN KEY (CSSN) REFERENCES CUSTOMER(SSN),
CONSTRAINT FK_Floor FOREIGN KEY (Floor) REFERENCES Floor(Name))

```
create table Tables1
(TNumber Number(3) NOT NULL,
Floor varchar(25) NOT NULL,
NoSeats number(2),
CSSN number(8),
primary key (TNumber),
CONSTRAINT FK_CSSN FOREIGN KEY (CSSN) REFERENCES CUSTOMER(SSN),
CONSTRAINT FK_Floor FOREIGN KEY (Floor) REFERENCES Floor(Name))
```

Delivery:

create table Delivery

(LicenseID number(15) PRIMARY KEY NOT NULL,

Name varchar(50) NOT NULL,

RideType varchar(50) NOT NULL,

Baslc float NOT NULL,

Commission float)

```
create table Delivery
(LicenseID number(15) PRIMARY KEY NOT NULL,
Name varchar(50) NOT NULL,
RideType varchar(50) NOT NULL,
baslc float NOT NULL,
Commission float)
```

Merchant:

create table Merchant

(Name varchar(50) PRIMARY KEY NOT NULL,

Phone1 varchar(10) NOT NULL,

Phone2 varchar(10))

```
create table Merchant
(Name varchar(50) PRIMARY KEY NOT NULL,
Phone1 varchar(10) NOT NULL,
Phone2 varchar(10))
```

Results	Explain	Describe	Saved SQL	Histo
---------	---------	----------	-----------	-------

Table created.

0.15 seconds

Supply:

create table Supply

(Type varchar(50) PRIMARY KEY NOT NULL,

Amount number(10) NOT NULL,

price float,

Notes varchar(100),

SupplierName varchar(50),

CONSTRAINT FK_SupplierName FOREIGN KEY (SupplierName) REFERENCES Merchant(Name))

```
create table Supply
(Type varchar(50) PRIMARY KEY NOT NULL,
Amount number(10) NOT NULL,
price float,
Notes varchar(100),
SupplierName varchar(50),
CONSTRAINT FK_SupplierName FOREIGN KEY (SupplierName) REFERENCES Merchant(Name))
```

Results	Explain	Describe	Saved SQL	History
Table created.				
0.13 seconds				

Chef:

create table Chef

(ID number(8) PRIMARY KEY NOT NULL,

Name varchar(50) NOT NULL,

Specialty varchar(50) NOT NULL,

salary float NOT NULL)

```
create table Chef
(ID number(8) PRIMARY KEY NOT NULL,
Name varchar(50) NOT NULL,
Specialty varchar(50) NOT NULL,
salary float NOT NULL)
```

Results	Explain	Describe	Saved SQL
Table created.			
0.18 seconds			

Waiter:

create table Waiter

(ID number(8) PRIMARY KEY NOT NULL,

Name varchar(50) NOT NULL,

Basic float NOT NULL,

Tips float)

```
create table Waiter
(ID number(8) PRIMARY KEY NOT NULL,
Name varchar(50) NOT NULL,
Basic float NOT NULL,
Tips float)
```

Meal:

create table Meal

(Name varchar(50) PRIMARY KEY NOT NULL,

Calories float(10),

Types varchar(50),

Descriptions varchar(100))

```
create table Meal
(Name varchar(50) PRIMARY KEY NOT NULL,
Calories float(10),
Types varchar(50),
Descriptions varchar(100))
```

Orders:

create table Orders

(ONumber number(10) NOT NULL,

Day varchar(10) NOT NULL,

Month varchar(10) NOT NULL,

Year number(4) NOT NULL,

BillNumber NUMBER(8) Not NULL,

CSSN NUMBER(8) Not NULL,

WaiterID NUMBER(8) Not NULL,

Meal varchar(50),

Primary key (ONumber)

CONSTRAINT FK_BillNumber FOREIGN KEY (BillNumber) REFERENCES Bill(BILLNUMBER),

CONSTRAINT FK_CSSN2 FOREIGN KEY (CSSN) REFERENCES Customer(SSN),

CONSTRAINT FK_WaiterID FOREIGN KEY (WaiterID) REFERENCES Waiter(ID),

CONSTRAINT FK_Meal FOREIGN KEY (Meal) REFERENCES Meal(Name))

```
create table Orders
(ONumber number(10) NOT NULL,
Day varchar(10) NOT NULL,
Month varchar(10) NOT NULL,
Year number(4) NOT NULL,
BillNumber NUMBER(8) Not NULL,
CSSN NUMBER(8) Not NULL,
WaiterID NUMBER(8) Not NULL,
Meal varchar(50),
Primary key (ONumber, Day, Month, Year),
CONSTRAINT FK_BillNumber FOREIGN KEY (BillNumber) REFERENCES Bill(BILLNUMBER),
CONSTRAINT FK_CSSN2 FOREIGN KEY (CSSN) REFERENCES Customer(SSN),
CONSTRAINT FK_WaiterID FOREIGN KEY (WaiterID) REFERENCES Waiter(ID),
CONSTRAINT FK_Meal FOREIGN KEY (Meal) REFERENCES Meal(Name))
```

Results Explain Describe Saved SQL History

Table created.

0.38 seconds

Receiving:

create table Receiving

(OrderNo number(10) NOT NULL,

DAY VARCHAR(10) NOT NULL,

MONTH VARCHAR(10) NOT NULL,

YEAR NUMBER(4) NOT NULL,

ChefID number(8) NOT NULL,

Primary key (OrderNo, ChefID, DAY, Month, Year),

CONSTRAINT FK_OrderNo FOREIGN KEY (OrderNo, Day, Month, Year) REFERENCES Orders(ONUMBER, Day, Month, Year),

CONSTRAINT FK_ChefID FOREIGN KEY (ChefID) REFERENCES Chef(ID))

```
create table Receiving
(OrderNo number(10) NOT NULL,
DAY VARCHAR(10) NOT NULL,
MONTH VARCHAR(10) NOT NULL,
YEAR NUMBER(4) NOT NULL,
ChefID number(8) NOT NULL,
Primary key (OrderNo, ChefID, DAY, Month, Year),
CONSTRAINT FK_OrderNo FOREIGN KEY (OrderNo, Day, Month, Year) REFERENCES Orders(ONUMBER, Day, Month, Year),
CONSTRAINT FK_ChefID FOREIGN KEY (ChefID) REFERENCES Chef(ID))
```

Results Explain Describe Saved SQL History

Table created.

0.28 seconds

Making:

create table Making

(CustomerSSN number(10) NOT NULL,

OrderNo number(10) NOT NULL,

DAY VARCHAR(10) NOT NULL,

MONTH VARCHAR(10) NOT NULL,

YEAR NUMBER(4) NOT NULL,

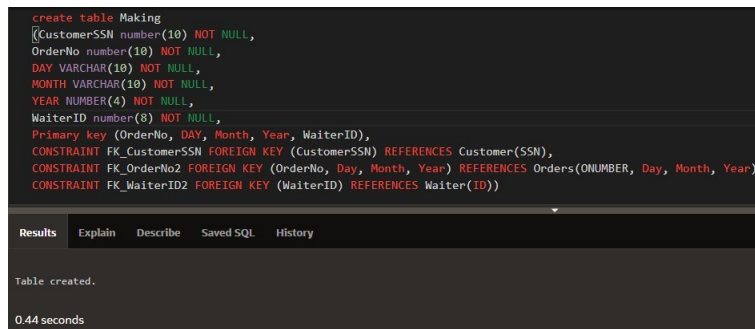
WaiterID number(8) NOT NULL,

Primary key (OrderNo, DAY, Month, Year, WaiterID),

CONSTRAINT FK_CustomerSSN FOREIGN KEY (CustomerSSN) REFERENCES Customer(SSN),

CONSTRAINT FK_OrderNo2 FOREIGN KEY (OrderNo, Day, Month, Year) REFERENCES Orders(ONUMBER, Day, Month, Year),

CONSTRAINT FK_WaiterID2 FOREIGN KEY (WaiterID) REFERENCES Waiter(ID))



```
create table Making
(
  CustomerSSN number(10) NOT NULL,
  OrderNo number(10) NOT NULL,
  DAY VARCHAR(10) NOT NULL,
  MONTH VARCHAR(10) NOT NULL,
  YEAR NUMBER(4) NOT NULL,
  WaiterID number(8) NOT NULL,
  Primary key (OrderNo, DAY, Month, Year, WaiterID),
  CONSTRAINT FK_CustomerSSN FOREIGN KEY (CustomerSSN) REFERENCES Customer(SSN),
  CONSTRAINT FK_OrderNo2 FOREIGN KEY (OrderNo, Day, Month, Year) REFERENCES Orders(ONUMBER, Day, Month, Year),
  CONSTRAINT FK_WaiterID2 FOREIGN KEY (WaiterID) REFERENCES Waiter(ID))
```

Results Explain Describe Saved SQL History

Table created.

0.44 seconds

Serving:

create table Serving

(TableNo number(3) NOT NULL,

Floor varchar(25) NOT NULL,

Meal varchar(50) NOT NULL,

WaiterID number(8) NOT NULL,

Primary key (TableNo, Floor, Meal, WaiterID),

CONSTRAINT FK_TableNo FOREIGN KEY (TableNo, Floor) REFERENCES Tables(TNumber, FLOOR),

CONSTRAINT FK_Meal2 FOREIGN KEY (Meal) REFERENCES Meal(Name),

CONSTRAINT FK_WaiterID3 FOREIGN KEY (WaiterID) REFERENCES Waiter(ID))

```
create table Serving
(TableNo number(3) NOT NULL,
Floor varchar(25) NOT NULL,
Meal varchar(50) NOT NULL,
WaiterID number(8) NOT NULL,
Primary key (TableNo, Floor, Meal, WaiterID),
CONSTRAINT FK_TableNo FOREIGN KEY (TableNo, Floor) REFERENCES Tables(TNumber, FLOOR),
CONSTRAINT FK_Meal2 FOREIGN KEY (Meal) REFERENCES Meal(Name),
CONSTRAINT FK_WaiterID3 FOREIGN KEY (WaiterID) REFERENCES Waiter(ID))
```

Results Explain Describe Saved SQL History

Table created.

0.35 seconds

Takeaway:

create table Takeaway

(DlvrID number(3) NOT NULL,

Meal varchar(50) NOT NULL,

Address varchar(100),

Primary key (DlvrID, Meal),

CONSTRAINT FK_DlvrID FOREIGN KEY (DlvrID) REFERENCES DELIVERY(LICENSEID),

CONSTRAINT FK_Meal3 FOREIGN KEY (Meal) REFERENCES Meal(Name))

```
create table Takeaway
(DlvrID number(3) NOT NULL,
Meal varchar(50) NOT NULL,
Address varchar(100),
Primary key (DlvrID, Meal),
CONSTRAINT FK_DlvrID FOREIGN KEY (DlvrID) REFERENCES DELIVERY(LICENSEID),
CONSTRAINT FK_Meal3 FOREIGN KEY (Meal) REFERENCES Meal(Name))
```

Results Explain Describe Saved SQL History

Table created.

0.27 seconds

Cooking:

create table Cooking

(ChefID number(10) NOT NULL,

Meal varchar(50) NOT NULL,

SplyType VARCHAR2(50) NOT NULL,

Primary key (ChefID, Meal, SplyType),

CONSTRAINT FK_ChefID2 FOREIGN KEY (ChefID) REFERENCES Chef(ID),

CONSTRAINT FK_Meal4 FOREIGN KEY (Meal) REFERENCES Meal(Name),

CONSTRAINT FK_SplyType FOREIGN KEY (SplyType) REFERENCES Supply(Type))


```
create table Cooking
(ChefID number(10) NOT NULL,
Meal varchar(50) NOT NULL,
SpIyType VARCHAR2(50) NOT NULL,
Primary key (ChefID, Meal, SpIyType),
CONSTRAINT FK_ChefID2 FOREIGN KEY (ChefID) REFERENCES Chef(ID),
CONSTRAINT FK_Meal4 FOREIGN KEY (Meal) REFERENCES Meal(Name),
CONSTRAINT FK_SpIyType FOREIGN KEY (SpIyType) REFERENCES Supply(Type))
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Table created.

0.29 seconds

Insert into

```
insert into Customer values(123,'a',04562582,'street01')
insert into Customer values(122,'b',04772582,'street02')
insert into Customer values(189,'e',04452582,'street03')
insert into Customer values(125,'c',04598582,'street04')
insert into Customer values(131,'d',04876582,'street05')
```

```
insert into Merchant ( Name, Phone1 ) values('B09',987647)
insert into Merchant values('A01',983097,9489435)
insert into Merchant values('F06',98767547,99865646)
insert into Merchant ( Name, Phone1 ) values('H10',84067547)
```

```
insert into Floor values('firest',20)
insert into Floor values('third',25)
insert into Floor values('scand',20)
insert into Floor values('fourth',10)
```

```
insert into Delivery values(987,'Ab','Motorcycle',1300,10)
insert into Delivery values(608,'Abd-Allah','Bike',1280,9.5)
insert into Delivery values(897,'Za','Motorcycle',1500,9)
insert into Delivery values(890,'Az','Bike',1500,19)
insert into Delivery values(709,'Ahmed','Bike',1270,9)
insert into Delivery values(937,'Bc','Motorcycle',1200,13)
insert into Delivery values(997,'Ac','Motorcycle',1500,10)
```

```
insert into Chef values(10,'abc','meat',2400)
insert into Chef values(19,'sah','sweet',2000)
insert into Chef values(13,'bac','meat',2400)
insert into Chef values(20,'has','sweet',1900)
```

```
insert into Bill values(105,50,50,5,0)
insert into Bill values(101,19,25,8,20)
insert into Bill values(103,27,40,5,20)
insert into Bill values(102,205,50,12,20)
```

```
insert into Waiter values(58,'hamed',1710,50)
insert into Waiter values(48,'hussin',1590,30)
insert into Waiter values(55,'hamza',1500,50)
insert into Waiter values(45,'rashed',1500,50)
```

```
insert into Meal (Name,Calories,Type) values('White Cake',200,'Sweet')
insert into Meal (Name,Calories,Type) values('Freid Chicken',1700,'Meat')
insert into Meal (Name,Calories,Type) values('Grilled Chicken',1100,'Meat')
insert into Meal (Name,Calories,Type) values('Whтите chocolate',2100,'Sweet')
```

Functions:

Function that returns a list of bills belong to a specific customer by accepting customerID(SSN):

```
CREATE OR REPLACE FUNCTION get_customer_bills(p_ssn NUMBER)
RETURN SYS_REFCURSOR
AS
    result_cursor SYS_REFCURSOR;
BEGIN
    OPEN result_cursor FOR
        SELECT bill.BillNumber, bill.BillCost, bill.Total, bill.Discount, bill.Remaining
        FROM Bill bill
        JOIN Orders ord ON ord.BillNumber = bill.BillNumber
        WHERE ord.CSSN = p_ssn;
    RETURN result_cursor;
END;
```

Function that returns a list of all orders delivered by a driver based on the driver's license ID:

```
CREATE OR REPLACE FUNCTION get_delivered_orders_by_driver_func(new_meal IN Takeaway.Meal%TYPE, new_dlv_r_id IN Takeaway.Dlv_rID%TYPE)
RETURN NUMBER IS
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM Orders o
    JOIN Delivery d ON o.WaiterID = d.LicenseID
    WHERE o.Meal = new_meal
    AND d.LicenseID = new_dlv_r_id;

    IF v_count > 0 THEN
        DBMS_OUTPUT.PUT_LINE('Delivery person ' || new_dlv_r_id || ' has delivered ' || v_count || ' orders for meal ' || new_meal);
    END IF;

    RETURN v_count;
END;
```

A function calculates the total bill cost based on the bill cost and discount:

```
CREATE OR REPLACE FUNCTION calc_total_bill (
    p_bill_cost IN Bill.BillCost%TYPE,
    p_discount IN Bill.Discount%TYPE
) RETURN Bill.Total%TYPE AS
BEGIN
    RETURN p_bill_cost - (p_bill_cost * p_discount);
END calc_total_bill;
```

Procedures:

Procedure to update the "notes" column in the "supply" table on any changes, and raises an exception in case the supplier has not data in the database:

```
CREATE OR REPLACE PROCEDURE update_supplier_notes (  
    p_supplier_name IN supply.SupplierName%TYPE,  
    p_new_notes IN VARCHAR2)  
AS  
BEGIN  
    UPDATE supply  
    SET notes = p_new_notes  
    WHERE SupplierName = p_supplier_name;  
  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
        dbms_output.put_line(' ||SupplierName|| does not exist in the suppliers table');  
END;
```

A procedure takes a customer's SSN as an input parameter (p_ssn), and deletes the corresponding record from the making table:

```
CREATE OR REPLACE PROCEDURE delete_customerID (p_ssn IN NUMBER)  
AS  
BEGIN  
    DELETE FROM making  
    WHERE CustomerSSN = p_ssn;  
END;
```

This stored procedure insert_bill inserts a new bill into the Bill table:

```
1  CREATE OR REPLACE PROCEDURE insert_bill (  
2      p_bill_number  IN Bill.BillNumber%TYPE,  
3      p_bill_cost    IN Bill.BillCost%TYPE,  
4      p_total        IN Bill.Total%TYPE,  
5      p_discount     IN Bill.Discount%TYPE,  
6      p_remaining    IN Bill.Remaining%TYPE  
7  ) AS  
8  BEGIN  
9      INSERT INTO Bill (BillNumber, BillCost, Total, Discount, Remaining)  
10     VALUES (p_bill_number, p_bill_cost, p_total, p_discount, p_remaining);  
11  END insert_bill;
```

```

DECLARE
    pp_supplier_name supply.SupplierName%TYPE := 'RAGHAD' ; -- example value for the supplier name
    new_notes VARCHAR2(100) := 'New supplier notes'; -- example value for the new notes
BEGIN
    update_supplier_notes(pp_supplier_name, new_notes);
    dbms_output.put_line( '||pp_supplier_name'|| 'notes updated successfully.' );
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        dbms_output.put_line('no data found');
END;

```

```

language PL/SQL Rows 10
DECLARE
    my_cursor SYS_REFCURSOR;
    onumber orders.onumber%TYPE;
BEGIN
    my_cursor := get_delivered_orders_by_driver(78);
    LOOP
        FETCH my_cursor INTO onumber;
        EXIT WHEN my_cursor%NOTFOUND;
        dbms_output.put_line('Delivered order number: ' || onumber);
    END LOOP;
    CLOSE my_cursor;
END;

insert into

```

```

DECLARE
    pp_supplier_name supply.SupplierName%TYPE := ' Ahmed ' ;
    new_notes VARCHAR2(100) := 'New supplier notes';

    update_supplier_notes(pp_supplier_name, new_notes)
    dbms_output.put_line( 'notes updated successfully');
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        dbms_output.put_line('no data found')
END

```

Triggers:

The trigger can check if the amount of the updated row has dropped to half of its original value and then display a warning message.

```

CREATE OR REPLACE TRIGGER supply_amount_warning
AFTER UPDATE ON supply
FOR EACH ROW
DECLARE
    v_old_amount NUMBER;
BEGIN
    -- Get the old value of the Amount column
    SELECT amount
    INTO v_old_amount
    FROM supply
    WHERE type = :old.type;

    -- Check if the new value is less than half of the old value
    IF :new.amount < v_old_amount/2 THEN
        -- Display a warning message to the user
        dbms_output.put_line('Warning: the amount for supply ' || :new.type || ' has dropped to half of its original value!');
    END IF;
END;

```

The trigger check if any change happens on name >> merchant table then trigger update the name on suppliername >> supply table

```
CREATE OR REPLACE TRIGGER merchant_name_update
AFTER UPDATE OF name ON merchant
FOR EACH ROW
BEGIN
    -- Update the SupplierName column in the Supply table for all rows that reference the old name
    UPDATE supply
    SET suppliername = :new.name
    WHERE suppliername = :old.name;
END;
```

The trigger check if any waiter take 100 order or more in one day the tips will be double

```
CREATE OR REPLACE TRIGGER double_tips_trigger
AFTER INSERT ON Orders
FOR EACH ROW
DECLARE
    v_order_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_order_count
    FROM Orders
    WHERE WaiterID = :NEW.WaiterID
    AND Day = :NEW.Day
    AND Month = :NEW.Month
    AND Year = :NEW.Year;

    IF v_order_count > 100 THEN
        UPDATE Waiter
        SET Tips = Tips * 2
        WHERE ID = :NEW.WaiterID;
    END IF;
END;
```

Package:

This package defines a set of functions and procedures related to a restaurant. It has two variables, to store the restaurant name and number of tables. The package includes a procedure `update_no_of_tables` to update the number of tables and a function `get_restaurant_name` to retrieve the restaurant name:

Package:

```
CREATE OR REPLACE PACKAGE restaurant_package AS
    no_of_tables NUMBER := 10;
    restaurant_name VARCHAR2(50) := 'Burger Restaurant';
    PROCEDURE update_no_of_tables (p_no_of_tables NUMBER);
    FUNCTION get_restaurant_name RETURN VARCHAR2;
END restaurant_package;
```

Package body:

```
CREATE OR REPLACE PACKAGE BODY restaurant_package AS

    PROCEDURE update_no_of_tables (p_no_of_tables NUMBER) IS
    BEGIN
        no_of_tables := p_no_of_tables;
    END update_no_of_tables;

    FUNCTION get_restaurant_name RETURN VARCHAR2 IS
    BEGIN
        RETURN restaurant_name;
    END get_restaurant_name;
END restaurant_package;
```


Cursor:

This procedure "show_meals" establishes a cursor called "meal_cursor" that retrieves data from the "Meal" table. The cursor is then opened and a loop is used to fetch each row of the result set and assign it to a record called "meal_record". Within the loop, the values of each column in the record are displayed by utilizing "DBMS_OUTPUT.PUT_LINE". Finally, the cursor is closed after all rows have been processed.

```
CREATE OR REPLACE PROCEDURE show_meals AS
  CURSOR meal_cursor IS
    SELECT Name, Calories, Typess, Descriptions
    FROM Meal;
  meal_record meal_cursor%ROWTYPE;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Name' || CHR(9) || 'Calories' || CHR(9) || 'Type' || CHR(9) || 'Description');
  OPEN meal_cursor;
  FETCH meal_cursor INTO meal_record;
  WHILE meal_cursor%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE(meal_record.Name || CHR(9) || meal_record.Calories || CHR(9) || meal_record.Typess || CHR(9) || meal_record.Descriptions);
    FETCH meal_cursor INTO meal_record;
  END LOOP;
  CLOSE meal_cursor;
END show_meals;
```

Roles and users:

Creating a role named `role_admin` and granting administrative privileges such as `CREATE SESSION`, `CREATE PROCEDURE`, `CREATE TABLE`, and `ALTER ANY TABLE` to that role. Then, a user named `user_admin` is created and granted the `role_admin` role with the password `Admin@123`.

```
1 CREATE ROLE role_admin;
2
3 GRANT CREATE SESSION, CREATE PROCEDURE, CREATE TABLE, ALTER ANY TABLE TO role_admin;
4
5 CREATE USER Raghad IDENTIFIED BY Raghadd;
6
7 GRANT role_admin TO Raghad;
```

Creating a role named `user` and granting privileges such as `CREATE SESSION`, `SELECT`, `INSERT`, `UPDATE`, and `DELETE` on all tables in the `restaurant` schema to that role. Then, two users are created without any password and granted the `user` role.

```
1 CREATE ROLE user;
2
3 GRANT CREATE SESSION, SELECT, INSERT, UPDATE, DELETE
4 ON restaurant.*
5 TO user;
6
7 CREATE USER Rand;
8 CREATE USER Maryam;
9
10 GRANT user TO Maryam, Rand;
```

Creating a role named `waiter` and grant it specific privileges (`SELECT`, `INSERT`, and `UPDATE`) on two tables `Orders` and `Serving`. It is also granted `SELECT` privilege on `Bill` and `Customer` tables.

```
1 CREATE ROLE waiter;
2
3 GRANT SELECT, INSERT, UPDATE ON restaurant.Orders TO waiter;
4 GRANT SELECT, INSERT, UPDATE ON restaurant.Serving TO waiter;
5 GRANT SELECT ON restaurant.Bill TO waiter;
6 GRANT SELECT ON restaurant.Customer TO waiter;
7
8 CREATE USER Sadeen IDENTIFIED BY Sadeenn;
9
10 GRANT waiter TO Sadeen;
```

Creating a role named Chef and granting it the privileges (SELECT, INSERT, UPDATE, and DELETE) on tables (Orders, Receiving, and Making). It is also granted SELECT privileges on three tables (Chef, Meal, and Supply).

```
1  CREATE ROLE Chef;
2
3  GRANT SELECT, INSERT, UPDATE, DELETE ON restaurant.Orders TO Chef;
4  GRANT SELECT, INSERT, UPDATE, DELETE ON restaurant.Receiving TO Chef;
5  GRANT SELECT, INSERT, UPDATE, DELETE ON restaurant.Making TO Chef;
6  GRANT SELECT ON restaurant.Chef TO Chef;
7  GRANT SELECT ON restaurant.Meal TO Chef;
8  GRANT SELECT ON restaurant.Supply TO Chef;
9
10 CREATE USER Reem IDENTIFIED BY Reemm;
11
12 GRANT Chef TO Reem;
```