Islamic University of Technology

CSE-4410
Database Management Systems - II Lab

Lab Report - 1

Name: Mukit Mahdin
ID: 200042170
Department: CSE
Program: SWE

Task :

Suppose you are given the task of automating the operations of an international food chain via a single platform. There are multiple franchises (KFC, Chester's, Pizza hut, Domino's Pizza) of the food chain spread across over 20 countries. Each of the franchises gets at least 10,0000 customers per year. Customers can register themselves under different franchises to order food from different branches of that specific franchise. Each franchise has multiple branches spread around the country. Each branch has its own team of chefs. Each of them is an expert in a particular cuisine. And any customer can see the basic information of the chefs such as special menus developed by them (up to 5 menus). Each franchise has multiple menus (some are unique and some are common) that they offer to customers. The menus are identified by their own name, cuisine, main ingredients(optional), price, calorie_count, etc. Further to build a proper food recommendation system for the food chain, information about customer details, their personal preferred cuisines (a person can have multiple preferred ones), and customers' ratings of any food that needs to be stored.

DDL Tables Explanation:

There are 7 tables to cover the whole scenario.

1. **customers:** stores customer information, including their preferred cuisine(s).
2. **franchises:** stores franchise information.
3. **branches:** stores branch information, including the franchise it belongs to.
4. **chefs:** stores chef information, including the branch they work at and their expertise in a particular cuisine.
5. **menus:** stores menu information, including the chef who created it, its name, cuisine, main ingredients (if any), price, and calorie count.
6. **customer_ratings:** stores customer ratings for a particular menu.
7. **Orders:** stores information about orders

The tables are related to each other through foreign keys:

- branches and franchises are related through **franchise_id.**
- chefs and branches are related through **branch_id**.
- menus and chefs are related through **chef_id.**
- customer_ratings and customers are related through **customer_id.**
- customer_ratings and menus are related through **menu_id.**
- The orders table has foreign keys to the customers, menus, and branches tables to establish relationships between these tables.

Task - 1 : DDL Code :

```
CREATE TABLE CUSTOMERS(
    ID INTEGER PRIMARY KEY,
    NAME VARCHAR(255) NOT NULL,
    EMAIL VARCHAR(255) NOT NULL,
    PREFERRED_FOOD VARCHAR(255) NOT NULL,
    FOREIGN KEY(PREFERRED_FOOD) REFERENCES MENUS(NAME);
);

CREATE TABLE FRANCHISES(
    ID INTEGER PRIMARY KEY,
    NAME VARCHAR(255) NOT NULL,
);


CREATE TABLE BRANCHES (
    ID INTEGER PRIMARY KEY,
    FRANCHISE_ID INTEGER NOT NULL,
    LOCATION VARCHAR(255) NOT NULL,
    FOREIGN KEY (FRANCHISE_ID) REFERENCES FRANCHISES(ID)
);


CREATE TABLE CHEFS(
    ID INTEGER PRIMARY KEY,
    BRANCH_ID INTEGER NOT NULL,
    NAME VARCHAR(255) NOT NULL,
    CUISINE VARCHAR(255) NOT NULL,
    FOREIGN KEY (BRANCH_ID) REFERENCES BRANCHES(ID)
);

CREATE TABLE MENUS (
    ID INTEGER PRIMARY KEY,
    CHEF_ID INTEGER NOT NULL,
    NAME VARCHAR(255) NOT NULL,
    CUISINE VARCHAR(255) NOT NULL,
    MAIN_INGREDIENTS VARCHAR(255) NOT NULL,
    PRICE DECIMAL NOT NULL,
    CALORIE_COUNT INTEGER NOT NULL,
    FOREIGN KEY (CHEF_ID) REFERENCES CHEFS(ID)
);
```

```
CREATE TABLE ORDERS(
    ID INTEGER PRIMARY KEY,
    CUSTOMER_ID INTEGER NOT NULL,
    MENU_ID INTEGER NOT NULL,
    BRANCH_ID INTEGER NOT NULL,
    ORDER_DATE DATE NOT NULL,
    FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMERS(ID),
    FOREIGN KEY (MENU_ID) REFERENCES MENUS(ID),
    FOREIGN KEY (BRANCH_ID) REFERENCES BRANCHES(ID)
);


CREATE TABLE CUSTOMER_RATINGS(
    ID INTEGER PRIMARY KEY,
    CUSTOMER_ID INTEGER NOT NULL,
    MENU_ID INTEGER NOT NULL,
    ORDER_ID INTEGER NOT NULL,
    RATING INTEGER NOT NULL,
    FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMERS(ID),
    FOREIGN KEY (MENU_ID) REFERENCES MENUS(ID),
    FOREIGN KEY (ORDER_ID) REFERENCES ORDERS(ID)
);
```

Task - 2: Write SQL statements for the following queries:

a. **Find the total number of customers for each franchise.**

```
SELECT F.NAME, COUNT(C.ID) AS TOTAL_CUSTOMERS
FROM FRANCHISES F
LEFT JOIN BRANCHES B ON B.FRANCHISE_ID = F.ID
GROUP BY F.NAME;
```

It selects the name column from the franchises table and the count of id values from the customers table (aliased as total_customers). Then performs a LEFT JOIN on the branches and franchises tables, using franchise_id as the join condition. This allows the query to retrieve all franchises, even if they don't have any branches. It groups the results by name from the franchises table.

b. Find the avg rating for each menu item among all franchises.

```
SELECT M.NAME, AVG(CR.RATING) AS AVG_RATING
FROM MENUS M
NATURAL JOIN CUSTOMER_RATINGS CR ON CR.MENU_ID = M.ID
GROUP BY M.NAME
ORDER BY AVG_RATING DESC;
```

It selects the name column from the menus table and the average of rating values from the customer_ratings table (aliased as avg_rating). Then joins the customer_ratings and menus tables using the menu_id column as the join condition. Group the results by name from the menus table.

c. Find the 5 top most popular items. It should be based on the number of times they were ordered.

```
SELECT M.NAME, COUNT(O.ID) AS TIMES_ORDERED
FROM MENUS M
LEFT JOIN ORDERS O ON O.MENU_ID = M.ID
GROUP BY M.NAME
ORDER BY TIMES_ORDERED DESC
LIMIT 5;
```

It selects the name column from the menus table and the count of id values from the orders table (aliased as times_ordered). Then, performs a LEFT JOIN on the orders and menus tables, using menu_id as the join condition. This allows the query to include all menu items, even if they haven't been ordered. Then group the results by name from the menus table. Then orders the results in descending order by times_ordered. It limits the results to the top 5 rows.

d. Find the names of all customers who have preferred food that is offered from at least 2 different franchises.

```
SELECT C.NAME
FROM CUSTOMERS C
JOIN (
    SELECT PREFERRED_FOOD, COUNT(DISTINCT FRANCHISE_ID) AS
NUM_FRANCHISES
    FROM (
        SELECT PREFERRED_FOOD, F.ID AS FRANCHISE_ID
        FROM CUSTOMERS C
        JOIN MENUS M ON M.CUISINE = C.PREFERRED_FOOD
        JOIN CHEFS CH ON CH.ID = M.CHEF_ID
        JOIN BRANCHES B ON B.ID = CH.BRANCH_ID
        JOIN FRANCHISES F ON F.ID = B.FRANCHISE_ID
    ) T

    GROUP BY PREFERRED_FOOD
    HAVING NUM_FRANCHISES >= 2
)
T ON T.PREFERRED_FOOD = C.PREFERRED_FOOD;
```

T selects the name column from the customers table. It joins a subquery (aliased as t) on the customers table using the preferred_food column as the join condition. The subquery selects the preferred_food and franchise_id columns and counts the number of distinct franchise_id values (aliased as num_franchises). The subquery is constructed using another subquery (aliased as t), which selects the preferred_food, franchise_id, and id columns from the customers, menus, chefs, branches, and franchises tables. It uses a series of JOIN clauses to relate these tables to each other.
The outermost subquery groups the results by preferred_food and filters out any rows where num_franchises is less than 2.

e. Find the names of all customers who have not placed any orders.

```
SELECT C.NAME
FROM CUSTOMERS C
LEFT JOIN ORDERS O ON O.CUSTOMER_ID = C.ID
WHERE O.ID IS NULL
```

It selects the name column from the customers table. Then performs a LEFT JOIN on the orders and customers tables, using customer_id as the join condition. This allows the query to include all customers, even if they haven't placed any orders. It filters the results to include only rows where the id column from the orders table is NULL, indicating that the customer has not placed any orders.