**DATABASE SYSTEMS-I**

**(CSC-371)**

**LAB MID TERM**



|  |  |
| --- | --- |
| **SUBMITTED BY** | Mian Muhammad Tayyab  Muhammad Moiz |
| **Registration**  **Number** | FA21-BSE-041  FA21-BSE-044 |
| **Class** | BSE-4A |
| **Instructor’s**  **Name** | MA’AM SADIA MARYUM |

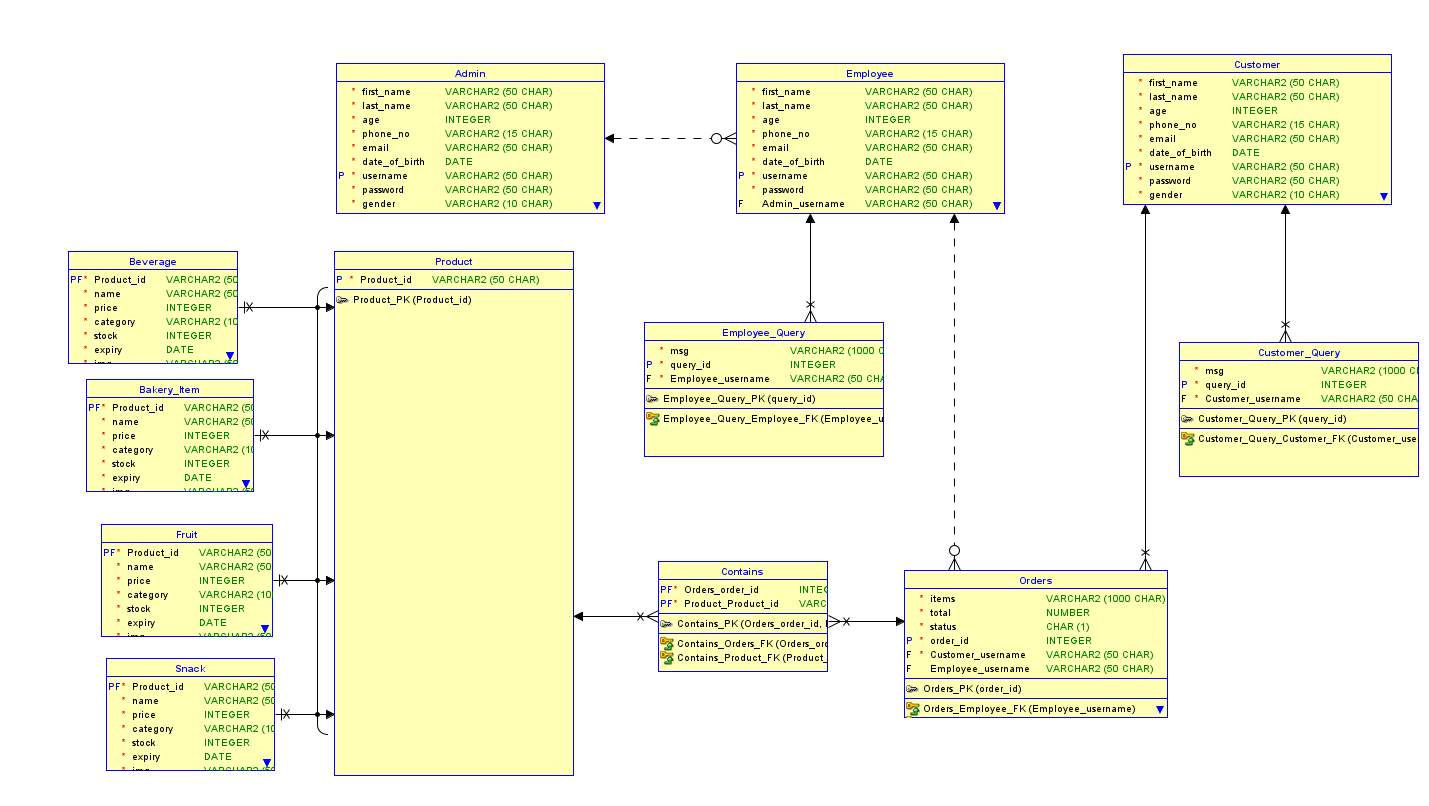
**Department of Computer Science**

**Comsats University, Islamabad**

**SUPERMARKET DATABASE SYSTEM**

**Part-I:** Design and develop your own project EER diagram, create the same using a modeling tool of your choice. (Oracle’s Data Modeler tool is preferred). **[2]**

**ER Diagram:**

****

**Part-II:** Export the DDL script, corresponding to the relational schema of the EERD created in Part-I, to Oracle Database System. **[2]**

**DDL Script:**

CREATE TABLE admin (

first\_name VARCHAR2(50 CHAR) NOT NULL,

last\_name VARCHAR2(50 CHAR) NOT NULL,

age INTEGER NOT NULL,

phone\_no VARCHAR2(15 CHAR) NOT NULL,

email VARCHAR2(50 CHAR) NOT NULL,

date\_of\_birth DATE NOT NULL,

username VARCHAR2(50 CHAR) NOT NULL,

password VARCHAR2(50 CHAR) NOT NULL,

gender VARCHAR2(10 CHAR) NOT NULL

);

ALTER TABLE admin ADD CONSTRAINT admin\_pk PRIMARY KEY ( username );

CREATE TABLE bakery\_item (

product\_id VARCHAR2(50 CHAR) NOT NULL,

name VARCHAR2(50 CHAR) NOT NULL,

price INTEGER NOT NULL,

category VARCHAR2(100 CHAR) NOT NULL,

stock INTEGER NOT NULL,

expiry DATE NOT NULL,

img VARCHAR2(50 CHAR) NOT NULL

);

ALTER TABLE bakery\_item ADD CONSTRAINT bakery\_item\_pk PRIMARY KEY ( product\_id );

CREATE TABLE beverage (

product\_id VARCHAR2(50 CHAR) NOT NULL,

name VARCHAR2(50 CHAR) NOT NULL,

price INTEGER NOT NULL,

category VARCHAR2(100 CHAR) NOT NULL,

stock INTEGER NOT NULL,

expiry DATE NOT NULL,

img VARCHAR2(50 CHAR) NOT NULL

);

ALTER TABLE beverage ADD CONSTRAINT beverage\_pk PRIMARY KEY ( product\_id );

CREATE TABLE contains (

orders\_order\_id INTEGER NOT NULL,

product\_product\_id VARCHAR2(50 CHAR) NOT NULL

);

ALTER TABLE contains ADD CONSTRAINT contains\_pk PRIMARY KEY ( orders\_order\_id,

product\_product\_id );

CREATE TABLE customer (

first\_name VARCHAR2(50 CHAR) NOT NULL,

last\_name VARCHAR2(50 CHAR) NOT NULL,

age INTEGER NOT NULL,

phone\_no VARCHAR2(15 CHAR) NOT NULL,

email VARCHAR2(50 CHAR) NOT NULL,

date\_of\_birth DATE NOT NULL,

username VARCHAR2(50 CHAR) NOT NULL,

password VARCHAR2(50 CHAR) NOT NULL,

gender VARCHAR2(10 CHAR) NOT NULL

);

ALTER TABLE customer ADD CONSTRAINT customer\_pk PRIMARY KEY ( username );

CREATE TABLE customer\_query (

msg VARCHAR2(1000 CHAR) NOT NULL,

query\_id INTEGER NOT NULL,

customer\_username VARCHAR2(50 CHAR) NOT NULL

);

ALTER TABLE customer\_query ADD CONSTRAINT customer\_query\_pk PRIMARY KEY ( query\_id );

CREATE TABLE employee (

first\_name VARCHAR2(50 CHAR) NOT NULL,

last\_name VARCHAR2(50 CHAR) NOT NULL,

age INTEGER NOT NULL,

phone\_no VARCHAR2(15 CHAR) NOT NULL,

email VARCHAR2(50 CHAR) NOT NULL,

date\_of\_birth DATE NOT NULL,

username VARCHAR2(50 CHAR) NOT NULL,

password VARCHAR2(50 CHAR) NOT NULL,

admin\_username VARCHAR2(50 CHAR),

gender VARCHAR2(10 CHAR) NOT NULL

);

ALTER TABLE employee ADD CONSTRAINT employee\_pk PRIMARY KEY ( username );

CREATE TABLE employee\_query (

msg VARCHAR2(1000 CHAR) NOT NULL,

query\_id INTEGER NOT NULL,

employee\_username VARCHAR2(50 CHAR) NOT NULL

);

ALTER TABLE employee\_query ADD CONSTRAINT employee\_query\_pk PRIMARY KEY ( query\_id );

CREATE TABLE fruit (

product\_id VARCHAR2(50 CHAR) NOT NULL,

name VARCHAR2(50 CHAR) NOT NULL,

price INTEGER NOT NULL,

category VARCHAR2(100 CHAR) NOT NULL,

stock INTEGER NOT NULL,

expiry DATE NOT NULL,

img VARCHAR2(50 CHAR) NOT NULL

);

ALTER TABLE fruit ADD CONSTRAINT fruit\_pk PRIMARY KEY ( product\_id );

CREATE TABLE orders (

items VARCHAR2(1000 CHAR) NOT NULL,

total NUMBER NOT NULL,

status CHAR(1) NOT NULL,

order\_id INTEGER NOT NULL,

customer\_username VARCHAR2(50 CHAR) NOT NULL,

employee\_username VARCHAR2(50 CHAR)

);

ALTER TABLE orders ADD CONSTRAINT orders\_pk PRIMARY KEY ( order\_id );

CREATE TABLE product (

product\_id VARCHAR2(50 CHAR) NOT NULL

);

ALTER TABLE product ADD CONSTRAINT product\_pk PRIMARY KEY ( product\_id );

CREATE TABLE snack (

product\_id VARCHAR2(50 CHAR) NOT NULL,

name VARCHAR2(50 CHAR) NOT NULL,

price INTEGER NOT NULL,

category VARCHAR2(100 CHAR) NOT NULL,

stock INTEGER NOT NULL,

expiry DATE NOT NULL,

img VARCHAR2(50 CHAR) NOT NULL

);

ALTER TABLE snack ADD CONSTRAINT snack\_pk PRIMARY KEY ( product\_id );

ALTER TABLE bakery\_item

ADD CONSTRAINT bakery\_item\_product\_fk FOREIGN KEY ( product\_id )

REFERENCES product ( product\_id )

ON DELETE CASCADE;

ALTER TABLE beverage

ADD CONSTRAINT beverage\_product\_fk FOREIGN KEY ( product\_id )

REFERENCES product ( product\_id )

ON DELETE CASCADE;

ALTER TABLE contains

ADD CONSTRAINT contains\_orders\_fk FOREIGN KEY ( orders\_order\_id )

REFERENCES orders ( order\_id )

ON DELETE CASCADE;

ALTER TABLE contains

ADD CONSTRAINT contains\_product\_fk FOREIGN KEY ( product\_product\_id )

REFERENCES product ( product\_id )

ON DELETE CASCADE;

ALTER TABLE customer\_query

ADD CONSTRAINT customer\_query\_customer\_fk FOREIGN KEY ( customer\_username )

REFERENCES customer ( username )

ON DELETE CASCADE;

ALTER TABLE employee

ADD CONSTRAINT employee\_admin\_fk FOREIGN KEY ( admin\_username )

REFERENCES admin ( username )

ON DELETE SET NULL;

ALTER TABLE employee\_query

ADD CONSTRAINT employee\_query\_employee\_fk FOREIGN KEY ( employee\_username )

REFERENCES employee ( username )

ON DELETE CASCADE;

ALTER TABLE fruit

ADD CONSTRAINT fruit\_product\_fk FOREIGN KEY ( product\_id )

REFERENCES product ( product\_id )

ON DELETE CASCADE;

ALTER TABLE orders

ADD CONSTRAINT orders\_customer\_fk FOREIGN KEY ( customer\_username )

REFERENCES customer ( username )

ON DELETE CASCADE;

ALTER TABLE orders

ADD CONSTRAINT orders\_employee\_fk FOREIGN KEY ( employee\_username )

REFERENCES employee ( username )

ON DELETE SET NULL;

ALTER TABLE snack

ADD CONSTRAINT snack\_product\_fk FOREIGN KEY ( product\_id )

REFERENCES product ( product\_id )

ON DELETE CASCADE;

**Part-III:** Implement the EER constraints 1-1, 1-many, many to many, “U”, “O”, “d”, “Partial participation”, “Total participation” , multi-attribute, super-class, sub-class, multiple-inheritance ( Can Java language ). **[8]**

1. **One to One Relationship:**

According to the above DDL script there is no explicit 1-1 relationship among any entities.

1. **One to Many Relationship:**

The relationship between the "CUSTOMER" table and the "ORDER" table represents a one-to-many relationship. Each customer can have multiple orders, while each order belongs to one customer. This is implemented by adding a foreign key constraint on orders table referencing customer table

1. **Many to Many Relationship:**

The relationship between the "PRODUCT" table and the "ORDERS" table represents a many-to-many relationship. Each product can have multiple orders and each order can have multiple products. This is implemented by introducing a junction table that holds the combination of product and order information. In this case, the "CONTAINS" table serves as the junction table, connecting product and order.

1. **Union (U) Constraints:**

There are no Union constraints defined according to the above DDL script

1. **Overlapping (O) Constraints:**

There are no overlapping constraints according to above DDL script

1. **Disjoint (d) Constraints:**

According to above DDL script, product can only be fruit, beverage, bakery item or snack at one time, which means there exists disjoint constraints in the above DDL script where Product is the super type of fruit, bakery items, snack and beverage.

1. **Partial Participation:**

Partial Participation can be seen in the relationship between the "EMPLOYEE" table and the "ADMIN" table.

1. **Total Participation:**

Total participation is not explicitly defined in the given DDL script.

1. **Multiple Attributes:**

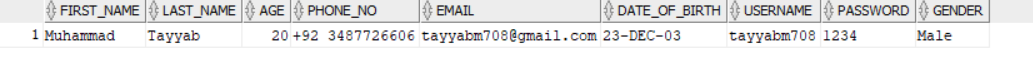
There are no multi attribute relationships according to above DDL script

**Super Class, Sub class and multiple inheritance:**

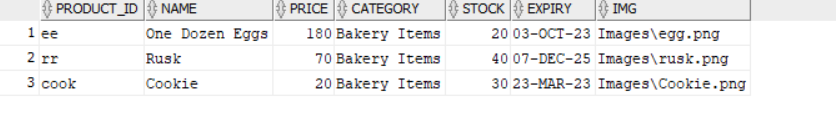
According to above DDL script, Product is the super class of bakery items, fruit, beverage, snack.

**Part-IV:** Insert a few meaningful tuples in the resultant relations **[3]**

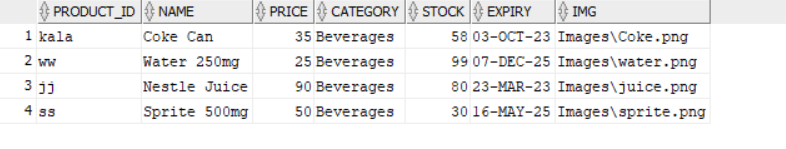
**Inserting into Admin Table:**



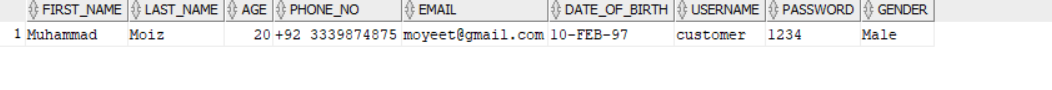
**Inserting into Bakery Item Table:**



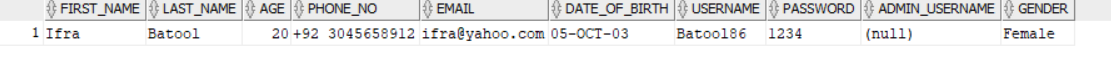
**Inserting into Beverage Table:**



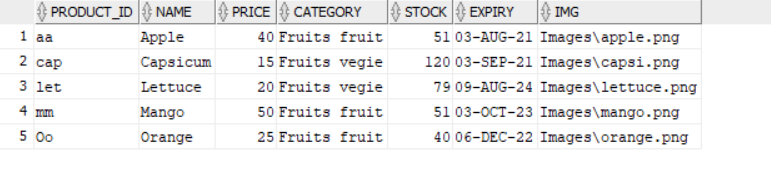
**Inserting into Customer Table:**



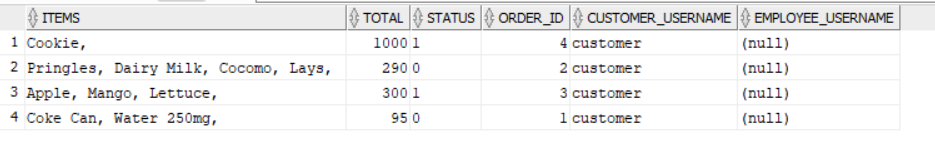
**Inserting into Employee Table:**



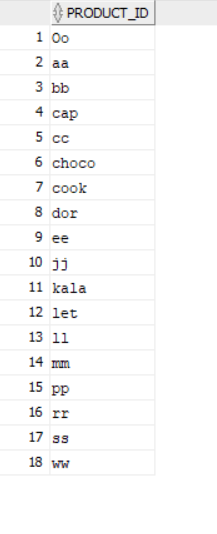
**Inserting into Fruit Table:**



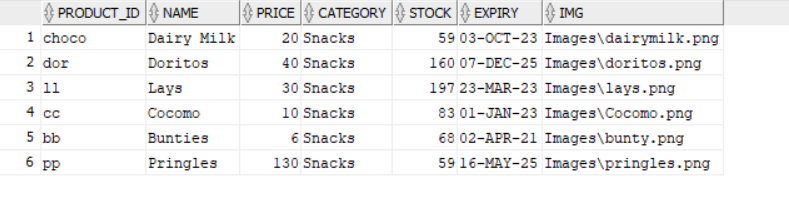
**Inserting into Orders Table:**



**Inserting into Product Table:**

****

**Inserting into Snacks Table:**

****

**Part-V:** Now in this part, you are going to first create a meaningful query in English and then in SQL for each of the following situations:

* **Think of a need that will require joining at least 3 tables. [3]**

**English Statement:**

Retrieve the names and contact information of customers who have placed an order containing bakery items, along with the details of the bakery items they ordered.

**SQL Query:**

SELECT c.first\_name, c.last\_name, c.phone\_no, c.email, b.name AS bakery\_item\_name

FROM customer c

JOIN orders o ON c.username = o.customer\_username

JOIN contains co ON o.order\_id = co.orders\_order\_id

JOIN bakery\_item b ON co.product\_product\_id = b.product\_id;

**Explanation:**

This query involves joining three tables (customer, orders, and contains) to retrieve information about customers and their orders. The join conditions are based on the relationships between these tables. The query selects the first name, last name, phone number, and email of customers from the customer table. It also includes the name of the bakery items they ordered by joining the orders, contains, and bakery\_item tables. The join between customer and orders is based on the customer username, the join between orders and contains is based on the order ID, and the join between contains and bakery\_item is based on the product ID. The result provides the names and contact information of customers who placed orders containing bakery items, along with the details of those bakery items.

* **Think of a need that will require the use of join with grouping and aggregation. [3]**

**English Statement:**

Retrieve the average price and the maximum stock quantity for each category of beverages.

**SQL Query:**

SELECT b.category, AVG(b.price) AS average\_price, MAX(b.stock) AS maximum\_stock

FROM beverage b

GROUP BY b.category;

**Explanation:**

This query requires the use of join, grouping, and aggregation. It retrieves the category of beverages, the average price for each category (average\_price), and the maximum stock quantity for each category (maximum\_stock). The join is not necessary in this case as we are not combining information from multiple tables. The grouping is done based on the category column in the beverage table. The AVG() function calculates the average price, and the MAX() function calculates the maximum stock quantity. The result is grouped by category, providing the desired information for each category of beverages.

* **Think of a need that involves the need of using any of set operators . [2]**

**English Statement:**

Retrieve a list of bakery items that are either out of stock or have expired.

**SQL Query:**

SELECT product\_id, name, category, stock, expiry

FROM bakery\_item

WHERE stock = 0

UNION

SELECT product\_id, name, category, stock, expiry

FROM bakery\_item

WHERE expiry < SYSDATE;

**Explanation:**

This query involves the use of set operators, specifically the UNION operator. It retrieves a list of bakery items that meet two conditions: either they are out of stock or they have expired. The first SELECT statement retrieves the product ID, name, category, stock, and expiry date of bakery items where the stock quantity is 0. The second SELECT statement retrieves the same information for bakery items where the expiry date is less than the current date (SYSDATE). The UNION operator combines the results of both SELECT statements and removes any duplicates. The final result is a list of bakery items that are either out of stock or have expired.

* **Think of a need that involves the need of using a sub query [2]**

**English Statement:**

Retrieve the names of customers who have placed orders with a total amount higher than the average total amount of all orders.

**SQL Query:**

SELECT first\_name, last\_name

FROM customer

WHERE username IN (

SELECT customer\_username

FROM orders

WHERE total > (

SELECT AVG(total)

FROM orders

)

);

**Explanation:**

This query requires the use of a subquery. It retrieves the names of customers who have placed orders with a total amount higher than the average total amount of all orders. The subquery calculates the average total amount from the orders table. The outer query then selects the first name and last name from the customer table where the customer username is found in the result of the subquery. The WHERE clause filters out customers whose orders' total amounts are higher than the calculated average. The final result is the names of customers who meet the specified condition.