Ice-Cream Manufacturing DBMS

By: Anthony Pochapsky, Frick Wu, Jason Chen
November 27, 2023
Tajali - 112

Project Description:

The Ice-cream manufacturing DBMS is a database that manages stock and price of three product types and handles the customers, orders and employee information of multiple locations throughout Toronto. Example tasks involve storing customer orders, updating stock whenever a purchase is completed, and managing the pay of employees in each location.

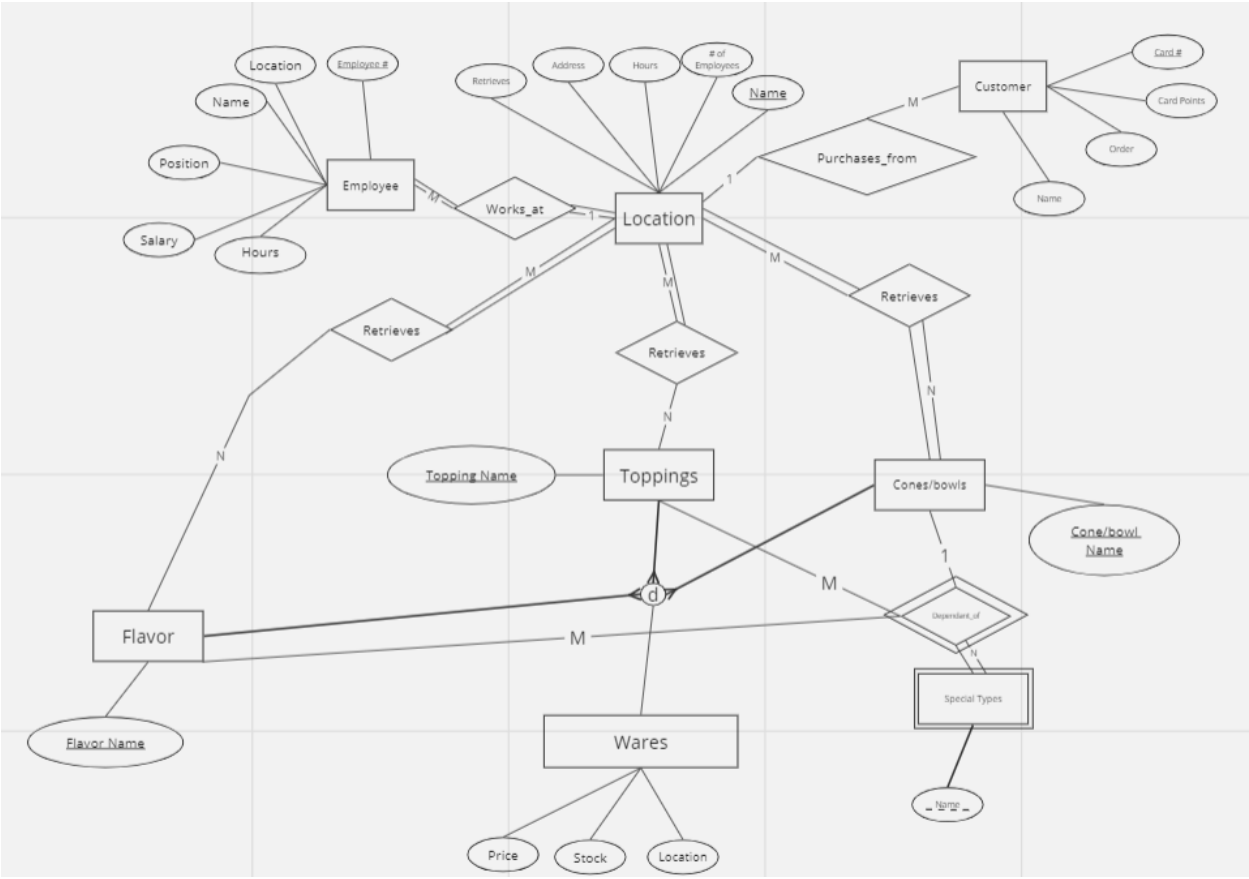
A1

In this assignment we talked about the general description of our project, and had the initial promises of "having one order purchase multiple products" and "Special Types" as a relation.

Revisions/Comments:

We ended up changing the database in the end to no longer include "Special types" but instead optimized our database more, with the interactions between locations, orders, and the three products (Toppings,Cones/Bowls,Flavors).

A2 (ER DIAGRAM)



Revisions/Comments:

Mentioned changes inside FINAL ER DIAGRAM:

- Special types and wares removed and placed into attributes for flavor,toppings and cones/bowls instead
- Orders table implemented to account for the products which the customer ordered (1:N Relationship)
- Customer relation changed to orders instead of to location because loc_id will be taken from order (1:N relationship)
- Flavors,cones/bowls/toppings now have location as an additional primary key to their pre-existing primary key xxxxName
- All relations between flavors,toppings,cones/bowls have become 1:N relations from M:N

A3

In this assignment we took inspiration from the ER diagram and made the source code in ORACLE. After the creation we used some insertion code to fill up some of our tables, here is an example:

1) Toppings

```
INSERT INTO TOPPINGS (name,price,stock,loc_id) VALUES ('sprinkles', 1.00, 5,1);
```

	NAME	PRICE	STOCK	LOC_ID
1	sprinkles	1	5	1

Revisions/Comments:

It was at this point that we realized some of our ER diagrams were off, and had to make some adjustments based on this. We removed the “Special Types” table that was from A2 and replaced it with

A4

In this assignment, we referenced the schema laid out in our SQL tables to create 8 simple queries, 2 views, and 4 advanced queries. The distinguishing implementations and intentions of these three tasks can be seen in the following three examples that represent one of each type.

The following query was used in the implementation of a view as it allowed for the sanitization of information in the employee table that disclosed more personal information than required about each employee as the query restricts the information to be only the employee name and number of those that work at location1.

```
/* Create a view that lists all employee names and employee numbers in ascending order
that work for the location of ID 1*/

CREATE VIEW Loc_1_employees AS
        (SELECT e.Name, employee_number
        FROM employee e, Location l
        WHERE (l.Loc_ID = 1) AND (e.location = l.Loc_ID)
        ORDER BY employee_number ASC);
```

The following simple query like all of our simple queries doesn't involve complex statements and thus allows for convenience in retrieving simple information from a specified table which is the intention of all of our simple queries. In this case, the following simple query data is present in just the location table to return all records of locations that work during the day with 10 or more employees.

```
/* List all locations by names and IDs in ascending order that are operational during the
day and have 10 or more employees */

SELECT Name AS 'Day Time Locations', Loc_ID
FROM Location
WHERE (Hours = 'Day') AND (Hires >= 10)
ORDER BY Loc_ID ASC;
```

In the case of advanced queries like the one in the following table, they focus on joining multiple tables together as the intended records can only be retrieved by uncovering related data between tables. In this case, the advanced query uses the aggregate function COUNT alongside the joining of the customer table with the orders table to display only the records in the customer table whose card numbers correspond to only a single record in the orders table by relating the appearance of the card number in the customer table to its appearance in the orders table.

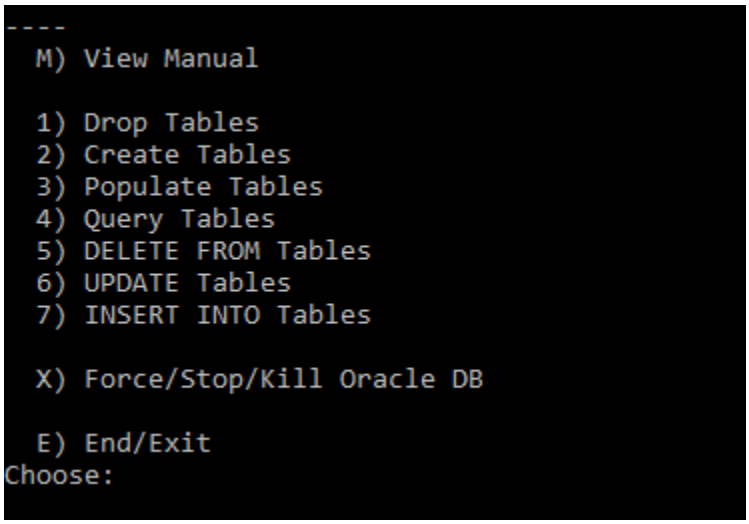
```
/* List all Customer names and card numbers of those who have placed exactly one order
*/

SELECT c.Name, c.CardNum, COUNT(o.OrderID) AS OrderCount
FROM Customer c, Orders o
WHERE (c.CardNum = o.CardNum)
GROUP BY c.Name, c.CardNum
HAVING COUNT(o.OrderID) = 1;
```

Revisions/Comments:

There were no revisions made to our views and queries because the initial tables that the views and queries referenced didn't require decomposition to achieve normalization in the 3NF and BCNF, allowing all of our queries and views to remain consistent with our table design before and after.

A5



M) View Manual
1) Drop Tables
2) Create Tables
3) Populate Tables
4) Query Tables
5) DELETE FROM Tables
6) UPDATE Tables
7) INSERT INTO Tables
X) Force/Stop/Kill Oracle DB
E) End/Exit
Choose:

```
SQL> SQL>
no rows selected

SQL> SQL>
NAME                          PRICE      STOCK     LOC_ID
------------------------- ---------- ---------- ----------
vanilla flavor                    3         10         10
chocolate flavor                  3          4         10
matcha flavor                     5          9         10
strawberry flavor                 4          1         10
mango flavor                      5         13         10
Vanilla Flavor                  4.5         15          1
Chocolate Flavor                4.5         12          1
Strawberry Flavor               4.5         10          1

8 rows selected.

SQL> SQL>
NAME                          PRICE      STOCK     LOC_ID
------------------------- ---------- ---------- ----------
pointed cone                      2          4         13
small bowl                        4          9         13
medium bowl                       6          9         13
large bowl                        9          9         13
waffle cone                       2          9         12
sugar cone                        2          5         12
cake cone                         1         11         12
pretzel cone                      4          7         12
vegan cone                        6         14         12
Small Cone                        3         20          1
Large Cone                        4         18          1

NAME                          PRICE      STOCK     LOC_ID
------------------------- ---------- ---------- ----------
Regular Bowl                      5         25          1

12 rows selected.

SQL> SQL>
NAME                          PRICE      STOCK     LOC_ID
------------------------- ---------- ---------- ----------
sprinkles                         1          5         11
cherry                            6          5         11
syrup                             4          5         11
marshmallows                      3          2         11
```

Revisions/Comments:

This is the structure used for our final UI, and allowed the smooth implementation of inserts, updates, and deletes,

A6-8

When determining all FDs in all of our tables we had to determine real-world facts surrounding the attributes in each of our tables to ensure that the logic of our FDs was intact. Doing so we concluded that the only FDs present in our tables contained a key on the LHS and because all of our tables only had a single candidate key it meant that the key on the LHS is the same for all FDs. Therefore, it is made possible to consolidate all FDs in every individual table into a single FD per table.

Location Table FD:
{loc_ID} → Name, Hours, Hires

Employee Table FD:
{employee_number} → Name, Salary, Hours, Position, location

Flavor, Toppings, and Cones_And_Bowls Table FD:
{Name, Loc_ID) → Price, Stock

Customer Table FD:
{Cardnum} → Cardpoints, Name

Orders Table FD:
{OrderID} → CardNum, Quantity, Loc_ID, ToppingName, FlavorName, ConeOrBowlName

Given that the FDs in all tables have a key on the LHS that results in all of our tables being normalized in the BCNF as BCNF requires that to be the case. Also, with our tables being in BCNF it means that 3NF has also been achieved.

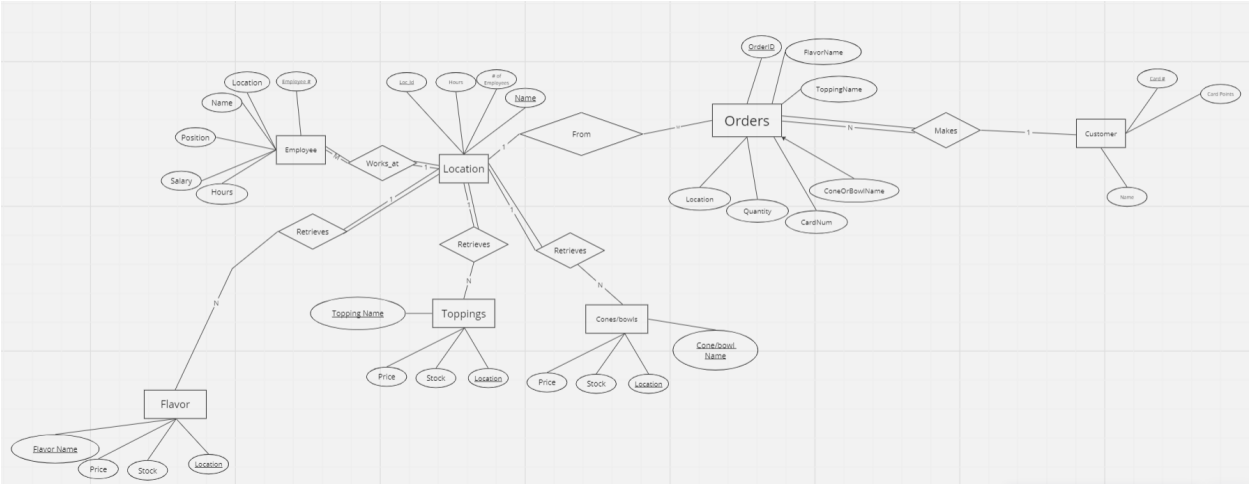
Revisions/Comments:

N/A

A9

Please refer to “FINAL UI” section below.

FINAL ER Diagram:



- Special types and wares removed and placed into attributes for flavor,toppings and cones/bowls instead
- Orders table implemented to account for the products which the customer ordered (1:N Relationship)
- Customer relation changed to orders instead of to location because loc_id will be taken from order (1:N relationship)
- Flavors,cones/bowls/toppings now have location as an additional primary key to their pre-existing primary key xxxxName
- All relations between flavors,toppings,cones/bowls have become 1:N relations from M:N

FINAL UI:

INSTRUCTIONS:
Download the files (16 total) onto moon.ssh
to run, execute menu.sh (you may need to give execute permissions)

SCREENSHOTS:

```
------------------------------------------------------------
----
  M) View Manual

  1) Drop Tables
  2) Create Tables
  3) Populate Tables
  4) Query Tables
  5) DELETE FROM Tables
  6) UPDATE Tables
  7) INSERT INTO Tables

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
```

4-7 Have distinct menus upon selection, whereas 1-3 have 1 button responses. Here are menus from 4-7 with completion examples:

4) Query

```
  E) End/Exit
Choose:
4
 ------------------------------------------------------------
 0) Show all queries
 1) Query 1: Employee names + numbers of loc 1
 2) Query 2: Show all flavours that are low on stock
 3) Query 3: Select customers who have not ordered, OR only ordered from 'dundas'
 4) Query 4: Select all items from location 1
 5) Query 5: Select customers with only 1 order
 6) Query 6: Select  customers with multiple orders
 7) Custom Query
Enter your choice (1-6): 7
Please write the customized query statement
SELECT * FROM Customer

SQL*Plus: Release 12.1.0.2.0 Production on Sat Nov 25 21:03:49 2023

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>   2
   CARDNUM CARDPOINTS NAME
---------- ---------- ------------------------
      6415        100 John
         1        800 Brandon
         2        300 Danny
       123          0 Jason

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Press Enter to continue...
```

5) DELETE FROM Tables

```
Choose:
5
Tables= (Employee, Customer, Flavors, Toppings, Cones_And_Bowls, Orders
Enter the table name: Toppings
enter a CONDITION FORMAT: att1 = val1
stock > 5

SQL*Plus: Release 12.1.0.2.0 Production on Sat Nov 25 21:05:06 2023

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
4 rows deleted.

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Press enter to continue...
```

6)UPDATE Tables

```
Choose:
6
----Tables= (Employee, Customer, Flavors, Toppings, Cones_And_Bowls, Orders
Enter the table name: Employee
----Enter a CONDITION FORMAT: att1 = val1
Name = 'John'
----Enter a SET FORMAT: column1 = value1, column2=value2,...
name = 'Kyle'

SQL*Plus: Release 12.1.0.2.0 Production on Sat Nov 25 21:05:54 2023

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>   2    3
1 row updated.

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Press enter to continue...
```

7) INSERT INTO Tables

```
Choose:
7
Tables= (Employee, Customer, Flavors, Toppings, Cones_And_Bowls, Orders
Enter the table name: Cones_And_Bowls
INSERT STATEMENT FORMAT = (val1,val2,val3,...)
Enter the SQL INSERT statement: ('pointed cone', 2.00, 4, 13)

SQL*Plus: Release 12.1.0.2.0 Production on Sat Nov 25 21:07:57 2023

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
1 row created.

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Press enter to continue...
```

RA QUERY NOTATION:

/* List all locations by names and IDs in ascending order that are operational during the day and have 10 or more employees */

SELECT Name AS ‘Day Time Locations’, Loc_ID
FROM Location
WHERE (Hours = ‘Day’) AND (Hires >= 10)
ORDER BY Loc_ID ASC;

$\rho_{\text{Location(Day Time Locations, Loc\_ID)}}(\prod_{\text{Name, Loc\_ID}}(\sigma_{\text{Hours = ‘Day’ AND Hires} \geq 10}\ \text{(Location)}))$

/* List the stock of every available flavor with respect to the flavor name in ascending order of stock */

SELECT name, SUM(stock) as TotalStock
FROM flavors
WHERE (Stock > 5)
GROUP BY name
ORDER BY TotalStock ASC;

$\rho_{flavors(name,\ TotalStock)}(_{name}\ F_{SUM\ stock}(\sigma_{Stock\ >\ 5}\ (flavors))))$

/* List name and price of toppings that cost at least $4.99 in descending order of price */

```
SELECT name, price
FROM toppings
WHERE (price >= 4.99)
ORDER BY price DESC;
```

$(\prod_{name,\ price}(\sigma_{price\ \geq\ 4.99}\ (toppings))))$

/* List of Ice_Cream_Types by name and price that have a topping and are served in any sized bowl that are sorted in descending order by price. */

```
SELECT Name, Price
FROM Ice_Cream_types
WHERE (Cone_or_bowl == ‘small bowl’ OR Cone_or_bowl == ‘medium bowl’  OR
Cone_or_bowl == ‘large bowl’ ) AND ( Topping IS NOT NULL)
ORDER BY Price DESC;
```

$(\prod_{Name,\ Price}(\sigma_{(Cone\_or\_bowl\ =\ ‘small\ bowl’\ OR\ Cone\_or\_bowl\ =\ ‘medium\ bowl’\ \ OR\ Cone\_or\_bowl\ =\ ‘large\ bowl’\ )\ AND\ (\ Topping\ \neq\ NULL)}\ (Ice\_Cream\_types))))$

/* List all Icecream orders by OrderID and name that occur in batches of 10 or more and are Sundaes */

```
SELECT OrderID, ItemName
FROM Orders
Where (Quantity >= 10) AND (ItemName LIKE ‘%Sundae%’);
```

$(\prod_{OrderID,\ ItemName}(\sigma_{(Quantity\ \geq\ 10)\ AND\ (ItemName\ LIKE\ ‘\%Sundae\%’)}(Orders))))$

/* Counts the amount of employees inside each location */

```
SELECT  location, COUNT(employee_number) As num_employees
FROM Employee
GROUP BY location
ORDER BY location;
```

$\rho_{Employee(location,\ num\_employees)}({}_{location}\mathcal{F}_{COUNT\ employee\_number}(Employee))$

/* Grabs the CustomerOrders that were made by the customer with CardNum 13*/

```
SELECT *
FROM CustomerOrders
WHERE CardNum = 13
ORDER BY CustomerOrderID;
```

$\sigma_{CardNum = 13}(CustomerOrders)$

/* Grabs customers with cardpoints above 100, ordered by cardpoints descending*/

```
SELECT *
FROM Customers c
WHERE c.cardpoints > 100
ORDER BY c.cardpoints DESC;
```

$\sigma_{cardpoints > 100}(Customers)$

/* List Customers who have either NOT ORDERED anything or has ordered from location with name ‘dundas’ */

```
SELECT c.CardNum, c.Name
FROM Customer c, Orders o
WHERE NOT EXISTS (
        SELECT 1
        From Orders
        WHERE Orders.CardNum = c.CardNum
)
UNION
SELECT c.CardNum,c.Name
FROM Customer c
WHERE EXISTS
(       SELECT 1
        FROM Orders
        WHERE O.CardNum = C.CardNum
        AND EXISTS (
                SELECT 1
                FROM Location L
```

```
                    WHERE O.Loc_ID = L.Loc_ID
                    AND L.Name = 'dundas'
                )
        );
```

$(\prod_{CardNum,Name}(Customers - (Customers \bowtie Orders))) \cup (\prod_{CardNum,Name}(\sigma_{Location.Name = \text{'dundas'}}(Customers \bowtie Orders \bowtie Location)))$

/* Selects all the toppings, cones, bowls that belong to Location with ID = 1*/

```
SELECT * FROM
(
   SELECT * FROM Toppings t
   WHERE t.Loc_ID = 1
UNION
   SELECT  * FROM Flavors f
   WHERE f.Loc_ID = 1
UNION
   SELECT * FROM Cones_And_Bowls c
   WHERE c.Loc_ID = 1
)
ORDER BY stock ASC;
```

$(\sigma_{Loc\_ID = 1}(toppings)) \cup (\sigma_{Loc\_ID = 1}(Flavors)) \cup (\sigma_{Loc\_ID = 1}(Cones\_And\_Bowls))$

/* List all Customer names and card numbers of those who have placed exactly one order */

```
SELECT c.Name, c.CardNum, COUNT(o.OrderID) AS OrderCount
FROM Customer c, Orders o
WHERE (c.CardNum = o.CardNum)
GROUP BY c.Name, c.CardNum
HAVING COUNT(o.OrderID) = 1;
```

$(\prod_{Name,CardNum,} F_{COUNT\ orderID}(\sigma\ F_{Count\ OrderID = 1}(Customers \bowtie_{Customers.Cardnum=Orders.CardNum} Orders)))$

/* List all Customer names and card numbers of those who have placed more than one order */

```
SELECT c.Name, c.CardNum, COUNT(o.OrderID) AS OrderCount
FROM Customer c, Orders o
WHERE (c.CardNum = o.CardNum)
GROUP BY c.Name, c.CardNum
HAVING COUNT(o.OrderID) > 1;
```

$$(\prod_{Name,CardNum,} F_{\ COUNT\ orderID}\ (\sigma\ F_{\ Count\ OrderID\ >\ 1}\ (Customers\ \bowtie_{Customers.Cardnum=Orders.CardNum}\ Orders)))$$