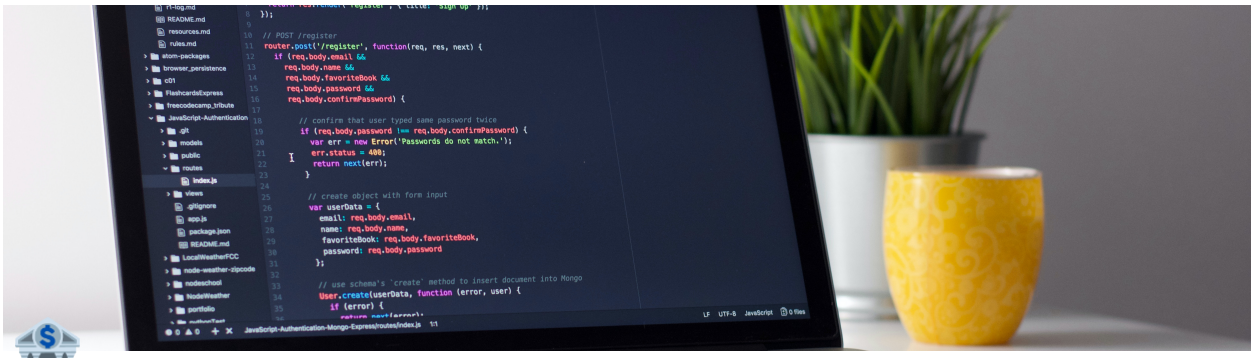


DBMS Deadline-6

Indian Dairy Authority(IDA)

💡 Shlok Vinodkumar Mehroliya 2021421 || Sumeet Mehra 2021428

Project Scope:

In this project we are designing and developing an e2e db application for the dairy company IDA(Indian Dairy Authority)which will be used for various purposes including but not limited to inventory management,serializing and categorizing products and managing the company sales dynamically while also giving them notifications if a product is going out of stock,is out of stock or is being procured by customers at a rapid pace as the interface is being used.We hope to create a experience through which both the user gets a smooth ride through the website and one where the company is able to manage their products with ease.
‘

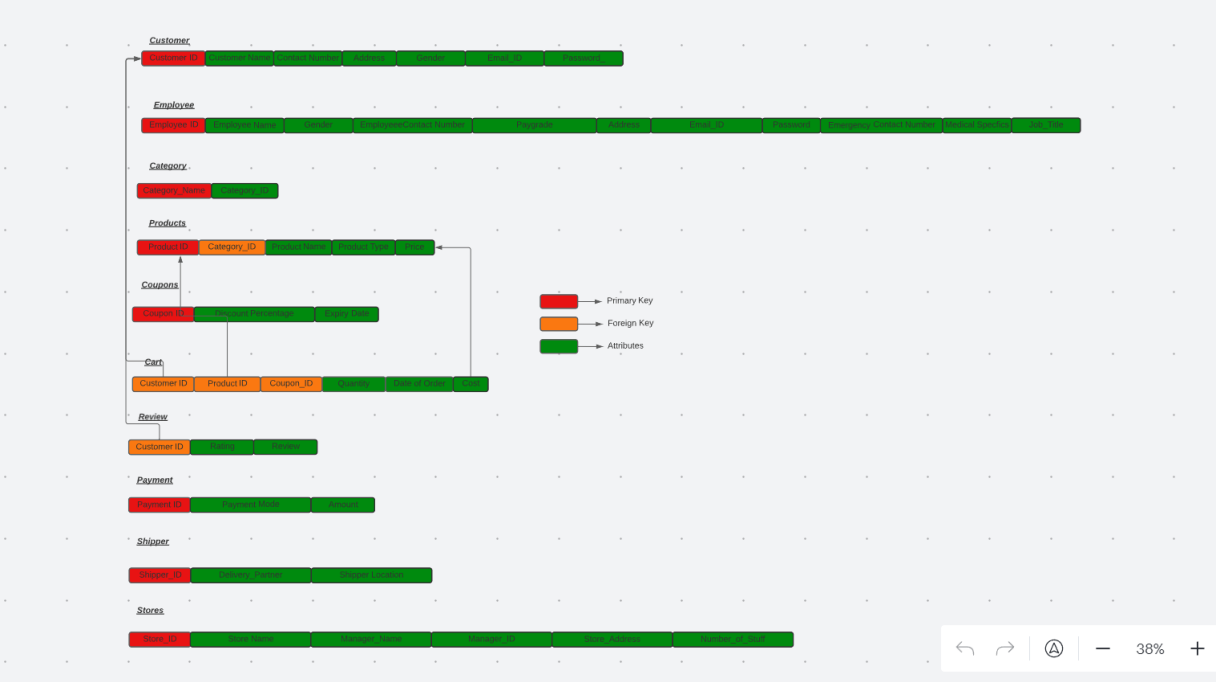
TECHSTACK:

For the front-end , we plan on using the following:

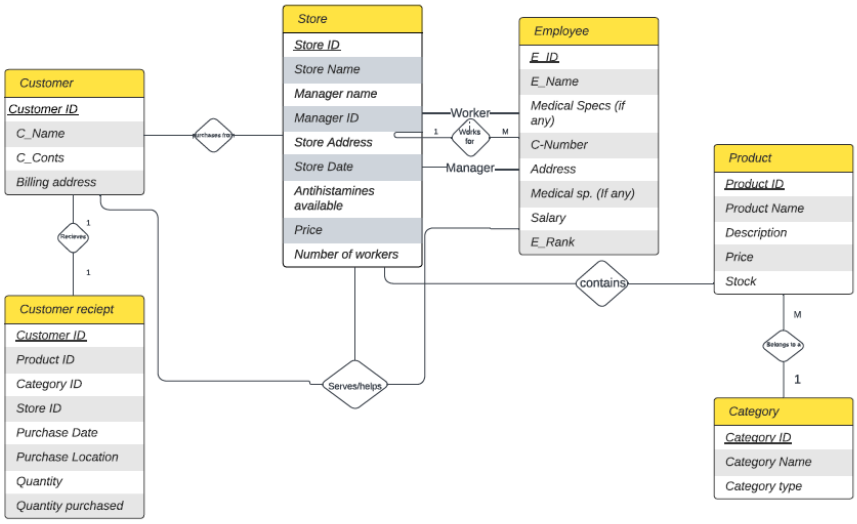
1. MySQL*(Managing Database)*
2. Python*(CLI)*

We need to ensure payment gateway integration, product inventory management, shopping cart and checkout functionality, order tracking and management, delivery management, and raw materials supply to facilitate the smooth running of the factory and website.

Relational Schema



ER Diagram



Database Schema:

```
CREATE TABLE CUSTOMER_F (
C_ID int(11) NOT NULL,
C_Name varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Contact_Num varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Address varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
PRIMARY KEY (C_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
CREATE TABLE Customer_Reciept (
C_ID int(11) NOT NULL,
Payment_type varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Payment_ID int(11) NOT NULL,
Prod_id int(11) NOT NULL,
Store_ID int(11) NOT NULL,
Quantity int(11) NOT NULL,
PRIMARY KEY (C_ID),
KEY Prod_id (Prod_id),
KEY Store_ID (Store_ID),
CONSTRAINT Customer_Reciept_ibfk_1 FOREIGN KEY (Prod_id) REFERENCES Products (Prod_ID),
CONSTRAINT Customer_Reciept_ibfk_2 FOREIGN KEY (Store_ID) REFERENCES Store (S_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
CREATE TABLE Employee (
E_ID int(20) NOT NULL,
E_Name varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Contact_Num varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Address varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Salary varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
PRIMARY KEY (E_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
CREATE TABLE Payment_Portal (
Payment_ID int(11) NOT NULL,
Payment_Type varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Total_invoice varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Payment_Status varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
PRIMARY KEY (Payment_ID),
CONSTRAINT Payment_Portal_ibfk_1 FOREIGN KEY (Payment_ID) REFERENCES Customer_Reciept (C_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
CREATE TABLE Products (
Prod_ID int(21) NOT NULL,
Prod_Name varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Prod_Desc varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Prod_price varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Prod_Stock varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
PRIMARY KEY (Prod_ID),
CHECK (Prod_ID > 0),
CHECK (Prod_price > 0)) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
CREATE TABLE Store (
S_ID int(21) NOT NULL ,
Manager_name varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
M_ID int(11) NOT NULL,
Address varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
Num_Workers int(11) NOT NULL,
PRIMARY KEY (S_ID),
CHECK (S_ID >= 0)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

SQL Queries:

```
#Query1
select * from Store where Num_Workers>=100 and Num_Workers<150 and S_ID>150;
```

```
#Query2
SELECT Payment_Status, COUNT(*), Payment_Type FROM Payment_Portal GROUP BY Payment_Status,Payment_Type;
```

```
#Query3
SELECT Payment_Type, COUNT(*) FROM Payment_Portal GROUP BY Payment_Type;
```

```
#Query4
SELECT Payment_Portal.Payment_ID, Payment_Portal.Payment_Type, Customer_Reciept.Store_ID,Customer_Reciept.Quantity
FROM Payment_Portal INNER JOIN Customer_Reciept ON Payment_Portal.Payment_ID = Customer_Reciept.C_ID;
```

```
#Query5
SELECT Payment_Portal.Payment_ID, Customer_Reciept.C_ID,Customer_Reciept.Prod_id,Customer_Reciept.Payment_type
FROM Customer_Reciept
LEFT OUTER JOIN Payment_Portal
ON Payment_Portal.Payment_ID =Customer_Reciept.C_ID;
```

```
#Query6
UPDATE Products
```

```
SET Prod_price = 65
WHERE Prod_ID = 1703;
```

```
#Query7
select * from employee union select * from Store;
```

```
#Query8
select sum(Prod_price),Prod_Name as Products_Sale from products group by Prod_Name;
```

```
#Query9
SELECT C_Name, TotalAmt
FROM Customer_F
JOIN
(SELECT C_ID, SUM(Quantity * Prod_price) AS TotalAmt
FROM Customer_Reciept
JOIN Products ON Customer_Reciept.Prod_ID = Products.Prod_ID
WHERE Payment_Type = 'UPI'
GROUP BY C_ID)
AS CustomerTotal ON Customer_F.C_ID = CustomerTotal.C_ID;
```

```
#Query10
SELECT C_Name, Contact_Num, Prod_name
FROM Customer_F
JOIN
(SELECT Customer_Reciept.C_ID, Products.Prod_id, Prod_name, Prod_desc
FROM Products
JOIN Customer_Reciept ON Products.Prod_id = Customer_Reciept.Prod_id
WHERE Prod_stock >= 100) AS High_req_customer ON Customer_F.C_ID = High_req_customer.C_ID;
```

Explanation of queries:

1. Selects all columns from the Store table where the number of workers is between 100 and 149, and the store ID is greater than 150.
2. Counts the number of records for each distinct combination of Payment_Status and Payment_Type from the Payment_Portal table.
3. Counts the number of records for each distinct Payment_Type from the Payment_Portal table.
4. Selects columns from the Payment_Portal and Customer_Reciept tables where the Payment_ID in Payment_Portal matches the C_ID in Customer_Reciept.

5. Selects columns from the Customer_Reciept table, with an additional column for Payment_ID from Payment_Portal that matches the C_ID in Customer_Reciept, using a LEFT OUTER JOIN.
   6: Updates the price of a product with a specific ID to $65 in the "Products" table.
   7: Combines all the rows from "employee" and "Store" tables into a single result set.
   8: Calculates the sum of the "Prod_price" for each distinct "Prod_Name" and displays the results as "Products_Sale" in the "products" table.
   #9: This SQL command selects the customer name and the total amount spent by each customer on products purchased through UPI payment method by joining the Customer_F and Customer_Reciept tables, and grouping the results by customer ID.
   #10: This SQL command selects the customer name, contact number, and product name for all customers who have made purchases of products with a stock quantity of at least 100, by joining the Customer_F and Customer_Reciept tables with the Products table.

Embedded Queries:

```
# import pymysql
import mysql.connector

# Connect to the database
i = mysql.connector.connect(host='localhost', user='root', port="3306", password='Sumeethld*13', db='shlok')

while True:
    print("Select the query you want to run : \n Press 1 for - Query to retrieve all columns from the Store table where no. of workers is g
    x = int(input())
    if(x==1):

        #Embedded Query 1

        # create a cursor object
        cursor = i.cursor()

        # execute the embedded SQL query
        cursor.execute("SELECT * FROM Store WHERE Num_Workers >= %s AND Num_Workers < %s AND S_ID > %s", (100, 150, 150))

        # fetch the query results
        res = cursor.fetchall()

        # print the results
        for row in res:
            print(row)

        # close the database connection
        i.close()

    elif(x==2):
        #Embedded Query 2

        # create a cursor object
        cursor = i.cursor()

        # execute the embedded SQL query
        cursor.execute("SELECT * FROM Store WHERE Num_Workers >= %s AND Num_Workers < %s AND S_ID > %s", (100, 150, 150))

        # fetch the query results
        res = cursor.fetchall()

        # print the results
        for row in res:
            print(row)

        # close the database connection
        i.close()

    else:
        print("enter only 1 or 2, try again!!!")
```

OLAP Queries:

```
#Query 1
select Prod_Name,Prod_Stock from Products where Prod_Stock>50 group by Prod_Name,Prod_Stock;
```

```
#Query 2 - group the products by name and then calculate the total stock and average price for each product
SELECT Prod_Name, SUM(Prod_Stock) AS Total_Stock, AVG(Prod_Price) AS AvgPrice
FROM Products
GROUP BY Prod_Name;
```

```
#Query 3 - this one will show the total number of unique customers, contact numbers, and addresses associated with each name
SELECT
C_Name,
COUNT(DISTINCT C_ID) AS Num_Customers,
COUNT(DISTINCT Contact_Num) AS Num_Contacts,
COUNT(DISTINCT Address) AS Num_Addresses
FROM
Customer_F
GROUP BY
C_Name;
```

```
#Query 4- this one shows the number of unique employees in the system, as well as the average, minimum, and maximum salaries for all employ
SELECT
COUNT(DISTINCT E_ID) AS Num_Employees,
AVG(Salary) AS Avg_Salary,
MIN(Salary) AS Min_Salary,
MAX(Salary) AS Max_Salary
FROM
Employee;
```

Triggers:

```
#Trigger 1
DELIMITER $$
CREATE TRIGGER update_product_stock
AFTER INSERT ON Customer_Reciept
FOR EACH ROW
BEGIN
IF NEW.Payment_type = 'UPI' THEN
UPDATE Products
SET Prod_Stock = Prod_Stock + NEW.Quantity
WHERE Prod_ID = NEW.Prod_ID;
ELSEIF NEW.Payment_type = 'CASH' THEN
UPDATE Products
SET Prod_Stock = Prod_Stock - NEW.Quantity
WHERE Prod_ID = NEW.Prod_ID;
END IF;
END $$
```

```
DELIMITER ;
#This trigger updates stock of the product by adding NEW.quantity value to the existing stock
```

```
***#Trigger 2***
DELIMITER $$CREATE TRIGGER calculate_cost
AFTER INSERT ON Customer_Reciept
FOR EACH ROW
BEGIN
DECLARE purchase_total DECIMAL(10,2);
SELECT SUM(Prod_price * Quantity) INTO purchase_total
FROM Products
WHERE Prod_ID = NEW.Prod_ID;

INSERT INTO Payment_Portal (C_ID, Purchase_Total)
VALUES (NEW.C_ID, purchase_total, NOW());

END $$
DELIMITER ;

/*THIS TRIGGER CALCULATES THE TOTAL COST OF CUSTOMER PURCHASE AND CREATES A NEW COLUMN IN THE CUSTOMER_RECIUEPT TABLE CONSISTING OF THE SAM
```

User Guide:

Our project is a solution for a dairy vendor trying to keep track of daily purchases occurring in his shop while simultaneously keeping a close eye on his inventory. It uses various tables shown below in our relational schema and ER diagrams and also links all tables together to ensure a smooth operation of the shop.Using the Command Line Input(CLI) we have incorporated an easy to access and use UI.

The various functionalities of our CLI use:

1. Insert a product:
    - Prompt the user to enter the values for the new entry (product ID, product name, product type, product price, and product stock).
    - Insert the new entry into the Products table.
2. Update price of all products by reducing them by 50%:
    - Update the Products table by setting the product price to half of its current value if the product stock is greater than zero.
3. Update payment status of a specific payment type as paid in Payment_Portal:
    - Prompt the user to enter the payment type for which the payment status needs to be updated.
    - Update the Payment_Portal table by setting the payment status to 'PAID' for the specified payment type.
4. Delete a store with less than a specific number of workers:
    - Prompt the user to enter the minimum number of workers for a store to be deleted.
    - Delete all rows from the Store table where the number of workers is less than the specified number.