**CRAZY KITCHEN**

**USE CASE STUDY REPORT**

**Group No**: Group 08

**Student Names**: Rohan Reddy Pathi and Yuandi Tang

Introduction:

Crazy Kitchen is a popular restaurant known for its diverse and innovative cuisine. As the restaurant continues to grow, the need for an efficient and organized data management system becomes increasingly important. Currently, the restaurant faces challenges in managing reservations, inventory, customer orders, and employee data. A well-structured database can address these challenges by centralizing data, streamlining operations, and improving decision-making processes. This project aims to design and implement a comprehensive database system tailored to the specific needs of Crazy Kitchen, enhancing its operational efficiency and customer service.

Problem Statement:

Crazy Kitchen's existing data management approach relies heavily on manual processes and disparate data storage systems, leading to inefficiencies, errors, and delays. These challenges include:

* Inefficient Reservation Management: The current system lacks a centralized way to manage reservations, leading to overbooking or double booking.
* Inventory Control Issues: Inadequate tracking of inventory levels results in either surplus or shortage of ingredients, affecting the kitchen's efficiency and profitability.
* Inconsistent Order Processing: The absence of an integrated order management system leads to confusion and delays in processing customer orders.

To address these issues, Crazy Kitchen needs a robust database solution that can integrate all aspects of its operations into a single, cohesive system.

Objectives of Project:

The primary objective of this project is to design and implement a fully functional and efficient database system for Crazy Kitchen. The specific goals include:

* Centralized Data Management: Create a unified database that consolidates all operational data, including reservations, inventory, orders, and employee schedules.
* Reservation System: Develop a module to efficiently manage table reservations, preventing overbooking and ensuring optimal seating arrangements.
* Inventory Management: Implement an inventory tracking system to monitor stock levels, reduce waste, and streamline ordering processes.
* Order Processing System: Design an order management system that accurately records and tracks customer orders from placement to fulfillment.
* Scalability and Security: Ensure the database is scalable to accommodate future growth and includes robust security measures to protect sensitive information.

By achieving these objectives, the project aims to enhance Crazy Kitchen's operational efficiency, improve customer satisfaction, and support the restaurant's continued growth and success.

# Conceptual Data Modeling

## EER Diagram

A diagram of a diagram

Description automatically generated with medium confidence

## Entity Descriptions:

* **Customer (Member/Non-member):** Manages customer details and their membership status, differentiating between members and non-members.

**Attributes:**

* + Customer ID (Key)
  + Name
  + Email
  + Phone
  + Membership Status
* **Staff:** Tracks employee information and their hierarchical relationships within the organization.

**Attributes**:

* + Staff ID (Key)
  + Name
  + Email
  + Phone
  + Wage
  + Manager ID
  + Department
  + Hire Date
* **Order:** Handles the processing of orders.

**Attributes:**

* + Order ID (Key)
  + Order Date
* **Supplier:** maintains information about suppliers who provide ingredients and other supplies.

**Attributes:**

* + Supplier ID (Key)
  + Name
  + Phone
* **Table:** Manages the availability and capacity of tables in the restaurant.

**Attributes:**

* + Table Number (Key)
  + Capacity
* **Dish:** Tracks menu items in the restaurant.

**Attributes:**

* + Dish Name (Key)
  + Description
  + Price
* **Ingredient:** Monitors the consumption and stock of ingredients used in dishes.

**Attributes:**

* + Ingredient ID (Key)
  + Ingredient Name
  + Expiry Date
* **Receipt:** Records sales transactions.

**Attributes:**

* + Receipt ID (Key)
  + Amount
  + Sale Date
  + Payment Method

## Entity Relationships

* **Customer *Places* Order:** One-to-Many (A customer can place multiple orders, but each order is linked to one customer)
* **Order *Allocates at* Table:** One-to-Many (A table can take zero to multiple orders, but each order is allocated at a table)
* **Staff *Handles* Order:** One-to-Many (A staff member can handle multiple orders, but each order is linked to one staff member)
* **Dish *Uses* Ingredient:** Many-to-Many (A dish can use multiple ingredients, and each ingredient can be used in multiple dishes.)

**Attributes:**

* + Quantity
* **Order *composes* Dish:** Many-to-Many (An order must have one or more dishes, but a dish can be multiple or not be in any order)

**Attributes:**

* + Quantity
* **Order *Generates* Receipt:** One-to-One (Each order generates a receipt, vice versa)
* **Supplier *Supplies* Ingredients:** Many-to-Many (A supplier can provide multiple ingredients, and each ingredient can link to different supplier)

**Attributes:**

* + Quantity
* **Staff *Manages* Staff (Manager):** One-to-Many (A staff member can manage multiple staff, but each staff member is managed by one manager)

**UML Diagram:**

A diagram of a computer

Description automatically generated with medium confidence

**Mapping Conceptual Model to Relational Model**

The One-One relationships and One-Many relationships are easily mapped from EER to Relational model by adding a Foreign Key in one of the tables. But to handle Many-Many relationships, we have to create a new Relation containing a composite key which consists of the Primary Keys of both the involved tables. The Attributes also act as Foreign Keys which refer to their respective relations.

* **OrderedDish Table:**

*In the conceptual model, there's an association between Order and Dish entities, indicating that orders can include multiple dishes and each dish can be part of multiple orders. This is a classic many-to-many relationship, which cannot be directly represented in a relational model using only two tables without violating normalization principles. The OrderedDish table is created to resolve this many-to-many relationship. It acts as a bridge table that Contains foreign keys referencing the primary keys of the Order and Dish tables.*

* **IngredientsUsed Table:**

*The conceptual model shows that a Dish can be composed of multiple Ingredients, and an Ingredient can be used in multiple Dishes. This is another many-to-many relationship, which needs to be addressed in the relational model. The IngredientsUsed table is introduced to include the Quantity of each ingredient used in a dish, providing essential information for recipe management or inventory tracking.*

* **IngredientsSupplied Table:**

*The relationship between Supplier and Ingredient entities indicates that suppliers can supply multiple ingredients, and each ingredient can be sourced from multiple suppliers—a many-to-many relationship. The IngredientsSupplied table captures the Quantity of each ingredient supplied by each supplier, which could be useful for procurement and inventory control.*

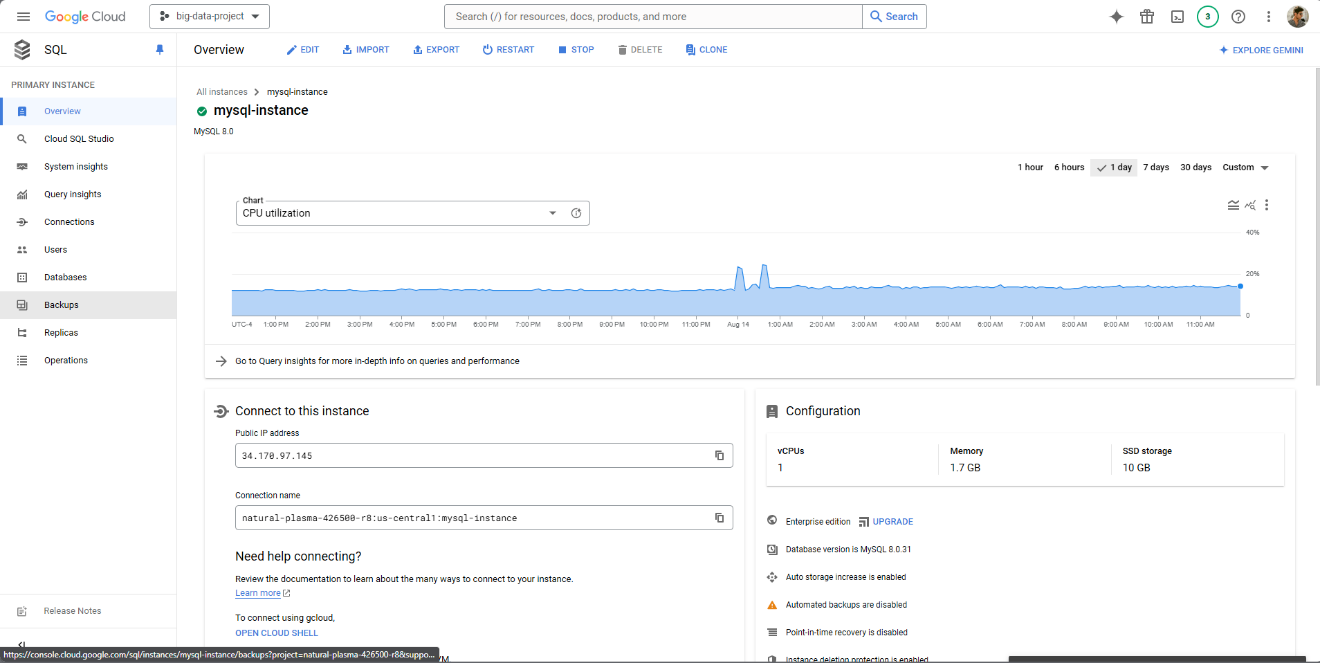
**Relational Model in BCNF:**

* Customer (Customer ID, Name, Email, Phone, MemberID)
  + FOREIGN KEY MemberID refers to MemberID in Customer; NULL NOT ALLOWED
* Member (MemberID, Discount, JoinDate, ExpiryDate)
* Staff (StaffID, Name, Email, Phone, Wage, ManagerID, Department, HireDate)
  + FOREIGN KEY ManagerID refers to StaffID in Staff; NULL NOT ALLOWED
* Order (OrderID, OrderDate, CustomerID, StaffID, TableNumber)
  + FOREIGN KEY CustomerID refers to CustomerID in Customer; NULL NOT ALLOWED
  + FOREIGN KEY StaffID refers to StaffID in Staff; NULL NOT ALLOWED
  + FOREIGN KEY TableNumber refers to TableNumber in Table; NULL ALLOWED
* OrderedDish (OrderID, DishName, Quantity)
  + FOREIGN KEY OrderID refers to OrderID in Order; NULL NOT ALLOWED
  + FOREIGN KEY DishName refers to DishName in Dish; NULL NOT ALLOWED
* Supplier (Supplier ID, Name, Phone)
* Table (TableNumber, Capacity)
* Dish (DishName, Description, Price)
* Ingredient (IngredientID, IngredientName)
* IngredientsUsed (*IngredientID*, *DishName*, Quantity)
  + FOREIGN KEY IngredientID refers to IngredientID in Ingredient; NULL NOT ALLOWED
  + FOREIGN KEY DishName refers to DishName in Dish; NULL NOT ALLOWED
* IngredientsSupplied (*IngredientID*, *SupplierID*, Quantity)
  + FOREIGN KEY IngredientID refers to IngredientID in Ingredient; NULL NOT ALLOWED
  + FOREIGN KEY SupplierID refers to SupplierID in Dish; NULL NOT ALLOWED
* Receipt (ReceiptID, Amount, SaleDate, PaymentMethod, *OrderID*)

# IV. Implementation of Relation Model via MySQL and NoSQL

**Step 1: Deployment on Google Cloud**

Deploying a restaurant management system on Google Cloud, from setting up the infrastructure to configuring databases, ensuring security, and making the application accessible to users.



**STEP 2: Creating DB**

The first and foremost step would be to create the tables in the database. We shall use the DDL CREATE command to create the tables and enforce constraints.

**STEP 3: Data Generation**

After creating the tables, it is time to fill the tables with data. Since we do not have real world data, we can fill in data using random and fake data. In order to fill the tables using fake data, we can use the Faker module in python. The python code for the fake data creation can be found here: <https://github.com/tantalum-73/crazykitchen-db/blob/main/py-scripts/ck-data-insertion.ipynb>.

**STEP 4: Analytical Queries**

In this step, we outline a few analytical queries designed to extract meaningful insights from the Crazy Kitchen Restaurant Database. These queries are crafted to analyze various aspects of the data with the goal of supporting data-driven decision-making.

1. Top Selling Dishes

Purpose: This query helps identify which dishes are the most popular among customers.

SELECT DishName, SUM(Quantity) AS TotalQuantity

FROM OrderedDish

GROUP BY DishName

ORDER BY TotalQuantity DESC

LIMIT 5;

Output:

A screenshot of a computer

Description automatically generated

1. Average Sales by day of week

Purpose: This query provides insights into daily revenue trends and can help in understanding peak sales days of a week.

SELECT DAYNAME(SaleDateTime) AS SaleDay, AVG(Amount) AS TotalSales

FROM Receipt

GROUP BY SaleDay

ORDER BY SaleDay;

Output:

A screenshot of a data table

Description automatically generated

1. Ingredient Usage

Purpose: This query shows the total quantity of each ingredient used across all dishes. It helps in managing inventory and understanding ingredient demand.

SELECT i.IngredientName, SUM(iu.Quantity) AS TotalUsed

FROM IngredientsUsed iu

JOIN Ingredient i ON iu.IngredientID = i.IngredientID

GROUP BY i.IngredientName

ORDER BY TotalUsed DESC;

Output:

A screenshot of a menu

Description automatically generated

1. Sales performance of staff with respect to department

Purpose: This query calculates the sales performance of each staff member based on their department. This provides insights into individual staff contributions relative to their department. This is helpful while performing employee performance reviews.

WITH StaffSales AS (

SELECT s.StaffID, s.Name AS StaffName, s.Department,

SUM(r.Amount) AS TotalSales

FROM Receipt r

JOIN RestaurantOrder o ON r.OrderID = o.OrderID

JOIN Staff s ON o.StaffID = s.StaffID

GROUP BY s.StaffID

),

DepartmentSales AS (

SELECT Department, SUM(TotalSales) AS DepartmentTotalSales

FROM StaffSales

GROUP BY Department

)

SELECT ss.staffID, ss.StaffName, ss.Department, ss.TotalSales, ds.DepartmentTotalSales,

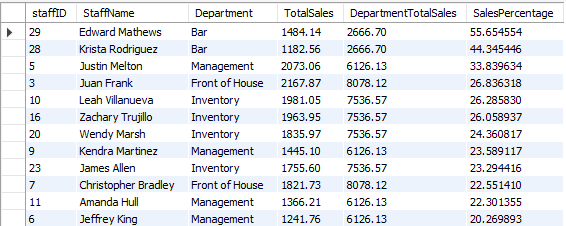
(ss.TotalSales / ds.DepartmentTotalSales) \* 100 AS SalesPercentage

FROM StaffSales ss

JOIN DepartmentSales ds ON ss.Department = ds.Department

ORDER BY SalesPercentage DESC;

Output:



1. Dish Popularity by time of the day

Purpose: This query provides the popularity of dishes by the hour of the day. It helps us understand which dishes are more popular at specific times of the day. This can guide us in menu planning.

SELECT HOUR(o.OrderDateTime) AS OrderHour,

d.DishName,

SUM(od.Quantity) AS TotalQuantity

FROM OrderedDish od

JOIN `RestaurantOrder` o ON od.OrderID = o.OrderID

JOIN Dish d ON od.DishName = d.DishName

GROUP BY OrderHour, d.DishName

HAVING OrderHour >= 10 AND OrderHour <= 22

ORDER BY OrderHour, TotalQuantity DESC;

Output:

*Note: I made a mistake while inserting fake data by not enforcing constraints on the order time. This resulted in OrderHour starting from 0 Hours and ending at 23 Hours.*

A screenshot of a menu

Description automatically generated

1. Identify Customers Who Have Not Ordered Anything in the Last 7 Days

Purpose: This query Identifies customers who have been inactive for a while, which can be used for targeted marketing campaigns.

SELECT C.CustomerID, C.Name AS CustomerName,

MAX(O.OrderDatetime) AS LastOrderDate

FROM Customer C

LEFT JOIN RestaurantOrder O ON C.CustomerID = O.CustomerID

GROUP BY C.CustomerID, C.Name

HAVING MAX(O.OrderDatetime) < CURDATE() - INTERVAL 7 DAY

ORDER BY LastOrderDate;

Output:

A screenshot of a computer

Description automatically generated

1. Customer Segmentation Based on Spend and Frequency

Purpose: This query segments customers into categories based on their total spending and order frequency. It helps in targeting marketing efforts and tailoring customer engagement strategies.

SELECT c.CustomerID, c.Name,

CASE

WHEN TotalSpent > 500 AND TotalOrders > 10 THEN 'High Value Frequent'

WHEN TotalSpent > 500 THEN 'High Value'

WHEN TotalOrders > 10 THEN 'Frequent'

ELSE 'Occasional'

END AS CustomerSegment

FROM Customer c JOIN (

SELECT CustomerID,

SUM(r.Amount) AS TotalSpent,

COUNT(o.OrderID) AS TotalOrders

FROM Receipt r

JOIN RestaurantOrder o ON r.OrderID = o.OrderID

GROUP BY CustomerID

) AS CustomerSummary ON c.CustomerID = CustomerSummary.CustomerID;

Output:

A screenshot of a computer

Description automatically generated

1. Staff Efficiency based on number of orders handled

Purpose: This query gives information on the efficiency of staff by the number of orders they handled. It offers insights into staff productivity and engagement, highlighting their performance in managing customer orders. Understanding this can help measuring employee effectiveness and morale which is crucial for workplace management.

SELECT s.StaffID,

s.Name AS StaffName,

COUNT(o.OrderID) AS OrdersHandled,

DENSE\_RANK() OVER (ORDER BY COUNT(o.OrderID) DESC) AS StaffRank

FROM RestaurantOrder o

JOIN Staff s ON o.StaffID = s.StaffID

GROUP BY s.StaffID, s.Name

ORDER BY OrdersHandled DESC;

Output:

A screenshot of a table

Description automatically generated

**STEP 5: Migrate Data from MySQL to MongoDB**

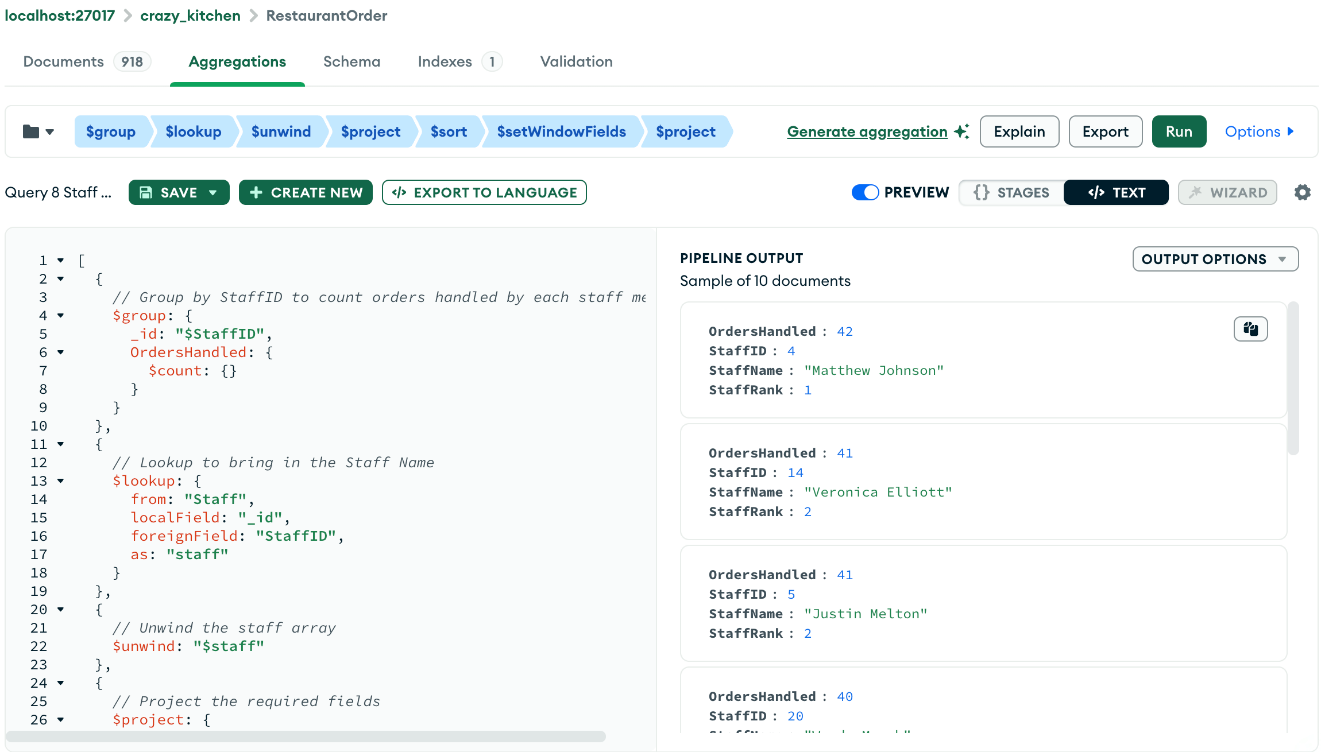
To migrate data from MySQL to MongoDB, we first export tables from MySQL into CSV format. Then we use a python script to import data from the CSV files into the MongoDB database. Each table in RDBMS becomes a collection in MongoDB.

The python script for importing CSV files into MongoDB is attached here: <https://github.com/tantalum-73/crazykitchen-db/blob/main/py-scripts/ck-mongodb-migration.ipynb>.

**STEP 6: MongoDB Aggregation and Queries**

MongoDB’s aggregation framework provides powerful tools for processing and analyzing data stored in collections. Here’s a guide on how to leverage MongoDB aggregation and queries to analyze data from the Crazy Kitchen database.

A sample query is shown below due to the constraint on length, the full code and queriers can be found here: <https://github.com/tantalum-73/crazykitchen-db/blob/main/py-scripts/ck-data-nosql-mongodb-queries.ipynb>.



# V. Database Access via R or Python

To access the MySQL database from Python, we can use the mysql.connector package to establish a connection, Then we use cursors to execute queries on the database for analytics. The code for establishing connection to the MySQL database and perform analytics can be found here:

The python script for connecting to the database and generating analytics is attached here: <https://github.com/tantalum-73/crazykitchen-db/blob/main/py-scripts/ck-data-visualization.ipynb>.

An example visualization which provides the Weekly sales trend can be shown below. The visualization can be used to gain insight on the overall performance and patterns in sales overtime.

A graph with a line

Description automatically generated

# VII. Summary and recommendation

The Crazy Kitchen database project successfully addressed critical data management challenges by centralizing data across reservations, inventory, and order processing. This unified approach has improved data accuracy, streamlined operations, and enhanced decision-making with valuable insights from analytical queries. The system's scalability and flexibility, especially with MongoDB integration, position the restaurant well for future growth and adaptability.

However, the project encountered some shortcomings, including issues with initial data constraints, the complexity of implementation, potential performance overhead, and challenges in data synchronization. These issues may impact the accuracy and efficiency of the system if not properly addressed.

To optimize the database system, recommendations include correcting data constraints, providing staff training, monitoring system performance, and establishing robust data synchronization procedures. Continuous improvement and user feedback will be crucial in maintaining the system’s effectiveness and supporting Crazy Kitchen’s ongoing success.