SRM E-Java Food Point

A PROJECT REPORT

Submitted by

M.ASEENA SULTHANA [RA2211030010317]

K.CHARANYA [RA2211030010304]

M.D.PRARTHANA [RA2212703010020]

Under the Guidance of

Dr. Preethiya T

Assistant Professor, Department of Networking and Communications

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

With specialization in cyber security

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603203

BONAFIDE CERTIFICATE

Register no. RA2212703010020, RA2211030010317,RA2211030010304 Certified to be the bonafide work done by **M Aseena Sulthana, K.Charanya, M.Prarthana** of II year/IV semester B.Tech Degree Course in the Project Course – **21CSC205P Database Management Systems** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur for the academic year 2023-2024.

Date: 03/04/2024

FACULTY INCHARGE
Dr. Preethiya T
Assistant Professor
Department of Networking
and Communications

HEAD OF THE DEPARTMENT
DR.Annapurani paniyappan.K
Professor & Head
Department of Networking
and Communications

ABSTARCT

This report outlines the conceptualization, development, and implementation of an Online Canteen System (OCS) tailored for the dynamic needs of modern university campuses. With the proliferation of digital technologies and the growing demand for convenience, traditional cafeteria models are increasingly being challenged to adapt. The OCS serves as a comprehensive solution to revolutionize campus dining experiences, offering students, faculty, and staff a seamless and efficient way to access a diverse range of food options.

Key features of the OCS include a user-friendly web interface and mobile application, enabling stakeholders to browse menus, place orders, and make payments remotely. By leveraging real-time data analytics and machine learning algorithms, the system optimizes menu recommendations, predicts demand patterns, and enhances inventory management, thereby minimizing food wastage and maximizing operational effecting.

The successful deployment of the Online Canteen System underscores its potential to transform campus dining experiences, fostering a more convenient, accessible, and inclusive environment for all members of the university community. Future endeavors will focus on scaling the system across additional campuses, integrating new features and functionalities, and fostering partnerships with external food vendors to further enrich the dining ecosystem.

In conclusion, developing and optimizing an online food ordering system requires a holistic approach that focuses on user experience, technical excellence, and strategic innovation. By implementing these strategies, businesses in the food industry can gain a competitive edge, expand their online presence, and capitalize on the vast opportunities offered by the digital marketplace.

TABLE OF CONTENTS

PROBLEM STATEMENT:

Within the confines of a bustling campus, dining experiences are often plagued by a series of challenges unique to the university environment. Crowded canteens become hotbeds of congestion during peak hours, leading to interminable queues that test the patience of students, faculty, and staff alike. Accessibility to dining facilities can also become an issue, with certain corners of the campus feeling distanced from the central canteens, resulting in inconvenience for those located farther away. Moreover, the limited variety of menu options fails to accommodate the diverse dietary preferences prevalent among the campus population, leaving many feeling underserved and dissatisfied. Inefficient service processes further exacerbate the situation, as manual order-taking and food preparation contribute to delays and errors, disrupting the dining experience. Additionally, the environmental impact of campus dining operations cannot be overlooked, with excessive food wastage and packaging waste contributing to sustainability concerns.

Chapter-1

Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the project

1.1 Problem understanding

The popularity of ordering food online has made it super convenient to get a meal without leaving home. But, let's be real, it's not always smooth sailing. Sometimes, finding the right platform to order from can be a bit of a headache. You might find yourself stuck with limited choices, high prices, or a website that's just plain hard to use.

Here's the deal: we all want different things when we order food online. Sure, we want it to be easy, but we also want options that suit our tastes and wallets. Unfortunately, not all online food ordering platforms deliver on these fronts. Some just don't have enough restaurants to choose from, others charge way too much, and some websites feel like a maze to navigate.

The issue lies in the mismatch between customer expectations and platform offerings. While convenience is paramount, customers also crave variety, affordability, and simplicity. Unfortunately, many existing platforms fall short on these fronts, failing to provide the diverse culinary options, reasonable prices, and intuitive interfaces that users desire.

Overall, by addressing these key challenges and focusing on providing a comprehensive solution that offers a wide range of restaurant and cuisine options at competitive prices, the new online food ordering platform can attract a diverse range of customers and become a go-to destination for online food ordering.

1.2 Identification of Entity and Relationships

The authentication mechanism ensures secure user access, while user profiles enable tailored interactions based on personal details and food preferences. Restaurants are core entities, detailing cuisine, menus, and locations. Orders capture items, quantities, and delivery preferences, integrating with payment and delivery services. Feedback entities allow users to review and rate experiences, fostering improvement. These interconnected entities form the foundation for a seamless and personalized online food ordering platform, enhancing user satisfaction and engagement.

Login entity:

The login entity is essentially the backbone of a system's security mechanism, playing a pivotal role in ensuring that only authorized users gain access to the system's features and resources. At its core, it comprises several essential elements that work seamlessly together to facilitate user authentication and access control. First and foremost, the login entity encompasses the user interface (UI) through which users interact with the system to log in. This UI typically includes fields for entering usernames or email addresses and passwords, along with options for actions like password recovery or account registration.

User entity:

This information includes personal details, such as name, contact information, and address, which are essential for processing orders and deliveries. Additionally, the user profile entity can store preferences, such as favorite cuisines, dietary restrictions, and preferred payment methods, to personalize the user experience. Order history is also typically stored in the user profile entity, allowing users to easily reorder their favorite meals and enabling the platform to provide tailored recommendations based on past orders. Overall, the user profile entity plays a crucial role in enhancing user satisfaction and driving engagement on the platform

Menu entity:

In the e-canteen system, the shop entity represents the virtual storefront where users browse and select food items for purchase. Within the shop entity, users can explore a diverse range of menu options, including breakfast, lunch, snacks, and beverages. Each item within the shop entity is meticulously categorized and described, providing users with essential details such as name, description, price, and availability. Additionally, the shop entity may feature promotional offers, seasonal specials, and recommended items to enhance user engagement and encourage exploration. By offering a visually appealing and user-friendly interface, the shop entity serves as the primary point of interaction between users and the e-canteen system, facilitating seamless browsing and selection of food items.

Bill entity:

The bill entity contains essential billing information, including details about the user or customer, such as their name, address, and contact information. These details are crucial for accurately invoicing the user and ensuring proper communication regarding billing matters. Moreover, the bill entity stores comprehensive transaction details, encompassing information about the products or services purchased, including item descriptions, quantities, prices, and total amounts due.

This allows for precise billing and reconciliation of transactions, ensuring transparency and accuracy in financial records Tracking the payment status of each transaction is another vital function of the bill entity. It indicates whether a bill has been paid in full, partially paid, or remains outstanding, enabling efficient monitoring of payment collection and follow-up on overdue invoices. Additionally, the bill entity includes information about accepted payment methods, enabling users to choose their preferred mode of payment during the checkout process. It also facilitates the generation and delivery of invoices, either

electronically or in physical format, based on user preferences and business requirements.

Items entity:

This includes details such as the name, description, category, and price of each item. These attributes provide users with essential information about the products available for purchase and help them make informed decisions. The items entity tracks the quantity of each item available in stock to ensure accurate inventory management. This information is crucial for preventing stockouts and managing replenishment orders effectively. For items with variations such as size, color, or configuration options, the items entity may include attributes and variants to represent different product variations. This allows users to select their preferred options when making a purchase.

Order entity:

Order entity in an online food ordering system represents a specific order placed by a user. It typically includes attributes such as order ID (a unique identifier for each order), user ID (identifying the user who placed the order), restaurant ID (identifying the restaurant from which the order was placed), total price (the total cost of the order), status (indicating the current status of the order, e.g., pending, confirmed, delivered), and timestamp (the date and time when the order was placed). The order entity is essential for tracking and managing orders, allowing users to view their order history and restaurants to manage incoming orders effectively.

Admin entity:

In database management systems, entities are represented as tables, and attributes are the columns within these tables. Each attribute describes a specific aspect of the entity being modeled. For example, in a "Person" entity, attributes could include "Name," "Age," "Address," and "Email."

Admin entities often have authority over specific areas or functions and play a crucial role in ensuring that things run smoothly and efficiently.

Payment entity:

A unique identifier for each payment transaction. This is essential for tracking and referencing specific payments. The monetary value of the payment. This attribute specifies how much money was transferred or paid. The date and time when the payment transaction occurred. This helps in tracking the timing of payments and can be useful for auditing and reconciliation purposes.

Indicates the method used for the payment, such as credit card, debit card, bank transfer, cash, or digital wallet.

Represents the current status of the payment transaction, such as "pending," "completed," "failed," or "refunded." This attribute provides insights into the progress of the payment process.

Sender and Receiver Information: Information about the entities involved in the transaction, including the sender (payer) and receiver (payee). This might include details like names, account numbers, or contact information.

Reference or Invoice Number: A unique identifier associated with the payment, often used to link the payment to a specific invoice or reference within the system.

Offers entity:

A unique identifier for each offer. This allows for easy referencing and tracking of specific offers within the system.

A descriptive name or title for the offer, helping users identify and understand the promotion. Additional information describing the details, terms, and conditions of the offer. This might include eligibility criteria, expiration date, usage limitations, and any required actions to redeem the offer.Specifies the type and amount of discount or benefit provided by the offer. This could be a percentage discount, a fixed amount off the purchase price, free items, or other incentives. The date range during which the offer is valid or active. This helps ensure that offers are only available for a specific period and can be automatically

activated or deactivated based on these dates. Defines the segment of customers or users to whom the offer is applicable. This could be based on demographics, purchase history, loyalty status, or other criteria.

Redemption Code: A unique code associated with the offer that users can use to redeem the promotion. This could be a coupon code, promo code, or barcode.

Applicable Products or Services: Specifies the products, services, or categories to which the offer applies. This ensures that the offer is only valid for specific items or transactions. Optionally, attributes such as maximum usage per customer, total usage limit, or restrictions on the number of times an offer can be redeemed may be included to manage offer availability and prevent abuse.

Checkout entity:

A unique identifier for each checkout transaction. This allows for easy tracking and reference of specific orders within the system. Details about the customer making the purchase, including name, contact information, shipping address, and billing address. Information about the items being purchased, including product IDs, names, quantities, prices, and any variations (e.g., size, color).The total price of the order, including the sum of the prices of all items, taxes, shipping costs, and any applicable discounts or promotions. Details about the payment method used to complete the transaction, such as credit card information, billing address, payment gateway used, and transaction status. The chosen shipping method for delivering the order, including options such as standard shipping, expedited shipping, or store pickup. Indicates the current status of the order within the checkout process, such as "pending," "processing," "shipped," "delivered," or "cancelled. "Timestamps indicating key events in the checkout process, such as when the order was placed, when payment was processed, and when the order was shipped or delivered. Optional notes or comments provided by the customer during the checkout process, such as special instructions for delivery or gift messages.: If applicable, any promotion codes or coupons applied to the order, along with details about the discounts they provide.

Information related to order fulfil ment, such as tracking numbers, shipping carrier information, and estimated delivery dates

Payment entity:

In the e-canteen system, the payment entity facilitates the secure and efficient processing of financial transactions between users and the system. Users interact with the payment entity to complete their orders by selecting their preferred payment method and providing necessary payment details. This entity encompasses various payment methods such as credit/debit cards, digital wallets, or prepaid accounts, ensuring flexibility and convenience for users. Each payment transaction within the entity is securely processed and recorded, including details such as transaction ID, amount, payment method, and timestamp. Additionally, the payment entity may incorporate features such as encryption, authentication, and fraud detection to safeguard user information and prevent unauthorized access. By facilitating seamless and secure payment processing, the payment entity enhances the overall user experience within the e-canteen system.

Upi entity:

In the context of an e-canteen system, the UPI (Unified Payments Interface) entity represents a popular payment method that allows users to make seamless and instant transactions directly from their bank accounts. Users interact with the UPI entity during the checkout process to select UPI as their preferred payment option. Upon selecting UPI, users are typically redirected to their preferred UPI-enabled app or platform where they can securely authenticate the transaction using their UPI ID, PIN, or biometric authentication. The UPI entity facilitates real-time communication between the e-canteen system and the user's bank, enabling swift and secure fund transfers without the need for additional intermediaries. By offering UPI as a payment option, the e-canteen system enhances convenience and flexibility for users, allowing them to complete transactions quickly and efficiently.

Net banking entity:

In the e-canteen system, the net banking entity serves as a secure and convenient payment option that allows users to make transactions directly from their bank accounts via internet banking portals. During the checkout process, users can select net banking as their preferred payment method and choose their bank from a list of available options. Upon selection, users are redirected to their bank's internet banking portal where they can securely authenticate the transaction using their login credentials or other authentication methods provided by the bank. The net banking entity facilitates seamless communication between the e-canteen system and the user's bank, ensuring that transactions are processed swiftly and securely. By offering net banking as a payment option, the e-canteen system enhances flexibility and accessibility for users, allowing them to make payments conveniently from anywhere with an internet connection.

Delivery entity:

In the e-canteen system, the delivery entity handles the process of transporting ordered food items from the canteen or kitchen to the designated location specified by the user. Users may opt for delivery during the checkout process, indicating their preferred delivery address and any specific instructions. Once the order is placed, the delivery entity coordinates with delivery personnel or third-party logistics providers to ensure timely and accurate delivery. This may involve assigning delivery drivers, optimizing delivery routes, and providing real-time tracking updates to users. The delivery entity plays a crucial role in fulfilling user expectations for prompt and reliable delivery, enhancing the overall convenience and satisfaction of the e-canteen system.

1.3 Construction of DB using ER Model for the project

To construct a e-canteen_database architecture for online food ordering system, we begin by delineating the entities and their attributes, thereby establishing a comprehensive foundation for data management and retrieval.

1. User:

This entity encapsulates essential demographic information about users, including their email address, name, date of birth, age, gender, phone number, and address. Each user is uniquely identified by their email address, serving as the primary key.

2. Login:

You use it to put in your username and password, and if they match what's stored in the system, it lets you access your account. It's also in charge of keeping your account safe, so it might ask you for extra security info sometimes. Think of it as your digital bouncer, making sure only the right people get into the party.

3. Bill:

The bill entity is kind of like your digital receipt—it keeps track of what you've bought and how much you owe. It stores details about your purchases, like the items you've bought, how much they cost, and whether you've paid for them yet. It also helps the system manage things like discounts and payment methods, making sure everything adds up correctly. So, think of it as your personal accountant, keeping tabs on your spending and making sure everything's squared away.

4. Items entity:

The items entity is like a digital catalog of everything available for purchase within the system. It holds all the details about products or services, such as their names, descriptions, and prices. It also keeps track of things like how many items are in stock and any variations, like different colors or sizes. Essentially, it's the system's virtual storefront, showcasing everything you can buy and helping you find what you're looking for.

5. Shop entity:

The shop entity in an online food ordering system represents the various restaurants available for users to order food from. It typically includes attributes such as restaurant ID (a unique identifier for each restaurant), name (the name of the restaurant), cuisine type (the type of cuisine offered by the restaurant, e.g., Italian, Chinese, Indian), address (the address of the restaurant), contact number (the phone number of the restaurant), and rating (the average rating of the restaurant based on customer reviews). The restaurant entity is essential for users to browse and select restaurants, view their menus, and place orders. It also allows restaurants to manage their information, including updating their menus and contact details. Overall, the restaurant entity plays a crucial role in facilitating the ordering process and providing users with a variety of dining options to choose from.

6. Order entity:

The order entity in an online food ordering system represents a specific order placed by a user. It typically includes attributes such as order ID (a unique identifier for each order), user ID (identifying the user who placed the order), restaurant ID (identifying the restaurant from which the order was placed), total price (the total cost of the order), status (indicating the current status of the order, e.g., pending, confirmed, delivered), and timestamp (the date and time when the order was placed). The order entity is essential for tracking and managing orders, allowing users to view their order history and restaurants to manage incoming orders effectively**.**

7. Admin entity:

Firstly, administrators use their username and password and it contains(Admin_id) to securely log in to the system, gaining access to administrative functions. Once logged in, they can define permissions and roles for themselves and other users, determining who can perform specific actions within the system.

8. Offers entity:

First and foremost, administrators utilize the offers entity to define the parameters of each promotion, including details such as the offer name, description, validity period, and any associated terms and conditions. These attributes provide clarity and transparency to users, ensuring they understand the benefits and limitations of each offer.Additionally, the offers entity incorporates attributes related to discount types and values, allowing administrators to specify whether offers entail percentage discounts, fixed amount discounts, or other promotional incentives. This flexibility enables administrators to tailor promotions to suit different marketing strategies and target audience preferences.

9. Payment entity:

The payment entity within the system acts as a vital component for managing financial transactions securely and efficiently. At its core, it comprises attributes that facilitate the processing, tracking, and reconciliation of payments, ensuring a seamless and reliable payment experience for users.Firstly, the payment entity includes attributes related to payment methods, allowing users to choose from various options such as credit/debit cards, digital wallets, bank transfers, or cash on delivery. Each payment method is associated with specific parameters, such as transaction fees, processing times, and security measures, providing users with flexibility and convenience in completing their transactions.

10. Checkout entity:

Firstly, the checkout entity includes attributes related to the selected items or services, providing a summary of the user's shopping cart contents, including product details, quantities, and prices. This allows users to review their selections before proceeding to payment, ensuring accuracy and completeness in their orders .Additionally, the checkout entity incorporates attributes for capturing user information necessary for order fulfil ment, such as shipping addresses, contact details, and preferred delivery options. This information enables the system to

process orders accurately and deliver products or services to the specified location in a timely manner.

11. Delivery entity:

The delivery entity is like the conductor of a symphony, orchestrating the smooth flow of orders from the system to the users' doorsteps. It includes details about the items being delivered and any special instructions from users. Additionally, it keeps track of delivery addresses and contact info, making sure delivery folks can find users easily and communicate if needed. Users get real-time updates on their orders' whereabouts and estimated arrival times, helping them plan their day.The delivery entity also plans the most efficient routes for deliveries, taking into account factors like traffic and delivery priorities to minimize wait times. It manages delivery personnel assignments, making sure the right people are in the right place at the right time. Lastly, it handles delivery documentation, like digital signatures and confirmations, providing users with proof of delivery and ensuring transparency in the process. Ultimately, it's all about getting orders to users quickly, smoothly, and with a smile.

12. Upi entity:

UPI (Unified Payments Interface) entity is like the digital wallet of the system, facilitating seamless and instant transactions between users and merchants. It encompasses attributes that streamline the payment process, ensuring secure and efficient fund transfers.Firstly, the UPI entity includes attributes related to user accounts and payment details, allowing users to link their bank accounts or virtual payment addresses (VPAs) to the system. This information enables users to initiate transactions directly from their UPI-enabled accounts, without the need for additional login credentials**.**

13. Net banking entiny:

The net banking entity is like the digital branch of a bank within the system, enabling users to perform various banking transactions conveniently and

securely. It encompasses attributes that facilitate online banking services, allowing users to manage their accounts, transfer funds, and perform other financial activities.Firstly, the net banking entity includes attributes related to user accounts and authentication, allowing users to log in securely using their credentials, such as usernames and passwords. This ensures that only authorized users can access their accounts and perform banking transactions.

Additionally, the net banking entity incorporates attributes for account management, enabling users to view their account balances, transaction history, and other account details online. This provides users with real-time access to their financial information, empowering them to monitor their accounts and make informed financial decisions.

ENTITY RELATIONSHIP DIAGRAM

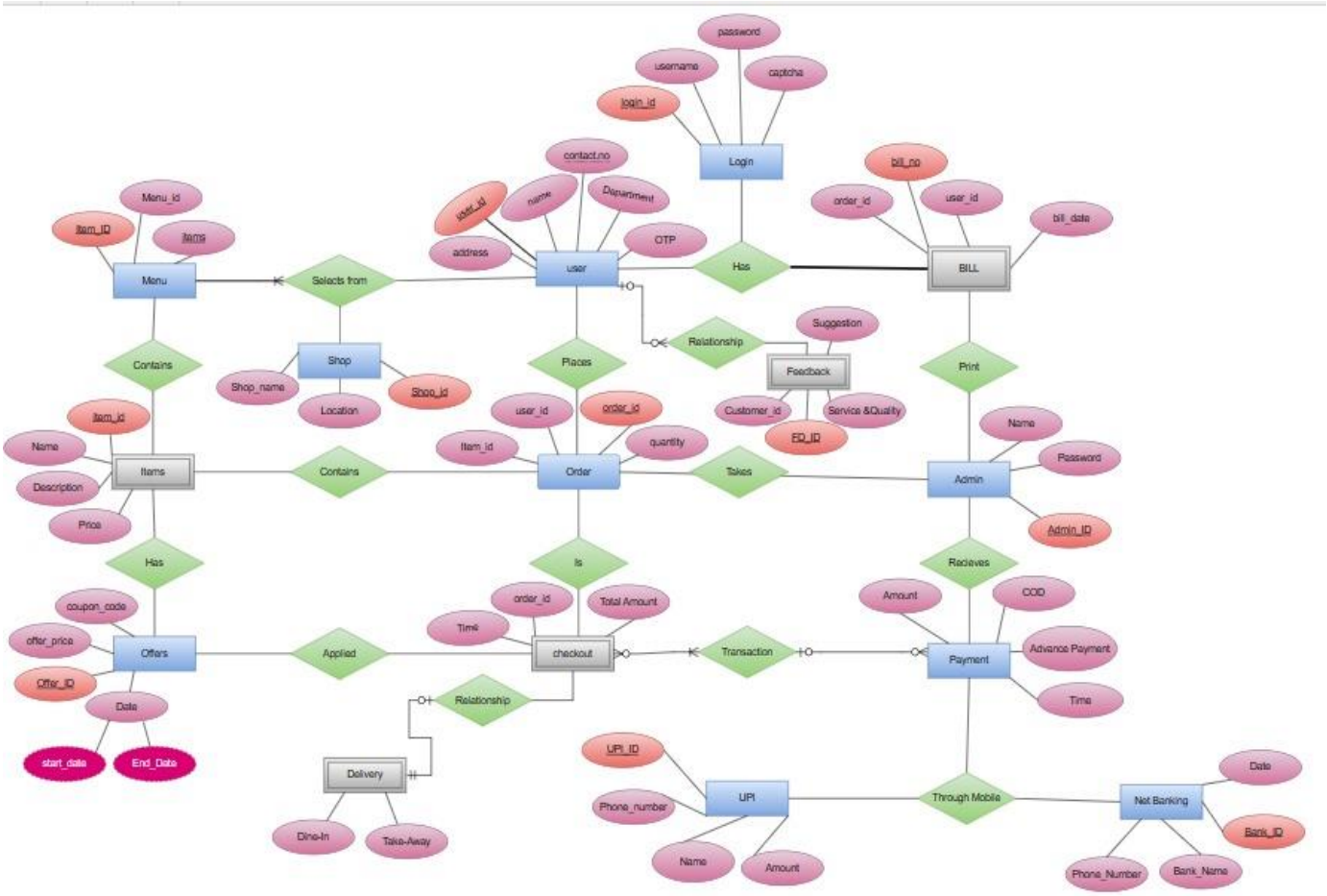


Figure1.1:ER Diagram

Chapter-2

Design of Relational Schemas, Creation of Database Tables for the project

2.1 Design of Relational Schemas

The relational schema design for the online food ordering platform is structured to efficiently manage data related to users, restaurants, menus, orders, and deliveries. The Users table serves as the central repository for user information, featuring attributes such as user ID, username, email, phone number, and address, crucial for personalized interactions and order processing. Complementing this is the Login table, enabling secure authentication with attributes like login ID, user ID (referencing the Users table), username, and password.

Shop is assigned a unique Shop_id, serving as the primary key of the table. This identifier distinguishes one restaurant from another within the system. The table also includes attributes that provide descriptive information about each restaurant. These attributes typically encompass details such as the restaurant's name, address, contact information (such as phone number), and operational hours. Additionally, the table may include attributes for location-related information, such as latitude and longitude coordinates, to enable location-based search functionalities.

Furthermore, the shop table may incorporate attributes related to the restaurant's cuisine type or specialty. This information helps users identify restaurants that offer specific types of cuisine or dishes, catering to their preferences and tastes

Orders and Order Items tables manage order information, with the Orders table capturing details such as order ID, user ID (referencing the Users table), restaurant ID (referencing the Restaurants table), total price, status, and timestamp, ensuring comprehensive order tracking. The Order Items table stores details of items included in each order, with attributes like order item ID, order ID (referencing the Orders table), item ID (referencing the Menu Items table), quantity, and subtotal, facilitating accurate order processing and fulfilment

Table contains attributes that capture details about the delivery itself. This may include information such as the delivery address, delivery status, and estimated delivery time. The delivery address attribute specifies the location where the order is to be delivered, while the delivery status attribute indicates the current status of the delivery (e.g., pending, in transit, delivered). The estimated delivery time attribute provides users with an estimated timeframe for when they can expect their order to arrive, enhancing transparency and managing expectations.

RELATIONAL SCHEMA:

- **User** (`id', 'username', 'department', 'address', 'password', 'contact number')
- **Login** (`login_id', 'captcha', 'username', 'password')
- **Shop** ('shop name', 'shop_id', 'location')
- **Items** (`item_id', 'name', 'description', 'price')
- **Order** (`user_id', 'order_id', 'quantity', 'item_id')
- **Feedback** ('suggestion', 'customer_id', 'service&quality', 'fd_id')
- **Admin** ('name', 'password', 'admin_id')
- **Bill** ('bill no', 'user_id', 'order_id', 'bill date')
- **Offers** ('coupon code', 'offer price', 'offer_id', 'date')
- **Payment** ('cod', 'advance payment', 'amount', 'time')
- **Net banking** ('date', 'bank name', 'phone number', 'bank_id')
- **Upi** ('upi id', 'phone number', 'amount', 'name')
- **Checkout** ('order_id', 'total amount', 'time')
- **Delivery** ('dine in', 'take away')

Schema diagram:

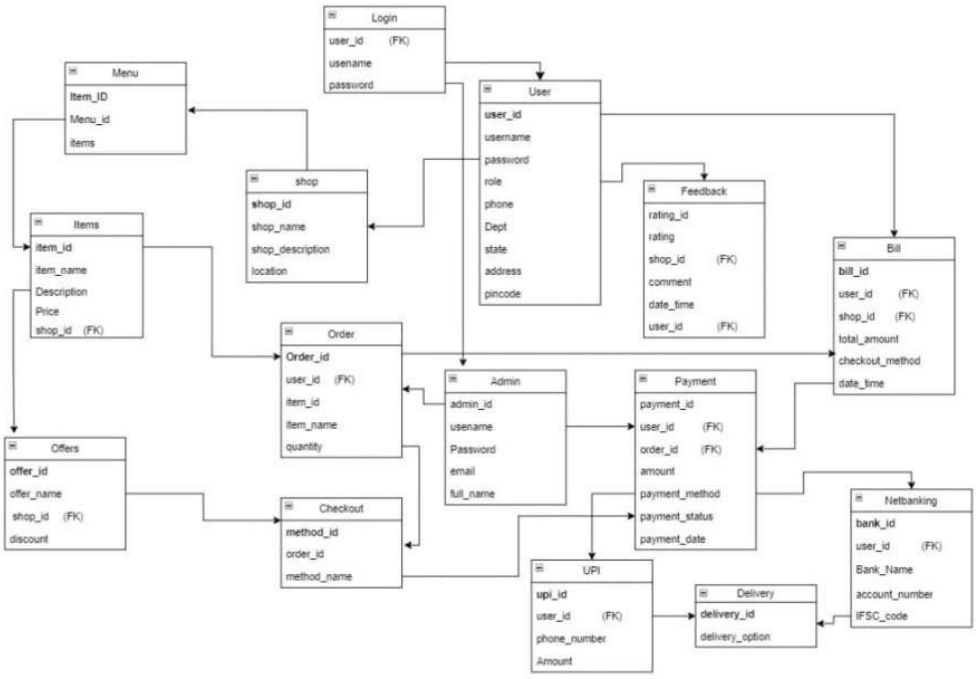


Figure 2.1 : Schema Diagram

2.2 Creation of Database Tables for the project Creating the User table:

User :

```
CREATE TABLE user (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50),
   last_name VARCHAR(50),
   username VARCHAR(50) UNIQUE,
   password VARCHAR(255),  role
   role VARCHAR(50),
   phone VARCHAR(15),
   Dept VARCHAR(50),
  State VARCHAR(50),
  address VARCHAR(255),
  pincode VARCHAR(10)
)
```

Login :

```
CREATE TABLE Login (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) UNIQUE,
  password VARCHAR(255),
  FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

Shop :

```
CREATE TABLE Shops (     shop_id INT PRIMARY KEY,
    shop_name VARCHAR(255) NOT NULL,
    shop_description TEXT,
    location VARCHAR(255)
);
```

Items:

```
CREATE TABLE Items (     item_id INT PRIMARY KEY,
    item_name VARCHAR(255) NOT NULL,
   shop_id INT,     price DECIMAL(10, 2),
    description TEXT,
FOREIGN KEY (shop_id) REFERENCES Shops(shop_id)
);
```

Order:

```
CREATE TABLE Order (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,     quantity INT,
    item_name VARCHAR(255),
    item_id INT,
    FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

Feedback :

```
CREATE TABLE Ratings (     rating_id INT PRIMARY KEY,
user_id INT,     shop_id INT,
rating INT,
comment TEXT,
date_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (user_id) REFERENCES User(user_id),
FOREIGN KEY (shop_id) REFERENCES Shops(shop_id)
);
```

Admin :

```
CREATE TABLE Admin (
admin_id INT AUTO_INCREMENT PRIMARY KEY,
username VARCHAR(50) UNIQUE,
password VARCHAR(255),
email VARCHAR(100),
full_name VARCHAR(100)
);
```

Bill :

```
CREATE TABLE Bill (     bill_id INT PRIMARY KEY,
user_id INT,
shop_id INT,
total_amount DECIMAL(10, 2),
checkout_method CHAR,
date_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
FOREIGN KEY (user_id) REFERENCES User(user_id),
FOREIGN KEY (shop_id) REFERENCES Shops(shop_id),
FOREIGN KEY (checkout_method_id) REFERENCES
CheckoutMethods(method_id)
);
```

Offers:

```
 CREATE TABLE Offers (     offer_id INT PRIMARY KEY,
 offer_name VARCHAR(255) NOT NULL,
 shop_id INT,     discount DECIMAL(5, 2),
 description TEXT,
 FOREIGN KEY (shop_id) REFERENCES Shops(shop_id)
);
```

Payment :

```
CREATE TABLE Payment (
    payment_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT,
    order_id INT,
    amount DECIMAL(10, 2),
    payment_method ENUM('Cash on Delivery', 'UPI', 'Net banking'),
    payment_status ENUM('pending', 'completed'),
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (order_id) REFERENCES Order(order_id)
);
```

Net Banking :

```
CREATE TABLE NetBanking (     id INT AUTO_INCREMENT PRIMARY KEY,
 user_id INT,
bank_name VARCHAR(100),
account_number VARCHAR(50),
IFSC_code VARCHAR(20),
 FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

UPI :

```
CREATE TABLE UPI ( id INT AUTO_INCREMENT PRIMARY KEY,
 phone_number VARCHAR(15),
 amount DECIMAL(10, 2),
 user_id INT,
FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

Checkout :

```
CREATE TABLE CheckoutMethods (     method_id INT
PRIMARY KEY,
 method_name VARCHAR(100) NOT NULL
);
```

Delivery :

```
CREATE TABLE Delivery (     delivery_id INT
AUTO_INCREMENT PRIMARY KEY,     delivery_option
 ENUM('dine-in', 'takeaway')
);
```

```
mysql> show tables;
+-------------------------------+
| Tables_in_e_canteen_database  |
+-------------------------------+
| admin                         |
| bill                          |
| checkoutmethods               |
| delivery                      |
| emp                           |
| items                         |
| login                         |
| netbanking                    |
| offers                        |
| order                         |
| payment                       |
| ratings                       |
| shops                         |
| upi                           |
| user                          |
+-------------------------------+
15 rows in set (0.06 sec)
```

```
mysql> desc login;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| user_id  | int          | NO   | PRI | NULL    | auto_increment |
| username | varchar(50)  | YES  | UNI | NULL    |                |
| password | varchar(255) | YES  |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> desc netbanking;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| id             | int          | NO   | PRI | NULL    | auto_increment |
| user_id        | int          | YES  | MUL | NULL    |                |
| bank_name      | varchar(100) | YES  |     | NULL    |                |
| account_number | varchar(50)  | YES  |     | NULL    |                |
| IFSC_code      | varchar(20)  | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)

mysql> desc offers;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| offer_id    | int          | NO   | PRI | NULL    |       |
| offer_name  | varchar(255) | NO   |     | NULL    |       |
| shop_id     | int          | YES  | MUL | NULL    |       |
| discount    | decimal(5,2) | YES  |     | NULL    |       |
| description | text         | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

```
mysql> desc upi;
+--------------+---------------+------+-----+---------+----------------+
| Field        | Type          | Null | Key | Default | Extra          |
+--------------+---------------+------+-----+---------+----------------+
| id           | int           | NO   | PRI | NULL    | auto_increment |
| phone_number | varchar(15)   | YES  |     | NULL    |                |
| amount       | decimal(10,2) | YES  |     | NULL    |                |
| user_id      | int           | YES  | MUL | NULL    |                |
+--------------+---------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> desc user;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| user_id    | int          | NO   | PRI | NULL    | auto_increment |
| first_name | varchar(50)  | YES  |     | NULL    |                |
| last_name  | varchar(50)  | YES  |     | NULL    |                |
| username   | varchar(50)  | YES  | UNI | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
| role       | varchar(50)  | YES  |     | NULL    |                |
| phone      | varchar(15)  | YES  |     | NULL    |                |
| country    | varchar(50)  | YES  |     | NULL    |                |
| state      | varchar(50)  | YES  |     | NULL    |                |
| address    | varchar(255) | YES  |     | NULL    |                |
| pincode    | varchar(10)  | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
11 rows in set (0.00 sec)
```

```
mysql> desc admin;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| admin_id  | int          | NO   | PRI | NULL    | auto_increment |
| username  | varchar(50)  | YES  | UNI | NULL    |                |
| password  | varchar(255) | YES  |     | NULL    |                |
| email     | varchar(100) | YES  |     | NULL    |                |
| full_name | varchar(100) | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
5 rows in set (0.02 sec)

mysql> desc bill;
+--------------------+---------------+------+-----+-------------------+--------------------+
| Field              | Type          | Null | Key | Default           | Extra              |
+--------------------+---------------+------+-----+-------------------+--------------------+
| bill_id            | int           | NO   | PRI | NULL              |                    |
| user_id            | int           | YES  | MUL | NULL              |                    |
| shop_id            | int           | YES  | MUL | NULL              |                    |
| total_amount       | decimal(10,2) | YES  |     | NULL              |                    |
| checkout_method_id | int           | YES  | MUL | NULL              |                    |
| date_time          | timestamp     | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED  |
+--------------------+---------------+------+-----+-------------------+--------------------+
6 rows in set (0.01 sec)

mysql> desc checkoutmethods;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| method_id   | int          | NO   | PRI | NULL    |       |
| method_name | varchar(100) | NO   |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

```
mysql> desc delivery;
+-----------------+-----------------------------+------+-----+---------+----------------+
| Field           | Type                        | Null | Key | Default | Extra          |
+-----------------+-----------------------------+------+-----+---------+----------------+
| delivery_id     | int                         | NO   | PRI | NULL    | auto_increment |
| delivery_option | enum('dine-in','takeaway') | YES  |     | NULL    |                |
+-----------------+-----------------------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> desc emp;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name_id | varchar(10) | YES  |     | NULL    |       |
| ph_no   | int         | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> desc items;
+-------------+---------------+------+-----+---------+-------+
| Field       | Type          | Null | Key | Default | Extra |
+-------------+---------------+------+-----+---------+-------+
| item_id     | int           | NO   | PRI | NULL    |       |
| item_name   | varchar(255)  | NO   |     | NULL    |       |
| shop_id     | int           | YES  | MUL | NULL    |       |
| price       | decimal(10,2) | YES  |     | NULL    |       |
| description | text          | YES  |     | NULL    |       |
+-------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

```
mysql> desc payment;
+----------------+-----------------------------------------------+------+-----+-------------------+-------------------+
| Field          | Type                                          | Null | Key | Default           | Extra             |
+----------------+-----------------------------------------------+------+-----+-------------------+-------------------+
| payment_id     | int                                           | NO   | PRI | NULL              | auto_increment    |
| user_id        | int                                           | YES  | MUL | NULL              |                   |
| order_id       | int                                           | YES  | MUL | NULL              |                   |
| amount         | decimal(10,2)                                 | YES  |     | NULL              |                   |
| payment_method | enum('Cash on Delivery','UPI','Net banking') | YES  |     | NULL              |                   |
| payment_status | enum('pending','completed')                   | YES  |     | NULL              |                   |
| payment_date   | timestamp                                     | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+----------------+-----------------------------------------------+------+-----+-------------------+-------------------+
7 rows in set (0.00 sec)

mysql> desc ratings;
+-----------+-----------+------+-----+-------------------+-------------------+
| Field     | Type      | Null | Key | Default           | Extra             |
+-----------+-----------+------+-----+-------------------+-------------------+
| rating_id | int       | NO   | PRI | NULL              |                   |
| user_id   | int       | YES  | MUL | NULL              |                   |
| shop_id   | int       | YES  | MUL | NULL              |                   |
| rating    | int       | YES  |     | NULL              |                   |
| comment   | text      | YES  |     | NULL              |                   |
| date_time | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+-----------+-----------+------+-----+-------------------+-------------------+
6 rows in set (0.00 sec)

mysql> desc shops;
+------------------+--------------+------+-----+---------+-------+
| Field            | Type         | Null | Key | Default | Extra |
+------------------+--------------+------+-----+---------+-------+
| shop_id          | int          | NO   | PRI | NULL    |       |
| shop_name        | varchar(255) | NO   |     | NULL    |       |
| shop_description | text         | YES  |     | NULL    |       |
| location         | varchar(255) | YES  |     | NULL    |       |
+------------------+--------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

CHAPTER 3

Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors

Subquery:

1.Subquery to retrieve users who have made orders:

```
SELECT Username
FROM Login
WHERE User_ID IN (SELECT DISTINCT User_ID FROM Orders);
```

	Username
▶	user1
	user2
	user3
	user4
	user5

2. Subquery to get the total number of orders per user:

```
SELECT Username, (SELECT COUNT(*) FROM Orders WHERE Orders.User_ID =
Login.User_ID) AS Total_Orders
FROM Login;
```

	Username	Total_Orders
▶	user1	1
	user2	1
	user3	1
	user4	1
	user5	1

3.Subquery to find users who haven't made any orders:

```
SELECT Username
```

```
FROM Login
WHERE User_ID NOT IN (SELECT DISTINCT User_ID FROM Orders);
```

Constarints:

Primary Key Constraints:

Each table likely has a primary key constraint to ensure that each row is uniquely identifiable.

Examples: id in the User table, login_id in the Login table, item_id in the Items table, etc.

Foreign Key Constraints:

Foreign key constraints establish relationships between tables by referencing the primary key of another table.

Example: user_id in the Order table referencing the id in the User table.

Unique Constraints:

Unique constraints ensure that no two rows have the same value for a specific attribute.

Examples: Unique constraint on username in the Login table, unique constraint on item_id in the Order table to prevent duplicate orders for the same item by the same user.

Not Null Constraints:

Not null constraints prevent the insertion of null values in specific columns.

Examples: name, address, and contact number in the User table are likely not nullable.

Check Constraints:

Check constraints enforce specific conditions on the values allowed in a column.

Example: Check constraint on quantity in the Order table to ensure it's greater than zero.

Referential Integrity Constraints:

These constraints ensure that relationships between tables remain consistent.

Example: Ensuring that the user_id in the Order table always references an existing id in the User table.

Entity Integrity Constraints:

These constraints ensure that primary keys are unique and not null.

Example: Ensuring that every row in the User table has a unique id and that it's not null.

Business Rules Constraints:

Constraints that enforce specific business rules or requirements.

Example: A constraint ensuring that the order_date in the Order table cannot be in the future.

JOINS:

Inner Join:

Inner join returns only the rows where there is a match between the columns in both tables.

>SELECT *

FROM Order

INNER JOIN Items ON Order.item_id = Items.item_id;

Left Join:

Left join returns all rows from the left table (Order in this case), and the matching rows from the right table (Items).

>SELECT *

FROM Order

LEFT JOIN Items ON Order.item_id = Items.item_id;

Right Join:

Right join returns all rows from the right table (Items in this case), and the matching rows from the left table (Order).

>SELECT *

FROM Order

RIGHT JOIN Items ON Order.item_id = Items.item_id;

Full Outer Join:

Full outer join returns all rows from both tables, with NULL values in places where there is no match.

>SELECT *

FROM Order

FULL OUTER JOIN Items ON Order.item_id = Items.item_id;

Self Join:

Self join is used to join a table to itself.

>SELECT a.name, b.name

FROM User a, User b

WHERE a.department = b.department

AND a.id != b.id;

SET OPERATIONS FOR THIS:

UNION:

The UNION operator combines the results of two or more SELECT statements into a single result set, eliminating duplicate rows.

-- Find all unique items purchased by users in orders

>SELECT item_id FROM Order

UNION

SELECT item_id FROM Items;

UNION ALL:

Similar to UNION, but it includes all rows from each SELECT statement, even if there are duplicates.

-- Find all items purchased in orders and their corresponding descriptions

SELECT item_id, description FROM Order

UNION ALL

SELECT item_id, description FROM Items;

INTERSECT:

The INTERSECT operator returns only the rows that appear in both result sets of two SELECT statements.

-- Find items that are both available in the shop and have been ordered

SELECT item_id FROM Items

INTERSECT

SELECT item_id FROM Order;

EXCEPT / MINUS:

The EXCEPT operator returns all distinct rows from the first SELECT statement that are not returned by the second SELECT statement.

Note: Some databases use MINUS instead of EXCEPT.

-- Find items available in the shop but not yet ordered

SELECT item_id FROM Items

EXCEPT

SELECT item_id FROM Order;

Intersections:

Intersection of Users in Orders and Users in Feedback:

Find users who have placed orders and provided feedback.

```
SELECT user_id FROM Order
INTERSECT
SELECT customer_id FROM Feedback;
```

Intersection of Items Ordered and Items Available in the Shop:

Find items that have been ordered and are also available in the shop.

```
SELECT item_id FROM Order
INTERSECT
SELECT item_id FROM Items;
```

Intersection of Users Placing Orders and Users in Feedback with High Service Quality:

Find users who have placed orders and also provided feedback with high service quality ratings.

```
SELECT user_id FROM Order
INTERSECT
SELECT customer_id FROM Feedback WHERE service&quality > 4;
```

Views:

UserOrdersView:

A view that combines user information with their corresponding orders.

sql

```
CREATE VIEW UserOrdersView AS
SELECT u.id AS user_id, u.name AS user_name, u.department, u.address, u.contact_number,
    o.order_id, o.quantity, o.item_id
FROM User u
INNER JOIN Order o ON u.id = o.user_id;
```

ItemDetailsView:

A view that provides detailed information about items, including their description.

sql

```
CREATE VIEW ItemDetailsView AS
SELECT i.item_id, i.name AS item_name, i.description, i.price, o.order_id
FROM Items i
LEFT JOIN Order o ON i.item_id = o.item_id;
```

FeedbackSummaryView:

A view that summarizes feedback provided by customers.

```
CREATE VIEW FeedbackSummaryView AS
SELECT customer_id, AVG(service&quality) AS avg_service_quality, COUNT(*) AS
total_feedback
FROM Feedback
GROUP BY customer_id;
```

OrdersWithBillingView:

A view that combines order information with billing details.

```
CREATE VIEW OrdersWithBillingView AS
SELECT o.order_id, o.user_id, o.quantity, o.item_id, b.bill_no, b.bill_date
FROM Order
LEFT JOIN Bill b ON o.order_id = b.order_id;
```

OfferExpiryView:

A view that lists offers along with their expiration dates.

```
CREATE VIEW OfferExpiryView AS
SELECT *, DATE_ADD(date, INTERVAL 1 MONTH) AS expiry_date
FROM Offers;
```

Triggers:

Audit Trigger for Orders:

This trigger logs changes to the Order table, capturing details such as who made the change, what action was performed, and when it occurred.

```
CREATE TRIGGER OrderAuditTrigger
AFTER INSERT, UPDATE, DELETE ON Order
FOR EACH ROW
BEGIN
   IF INSERTING THEN
      INSERT INTO OrderAudit (user_id, action, timestamp)
      VALUES (NEW.user_id, 'INSERT', NOW());
   ELSIF UPDATING THEN
      INSERT INTO OrderAudit (user_id, action, timestamp)
      VALUES (NEW.user_id, 'UPDATE', NOW());
   ELSIF DELETING THEN
      INSERT INTO OrderAudit (user_id, action, timestamp)
      VALUES (OLD.user_id, 'DELETE', NOW());
   END IF;
END;
```

Automatic Calculation of Total Amount in Bill:

This trigger automatically calculates the total amount in a bill whenever a new order is added or an existing order is updated or deleted.

```
CREATE TRIGGER CalculateTotalAmount
AFTER INSERT, UPDATE, DELETE ON Order
FOR EACH ROW
BEGIN
   DECLARE total_amount DECIMAL(10, 2);
```

```
    SET total_amount = (SELECT SUM(quantity * price) FROM Order WHERE order_id =
NEW.order_id);
    UPDATE Bill SET total_amount = total_amount WHERE order_id = NEW.order_id;
END;
```

Enforcing Referential Integrity for Orders:

This trigger ensures that orders cannot be placed for items that do not exist in the Items table.

```
CREATE TRIGGER CheckItemExistence
BEFORE INSERT ON Order
FOR EACH ROW
BEGIN
    DECLARE item_count INT;
    SELECT COUNT(*) INTO item_count FROM Items WHERE item_id = NEW.item_id;
    IF item_count = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid item ID';
    END IF;
END;
```

Auto-Generate Order ID:

This trigger automatically generates a unique order ID whenever a new order is inserted into the Order table.

```
CREATE TRIGGER GenerateOrderID
BEFORE INSERT ON Order
FOR EACH ROW
BEGIN
    SET NEW.order_id = CONCAT('ORD', LPAD((SELECT COUNT(*) FROM Order) + 1,
5, '0'));
END;
```

Enforcing Constraints in Feedback:

This trigger ensures that the service&quality rating provided in feedback is within a valid range (e.g., 1 to 5).

```
CREATE TRIGGER CheckFeedbackRating
BEFORE INSERT ON Feedback
```

```
FOR EACH ROW
BEGIN
   IF NEW.service&quality < 1 OR NEW.service&quality > 5 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid service&quality rating';
   END IF;
END;
```

Cursors:

```
DECLARE user_cursor CURSOR FOR
SELECT id, name, department, address, otp, contact_number
FROM User;

DECLARE login_cursor CURSOR FOR
SELECT login_id, captcha, username, password
FROM Login;

DECLARE shop_cursor CURSOR FOR
SELECT shop_name, shop_id, location
FROM Shop;

DECLARE items_cursor CURSOR FOR
SELECT item_id, name, description, price
FROM Items;

DECLARE order_cursor CURSOR FOR
SELECT user_id, order_id, quantity, item_id
FROM Order;

DECLARE feedback_cursor CURSOR FOR
SELECT suggestion, customer_id, service_quality, fd_id
FROM Feedback;
```

```
DECLARE admin_cursor CURSOR FOR
SELECT name, password, admin_id
FROM Admin;

DECLARE bill_cursor CURSOR FOR
SELECT bill_no, user_id, order_id, bill_date
FROM Bill;

DECLARE offers_cursor CURSOR FOR
SELECT coupon_code, offer_price, offer_id, date
FROM Offers;

DECLARE payment_cursor CURSOR FOR
SELECT cod, advance_payment, amount, time
FROM Payment;

DECLARE net_banking_cursor CURSOR FOR
SELECT date, bank_name, phone_number, bank_id
FROM Net_banking;

DECLARE upi_cursor CURSOR FOR
SELECT upi_id, phone_number, amount, name
FROM Upi;
DECLARE checkout_cursor CURSOR FOR
SELECT order_id, total_amount, time
FROM Checkout;
DECLARE delivery_cursor CURSOR FOR
SELECT dine_in, take_away
FROM Delivery;
```

Chapter-4

Analyzing the pitfalls, identifying the dependencies, and applying normalizations

Pitfalls in RDBMS design:

Redundancy:

Redundancy in database design leads to data inconsistencies, increased storage requirements, and risks of anomalies during updates, insertions, and deletions. It complicates data maintenance and management, requiring extra effort to ensure consistency across redundant copies. To mitigate these issues, database designers should aim for proper normalization to minimize redundancy and dependency, and enforce referential integrity constraints to maintain data consistency.

Inconsistency:

Inconsistency in database design arises from redundant data, leading to discrepancies between copies. It results in challenges in data management, with updates made to one copy not reflecting in others, risking data integrity. Addressing redundancy through normalization and enforcing constraints helps mitigate inconsistency issues, ensuring data accuracy and reliability.

Inefficiency:

Inefficiency in database design stems from poor indexing, over-normalization, and inadequate query optimization. It leads to slow query performance, resource wastage, and scalability limitations. Prioritizing indexing on frequently queried columns, balancing normalization with performance, and optimizing queries can mitigate inefficiency, enhancing overall database performance.

Complexity:

Complexity in database design results from over-normalization, inconsistent naming conventions, and inadequate documentation. It leads to difficulties in understanding and maintaining the database schema, increasing the likelihood of errors and inefficiencies. Simplifying normalization, establishing clear naming conventions, and thorough documentation can alleviate complexity, facilitating easier database management and development.

Functional dependencies:

Functional dependencies in a relational schema define the relationships between attributes in a table. Here are the functional dependencies based on the provided relational schema:

1. User

- id → name, department, address, otp, contact_number
- contact_number → id, name, department, address, otp

2. Login

- login_id → captcha, username, password
- username → captcha, login_id, password
- password → captcha, login_id, username

3. Shop

- shop_id → shop_name, location
- location → shop_id, shop_name

4. Items

- item_id → name, description, price
- name → item_id, description, price
- description → item_id, name, price
- price → item_id, name, description

5. Order

- (user_id, order_id) → quantity, item_id
- user_id → order_id, quantity, item_id
- order_id → user_id, quantity, item_id
- item_id → user_id, order_id, quantity

6. Feedback

- fd_id → suggestion, customer_id, service&quality
- customer_id → fd_id, suggestion, service&quality

7. Admin

- admin_id → name, password
- name → admin_id, password
- password → admin_id, name

8. Bill

- (user_id, order_id) → bill_no, bill_date
- bill_no → user_id, order_id, bill_date
- bill_date → user_id, order_id, bill_no

9. Offers

- offer_id → coupon_code, offer_price, date
- coupon_code → offer_id, offer_price, date
- date → offer_id, coupon_code, offer_price

10. Payment

- (cod, advance_payment) → amount, time
- amount → cod, advance_payment, time
- time → cod, advance_payment, amount

11. Net banking

- bank_id → date, bank_name, phone_number
- bank_name → bank_id, date, phone_number
- phone_number → bank_id, date, bank_name

12. Upi

- (upi_id, phone_number) → amount, name
- amount → upi_id, phone_number, name
- name → upi_id, phone_number, amount

13. Checkout

- order_id → total_amount, time
- total_amount → order_id, time
- time → order_id, total_amount

14. Delivery

- (dine_in, take_away) → NULL

These functional dependencies illustrate the relationships between attributes within each table in the relational schema, helping to maintain data integrity and ensure consistency in the database.

NORMALIZATION

Normalization in databases is the process of organizing data in a relational database to reduce redundancy and dependency. It involves breaking down large tables into smaller, more manageable tables and defining relationships between them. Normalization typically involves several normal forms (1NF, 2NF, 3NF, BCNF, etc.) to ensure data integrity and optimize database performance. The goal of normalization is to minimize data redundancy, improve data integrity, and make the database structure more flexible and adaptable to changes.

First Normal Form (1NF)

First Normal Form (1NF) is a fundamental concept in database normalization, ensuring that data is organized efficiently in a relational database. To adhere to 1NF, each attribute in a table must hold a single atomic value, meaning it cannot be further divided. This rule helps prevent data redundancy and inconsistency. For example, instead of having a "Phone Numbers" column that stores multiple phone numbers in a single row, each phone number should be stored in a separate row to maintain atomicity. Additionally, each attribute should contain only one value from its domain, avoiding the storage of lists or arrays. This principle promotes data integrity and simplifies query operations. Lastly, all values in a column should be of the same data type to ensure uniformity and consistency. By structuring data according to 1NF, databases can maintain organized, reliable, and easily accessible information.

Second Normal Form (2NF)

Second Normal Form (2NF) is a critical concept in database normalization, aimed at improving the integrity and structure of data within relational databases. To comply with 2NF, a table must first satisfy the requirements of First Normal Form (1NF). Beyond 1NF, 2NF addresses the issue of partial dependencies within a table. It ensures that every non-prime attribute (an attribute that is not part of any candidate key) is fully functionally dependent on the entire primary key. This means that if a table has a composite primary key (made up of multiple columns), each non-prime attribute should depend on the entire composite key, not just part of it. If any non-prime attribute is found to be dependent on only part of the primary key, it indicates a violation of 2NF, and the table needs to be restructured. By enforcing 2NF, databases can eliminate redundancy and inconsistency, leading to more efficient data storage and retrieval processes.

Third Normal Form (3NF)

Third Normal Form (3NF) is a crucial principle in database normalization, building upon the foundation of the First (1NF) and Second (2NF) Normal Forms. In essence, 3NF mandates that every non-prime attribute in a table is non-transitively dependent on the primary key. This means that no column should depend on another non-primary key column, thereby avoiding transitive dependencies. To achieve 3NF, tables must first adhere to 2NF and then ensure

that any non-prime attribute depends only on the primary key, not on another non-prime attribute. By enforcing 3NF, databases can significantly reduce data redundancy and anomalies, leading to more efficient storage, improved data integrity, and simplified data maintenance.

Boyce Codd Normal Form (BCNF)

Boyce-Codd Normal Form (BCNF), named after Raymond Boyce and Edgar Codd, is a stricter form of normalization in database design. BCNF is an extension of the Third Normal Form (3NF) and is based on functional dependencies in a table. A table is in BCNF if and only if every determinant (attribute or set of attributes that uniquely determines another attribute) is a candidate key. This means that for every non-trivial functional dependency $A \rightarrow B$ in a table, A must be a superkey. BCNF helps to eliminate anomalies that can occur due to non-trivial functional dependencies between attributes. By ensuring that all determinants are candidate keys, BCNF minimizes redundancy and improves data integrity in a database schema. However, achieving BCNF may result in more tables compared to lower normal forms, which can lead to more complex queries and potentially impact performance.

Fourth Normal Form (4NF)

Fourth Normal Form (4NF) is a level of database normalization that focuses on further reducing redundancy and anomalies in relational database schemas. It builds upon the concepts of the earlier normal forms (1NF, 2NF, and 3NF) by addressing certain types of complex dependencies called multivalued dependencies. In 4NF, a table is required to be in 3NF first, and then it must ensure that there are no non-trivial multivalued dependencies between sets of attributes other than those involving candidate keys. This means that every non-trivial multivalued dependency $X \rightarrow\rightarrow Y$ in a table must have either X as a superkey or Y as a subset of a candidate key. By adhering to 4NF, database designers can create more efficient and logically structured database schemas, though achieving 4NF may require additional tables or restructuring of existing tables to comply with its requirements.

Fifth Normal Form (5NF)

Fifth Normal Form (5NF), also known as Project-Join Normal Form (PJNF), is a level of database normalization that deals with the reduction of redundancy in relational database schemas. 5NF is based on the concept of join dependencies, which are a generalization of functional dependencies. In 5NF, a table is considered to be in 5NF if and only if every join dependency in the table is implied by the candidate keys. This means that the table cannot contain any redundancy that can be removed by splitting the table into smaller tables and using natural joins to retrieve the original information.Achieving 5NF can be complex and may require a deep understanding of the data and its dependencies. While 5NF helps to reduce redundancy and improve data integrity, it may lead to more complex query operations and potentially impact

Creating initial table of orders:

```
CREATE TABLE Orders (
    user_id INT,
    order_id INT,
    quantity INT,
    item_id INT,
    PRIMARY KEY (user_id, order_id), -- Composite primary key
    FOREIGN KEY (user_id) REFERENCES User(id),
    FOREIGN KEY (item_id) REFERENCES Items(item_id)
);
```

Inserting values:

```
INSERT INTO Orders (user_id, order_id, quantity, item_id)
VALUES
    (1, 1, 2, 101), -- User with id 1 orders 2 items with id 101
    (2, 2, 1, 102), -- User with id 2 orders 1 item with id 102
    (1, 3, 3, 103); -- User with id 1 orders 3 items with id 103
```

Based on the provided insertion values, here's how the "Orders" table would look like after the insertions:

user_id	order_id	quantity	item_id
1	1	2	101
2	2	1	102
1	3	3	103

1 NF Normal form :

```
CREATE TABLE Order (
    user_id INT,
    order_id INT,
    quantity INT,
    item_id INT,
    PRIMARY KEY (order_id),
    FOREIGN KEY (user_id) REFERENCES User(id),
    FOREIGN KEY (item_id) REFERENCES Items(item_id)
);
```

Insert into values:

```
INSERT INTO Orders (order_id, user_id, item_id, quantity)
VALUES
    (1, 1, 101, 2),
    (2, 2, 102, 1),
    (3, 1, 103, 3);
```

order_id	user_id	item_id	quantity
1	1	101	2
2	2	102	1
3	1	103	3

2NF order of table:

```
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    user_id INT,
    item_id INT,
    quantity INT,
    FOREIGN KEY (user_id) REFERENCES User(id),
    FOREIGN KEY (item_id) REFERENCES Items(item_id)
);
INSERT INTO Order_Details (Product_Details, Address, User_ID, Payment_ID)
| order_id | user_id | item_id | quantity |
|----------|---------|---------|----------|
|    1     |    1    |   101   |    2     |
|    2     |    2    |   102   |    1     |
|    3     |    1    |   103   |    3     |


CREATE TABLE User_Details (
User_ID INT PRIMARY KEY AUTO_INCREMENT,
 Email_ID VARCHAR(255),
  Name VARCHAR(255),
  DateOfBirth DATE,
  Age INT,
  Gender VARCHAR(10),
 Phone_Number VARCHAR(15),
 Address VARCHAR(255)


);
```

Inserting the values:

```
INSERT INTO User_Details (Email_ID, Name, DateOfBirth, Age, Gender, Phone_Number,
Address)

VALUES

('user1@example.com', 'User One', '1990-01-01', 32, 'Male', '1234567890', '123 Street'),
 ('user2@example.com', 'User Two', '1995-02-02', 27, 'Female', '2345678901', '456 Avenue'),
 ('user3@example.com', 'User Three', '1985-03-03', 37, 'Other', '3456789012', '789 Road'),
 ('user4@example.com', 'User Four', '1980-04-04', 42, 'Male', '4567890123', '012 Boulevard'),
 ('user5@example.com', 'User Five', '1975-05-05', 47, 'Female', '5678901234', '345 Lane');
```

Creating the User Table:

```
CREATE TABLE User_Details (
User_ID INT PRIMARY KEY AUTO_INCREMENT,
 Email_ID VARCHAR(255),
  Name VARCHAR(255),
  DateOfBirth DATE,
  Age INT,
  Gender VARCHAR(10),
 Phone_Number VARCHAR(15),
 Address VARCHAR(255)
);
```

Inserting the values:

INSERT INTO User_Details (Email_ID, Name, DateOfBirth, Age, Gender, Phone_Number, Address)

VALUES

('user1@example.com', 'User One', '1990-01-01', 32, 'Male', '1234567890', '123 Street'),
('user2@example.com', 'User Two', '1995-02-02', 27, 'Female', '2345678901', '456 Avenue'),
('user3@example.com', 'User Three', '1985-03-03', 37, 'Other', '3456789012', '789 Road'),
('user4@example.com', 'User Four', '1980-04-04', 42, 'Male', '4567890123', '012 Boulevard'),
('user5@example.com', 'User Five', '1975-05-05', 47, 'Female', '5678901234', '345 Lane');

Chapter-5

Implementation of concurrency control and recover mechanisms

Concurrency control in databases is a crucial aspect of ensuring data integrity and consistency in multiuser environments. It refers to the mechanisms and techniques used to manage and coordinate simultaneous access to data by multiple transactions. Without proper concurrency control, transactions could interfere with each other, leading to data corruption or inconsistencies.

One common approach to concurrency control is locking, where transactions acquire locks on data items to prevent other transactions from modifying them concurrently. There are different types of locks, such as read locks and write locks, which control the level of access permitted to a data item. Locking helps to maintain data integrity by ensuring that transactions can only access or modify data in a controlled manner.

Another approach to concurrency control is timestamping, where transactions are assigned unique timestamps based on their start times. Transactions are then ordered based on their timestamps, and conflicts are resolved by giving priority to transactions with earlier timestamps. Timestamping helps to ensure that transactions are executed in a serializable order, preventing conflicts and maintaining consistency.

Optimistic concurrency control is a different approach that assumes conflicts are rare. Transactions are allowed to proceed without acquiring locks, and conflicts are only checked for at the end of the transaction. If a conflict is detected, the transaction is rolled back and re-executed. This approach is less restrictive than locking but requires careful handling of conflicts to avoid unnecessary rollbacks.

Concurrency control is a complex area of database management, and different techniques may be more suitable depending on the specific requirements of the application. The goal of concurrency control is to ensure that transactions can execute concurrently without compromising data integrity, consistency, or performance.

The four important aspects of database management are:

1.Atomicity: Ensures that transactions are all or nothing, meaning either all operations within the transaction are successfully completed, or none of them are.

2.Consistency: Guarantees that the database remains in a consistent state before and after the execution of a transaction.

3.Isolation: Ensures that concurrent transactions do not interfere with each other, maintaining data integrity and preventing anomalies.

4.Durability: Ensures that once a transaction is committed, its effects are permanently saved and persisted, even in the event of a system failure.

Recovery Mechanism:

Recovery mechanisms in database management systems (DBMS) are essential for ensuring data durability and system reliability in the event of failures. These mechanisms are designed to restore the database to a consistent state after a failure and to minimize the impact of the failure on ongoing transactions. There are two main types of recovery mechanisms: undo recovery and redo recovery.

Undo recovery, also known as rollback, is the process of undoing the effects of transactions that were not completed at the time of failure. This ensures that any changes made by these transactions are not permanently applied to the database. Undo recovery typically involves using transaction logs to identify and reverse the changes made by incomplete transactions.

Redo recovery, on the other hand, is the process of reapplying the effects of transactions that were completed but not yet permanently stored in the database at the time of failure. This ensures that any changes made by these transactions are not lost. Redo recovery involves using transaction logs to identify and reapply the changes made by these transactions.

In addition to undo and redo recovery, DBMSs also use mechanisms such as checkpointing and logging to ensure data consistency and durability. Checkpointing involves periodically writing the current state of the database to disk, along with information about transactions that have been committed but not yet written to disk. This allows the system to recover to a consistent state after a failure by restoring the database to the last checkpoint and replaying the logs.

Logging is the process of recording all changes made to the database in a log file. This allows the system to reconstruct the state of the database at any point in time and to recover from failures by replaying the log from the last checkpoint. Logging also provides a way to ensure

the atomicity and durability of transactions, as changes are only considered committed once they have been recorded in the log.

Overall, recovery mechanisms are critical for maintaining data integrity and system reliability in database management systems. They ensure that data is not lost or corrupted in the event of

Chapter 6

Code for the project:

```
from flask import Flask, render_template, request, redirect, url_for
import mysql.connector
from datetime import datetime, timedelta
from flask import session

app = Flask(_name_)

# Database configuration
db_config = {
    'host': 'localhost',
    'port': 3306,
    'user': 'root',
    'password': 'aseena',
    'database': 'e_canteen_database'
}



# Connect to the database
conn = mysql.connector.connect(**db_config)
cursor = conn.cursor()

# Predefined admin password
ADMIN_PASSWORD = "1234@"
```

```
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/login', methods=['POST'])
def login_post():
    global username
    username = request.form.get('username')
    password = request.form.get('password')

    # Check if user is admin
    sql = "SELECT * FROM admin WHERE username = %s AND password =
%s"
    cursor.execute(sql, (username, password))
    admin = cursor.fetchone()

    if admin:
        return 'Admin login successful'
    else:
        # Check username and password against regular users table
        sql = "SELECT * FROM USER WHERE username = %s AND password =
%s"
        cursor.execute(sql, (username, password))
        user = cursor.fetchone()

        if user:
            return render_template('shop.html')
        else:
```

```
            return 'Invalid username or password'


@app.route('/login', methods=['GET'])
def login():
    return render_template('login.html')
@app.route('/cart')
def display_cart():
    try:
        # Fetch data from the cart table
        cursor.execute("SELECT * FROM cart")
        cart_items = cursor.fetchall()

        # Create a list to store the details of cart items
        cart_details = []

        # Fetch item details from the items table for each item in the cart
        for item in cart_items:
            item_id = item[0]

            # Fetch item details using item_id
            cursor.execute("SELECT * FROM items WHERE item_id = %s",
(item_id,))
            item_details = cursor.fetchone()

            # Append item details to the cart_details list
            if item_details:
                cart_details.append(item_details)
            else:
```

```
                print("No item details found for item_id:", item_id)

        # Pass the cart details to the template for rendering
        return render_template('cart.html', cart_details=cart_details)
    except Exception as e:
        print("Error fetching cart details:", e)
        return "An error occurred while fetching cart details."

@app.route('/add_to_cart', methods=['POST'])
def add_to_cart():
    if request.method == 'POST':
        # Retrieve data from the form
        shop_id = request.form.get('shop_id')
        item_id = request.form.get('item_id')

        # Retrieve username from session data



        if username:
            # Insert the data into the cart table
            sql_insert_cart = "INSERT INTO cart (shop_id, item_id, username, timestamp) VALUES (%s, %s, %s, %s)"
            timestamp = get_current_time()
            values = (shop_id, item_id, username, timestamp)
            cursor.execute(sql_insert_cart, values)
            conn.commit()

            return 'Item added to cart successfully'
```

```
        else:
            return 'User not logged in'
    else:
        return 'Invalid request method'
@app.route('/checkout',)
def checkout():

        return render_template('checkout.html')



@app.route('/new')
def new_user():
    return render_template('create_user_form.html')

@app.route('/new_user', methods=['POST'])
def add_user():
    if request.method == 'POST':
        # Get form data
        first_name = request.form['first_name']
        last_name = request.form['last_name']
        username = request.form['username']
        password = request.form['password']
        role = request.form['role']
        phone = request.form['phone']
        country = request.form['country']
        state = request.form['state']
```

```
        address = request.form['address']
        pincode = request.form['pincode']

        # Validate admin password if role is admin
        if role == "admin" and password != ADMIN_PASSWORD:
            return 'Incorrect admin password'

        # Check if username already exists
        sql_check_username = "SELECT * FROM USER WHERE username = %s"
        cursor.execute(sql_check_username, (username,))
        existing_user = cursor.fetchone()
        if existing_user:
            return 'Username already exists'

        # SQL query to insert user into the database
        sql = "INSERT INTO USER (first_name, last_name, username, password, role, phone, country, state, address, pincode) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"
        values = (first_name, last_name, username, password, role, phone, country, state, address, pincode)

        # Execute the SQL query
        cursor.execute(sql, values)
        conn.commit()

        return 'User added successfully'
    elif request.method == 'GET':
        return 'GET request is not supported for this route'
```

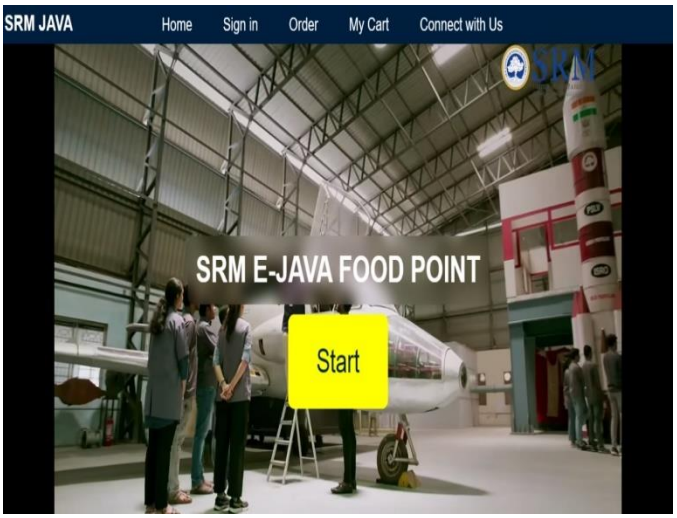
```
@app.route('/shop')
def shop():
    return render_template('shop.html')



# Route to handle shop selection and display items
# Route to handle shop selection and display items
@app.route('/shop/<shop_name>')
def display_items(shop_name):
    try:
        # Find shop ID based on shop name
        sql_shop_id = "SELECT shop_id FROM Shops WHERE shop_name = %s"
        cursor.execute(sql_shop_id, (shop_name,))
        shop_id_tuple = cursor.fetchone()

        # Extract the shop ID from the tuple
        shop_id = shop_id_tuple[0] if shop_id_tuple else None

        print("Requested shop name:", shop_name)
        print("Retrieved shop ID:", shop_id)

        if shop_id:
            # Use shop ID to search for items in Items table
            sql_items = "SELECT * FROM Items WHERE shop_id = %s"
            cursor.execute(sql_items, (shop_id,))
            items = cursor.fetchall()
```

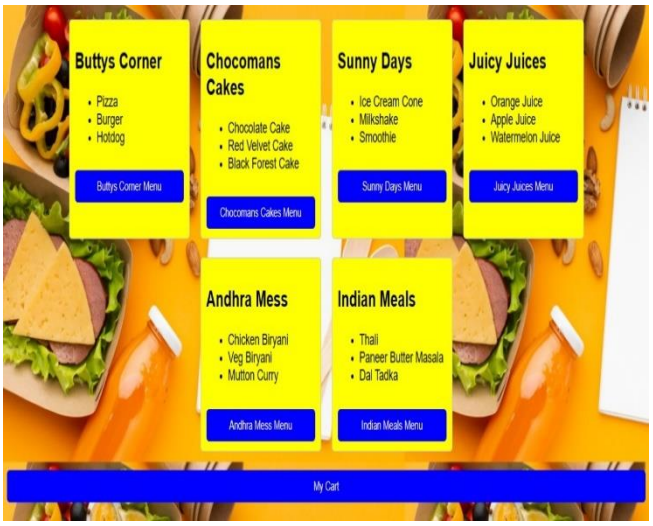
```
            return render_template('shop_items.html', items=items, shop_name=shop_name)
        else:
            return 'Shop not found'
    except Exception as e:
        return f"An error occurred: {str(e)}"



@app.route('/order', methods=['GET'])
def order_item():
    item_id = request.args.get('item_id')
    if item_id:
        # Process the item_id to fetch the price from the database
        cursor.execute("SELECT price FROM Items WHERE item_id = %s", (item_id,))
        price = cursor.fetchone()
        if price:
            amount = price[0]  # Retrieve the price from the database result
            print("Item ID:", item_id)
            print("Amount:", amount)
            # You can also perform additional database operations or processing here
            return render_template('payment.html', amount=amount)  # Pass the amount to the template
        else:
            return 'Price for item not found'
    else:
        return 'Item ID not provided'
@app.route('/payment')
```

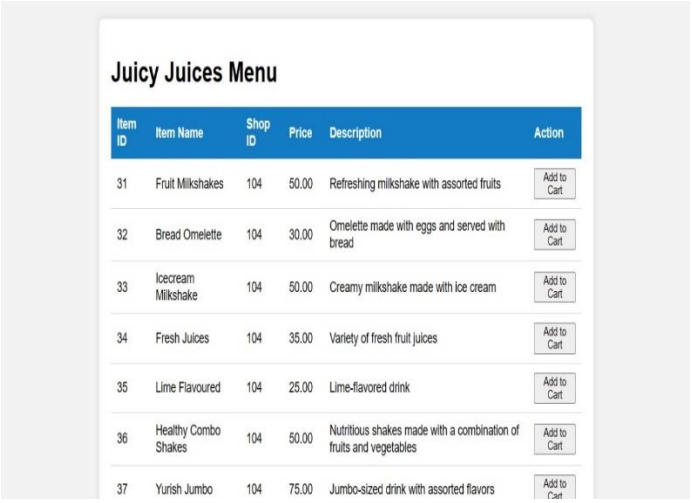
```
def payment():
    return render_template('payment.html')

@app.route('/place_order', methods=['POST'])
def place_order():
    # Add your order placement logic here
    # For now, let's just return a success message
    return 'Order placed successfully'

@app.route('/bill')
def process_payment():

  return render_template('bill.html')



def get_current_time():
    return datetime.now()



if _name_ == '_main_':
    app.run(debug=True)
```

Chapter 7

Result and discussion (Screen shots of the output)

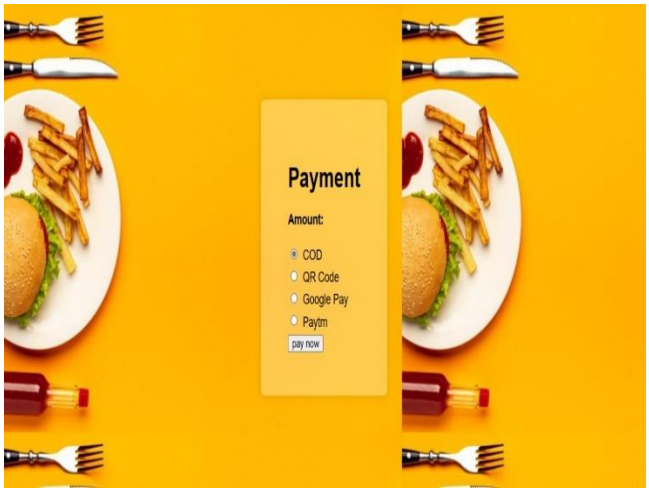


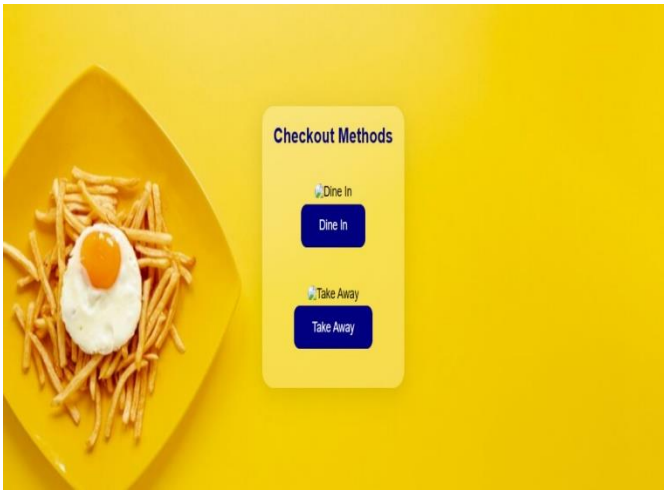


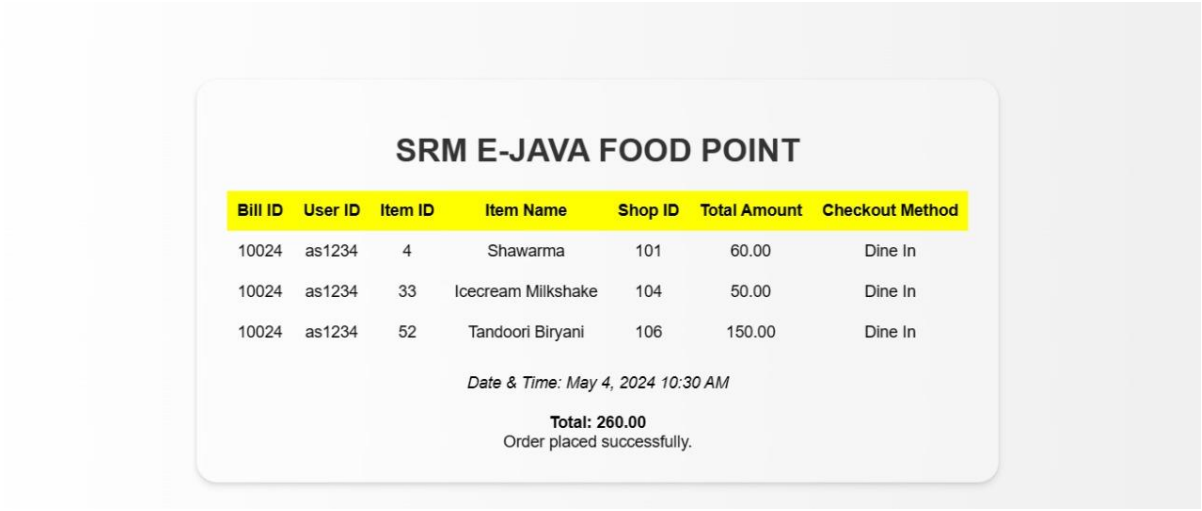


Juicy Juices Menu

Item ID	Item Name	Shop ID	Price	Description	Action
31	Fruit Milkshakes	104	50.00	Refreshing milkshake with assorted fruits	Add to Cart
32	Bread Omelette	104	30.00	Omelette made with eggs and served with bread	Add to Cart
33	Icecream Milkshake	104	50.00	Creamy milkshake made with ice cream	Add to Cart
34	Fresh Juices	104	35.00	Variety of fresh fruit juices	Add to Cart
35	Lime Flavoured	104	25.00	Lime-flavored drink	Add to Cart
36	Healthy Combo Shakes	104	50.00	Nutritious shakes made with a combination of fruits and vegetables	Add to Cart
37	Yurish Jumbo	104	75.00	Jumbo-sized drink with assorted flavors	Add to Cart







SRM E-JAVA FOOD POINT

Bill ID	User ID	Item ID	Item Name	Shop ID	Total Amount	Checkout Method
10024	as1234	4	Shawarma	101	60.00	Dine In
10024	as1234	33	Icecream Milkshake	104	50.00	Dine In
10024	as1234	52	Tandoori Biryani	106	150.00	Dine In

Date & Time: May 4, 2024 10:30 AM

Total: 260.00
Order placed successfully.

Chapter 8

Online course certification completion

udemy

Certificate no: UC-dd49c9ee-c878-46e0-bef5-5ed6190a9585
Certificate url: ude.my/UC-dd49c9ee-c878-46e0-bef5-5ed6190a9585
Reference Number: 0004

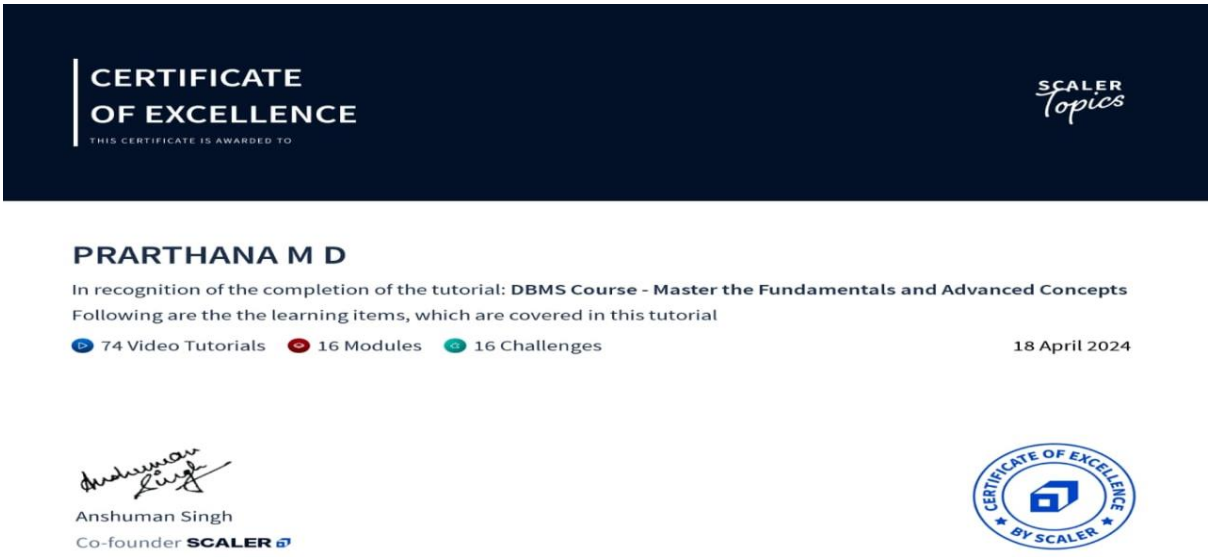
CERTIFICATE OF COMPLETION

Complete SQL and Databases Bootcamp

Instructors **Andrei Neagoie, Mo Binni**

M SULTHANA (RA2211030010317)

Date **April 18, 2024**
Length **24.5 total hours**



CERTIFICATE OF EXCELLENCE

THIS CERTIFICATE IS AWARDED TO

SCALER Topics

PRARTHANA M D

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advocated Concepts**
Following are the the learning items, which are covered in this tutorial

74 Video Tutorials · 16 Modules · 16 Challenges · 18 April 2024

Anshuman Singh
Co-founder **SCALER**

CERTIFICATE OF EXCELLENCE BY SCALER

CERTIFICATE OF EXCELLENCE

THIS CERTIFICATE IS AWARDED TO

SCALER Topics

CHARANYA KANAMARLAPUTI

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advocated Concepts**
Following are the the learning items, which are covered in this tutorial

74 Video Tutorials · 16 Modules · 16 Challenges · 18 April 2024

Anshuman Singh
Co-founder **SCALER**

CERTIFICATE OF EXCELLENCE BY SCALER