

DATABASE MANAGEMENT SYSTEM

PROJECT TITLE

RESTAURANT MANAGEMENT

Submitted by

PORAPU PAVAN SAI	A22126552172
KOLLI YAMINI	A22126552153
SEEPANA ANKITHA	A22126552178
KOYYAPU MADHURAM KARTHIKEYA	A22126552158
GUNDAPU DEVAYANI	A22126552213



DEPARTMENT OF CSE (AI & ML)

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(UGC AUTONOMOUS)

(Permanently Affiliated to AU, Approved by AICTE, and Accredited by NBA & NAAC with an 'A+' Grade)

SANGIVALASA, Bheemili Mandal, VISAKHAPATNAM – 531162

2023 - 2024

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(UGC AUTONOMOUS)

*(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & N with A+)
Sangivalasa, Bheemili Mandal, Visakhapatnam -531162. (A.P)*



BONAFIDE CERTIFICATE

This is to certify that the DBMS project report titled **RESTAURANT MANAGEMENT SYSTEM** is being submitted by P Pavan Sai (A22126552172), K Yamini (A22126552153), S Ankitha (A22126552178), M Karthikeya (A22126552158), Devayani (A2212655213) of III/IV B.Tech I semester Computer science and engineering(AI &ML) is a record bonafide work carried out by them. The results embodied in this report have not been submitted to any other University for the award of any degree.

Reviewer

Mrs. B Sailaja
Assistant Professor
Dept. of CSE (AI&ML, DS)

Head of the Department

Dr. K Selvani Deepthi
Professor
Dept. of CSE (AI&ML, DS)

CONTENTS

1. REQUIREMENT ANALYSIS	4
2. CONCEPTUAL DESIGN -ER DIAGRAM	5
3. CONVERTING ER DIAGRAM TO RELATIONS.....	7
4. IDENTIFYING THE INTEGRITY CONSTRAINTS.....	10
5. CREATING TABLES WITH INTEGRITY CONSTRAINTS	12
6. DML OPERATIONS	15
7. IMPLEMENTING QURIES	17
8. DATABASE CONNECTIVITY.....	23
9. PROJECT OVERVIEW	25
10.CONCLUSION	30

REQUIREMENT ANALYSIS

User Credentials :

- Manage user accounts in the application and log in.
- User can have password for security purpose.
- Address for showing nearby available restaurants.
- Phone number and email for contacting purpose.

Catalogue Management :

- The system should maintain a catalogue of food items available for ordering that are present in nearby restaurants.
- Each item should have attributes such as NAME, PRICE, DESCRIPTION so that it will be easy for the customer to access.

Orders :

- Users should be able to browse the catalogue and add items to their orders.
- Users can specify quantities and any customizations for each item.
- Orders should be associated with the user's account and unique customer identification number.

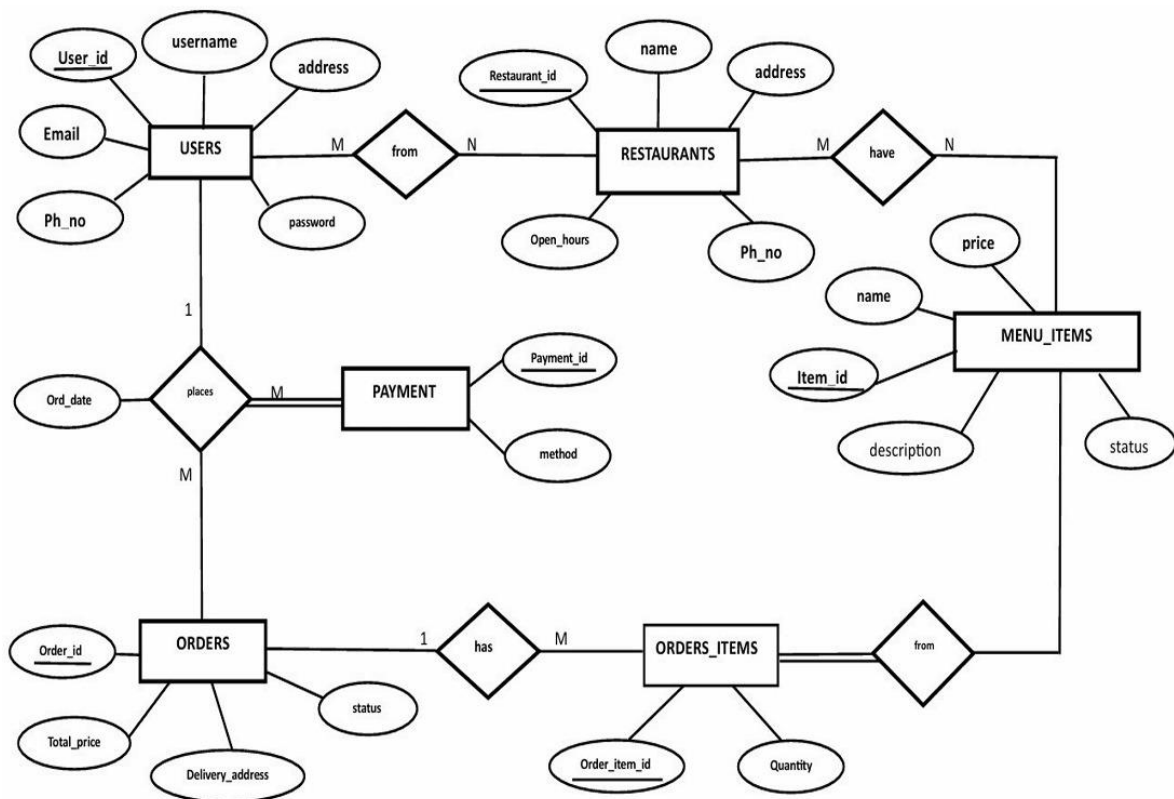
Management :

- The system should maintain records for regular customers, including names and addresses.
- New customers should be prompted to provide necessary information to establish an account. DELIVERY STATUS Checking the delivery status of order and updating it. SECURITY The system should implement security measures to protect user data and payment information. ORDER FULFILLMENT Orders should be shipped together whenever possible to minimize shipping costs and complexity.

User Credentials:

- User can create accounts in the application and log in.
- User can have password for security purpose.
- Address for showing nearby available restaurants.
- phone number and email for contacting purpose.

CONCEPTUAL DESIGN -ER DIAGRAM



Relationships

1. USERS – RESTAURANTS

- **M relationship:** Users can be associated with multiple restaurants, and a restaurant can be associated with multiple users. This relationship indicates that users might have favourite restaurants or make orders from multiple restaurants.

2. RESTAURANTS - MENU_ITEMS

- **M relationship:** A restaurant can have multiple menu items, and a menu item can belong to multiple restaurants. This relationship shows that different restaurants might offer the same or similar menu items.

3. USERS – ORDERS

- **1 relationship:** A user can place multiple orders, but each order is placed by one user. This relationship indicates the ordering activity of users.

4. ORDERS – PAYMENT

- **M:1 relationship:** Each order can have one payment associated with it, and a payment can be linked to multiple orders. This relationship captures the payment information for the orders.

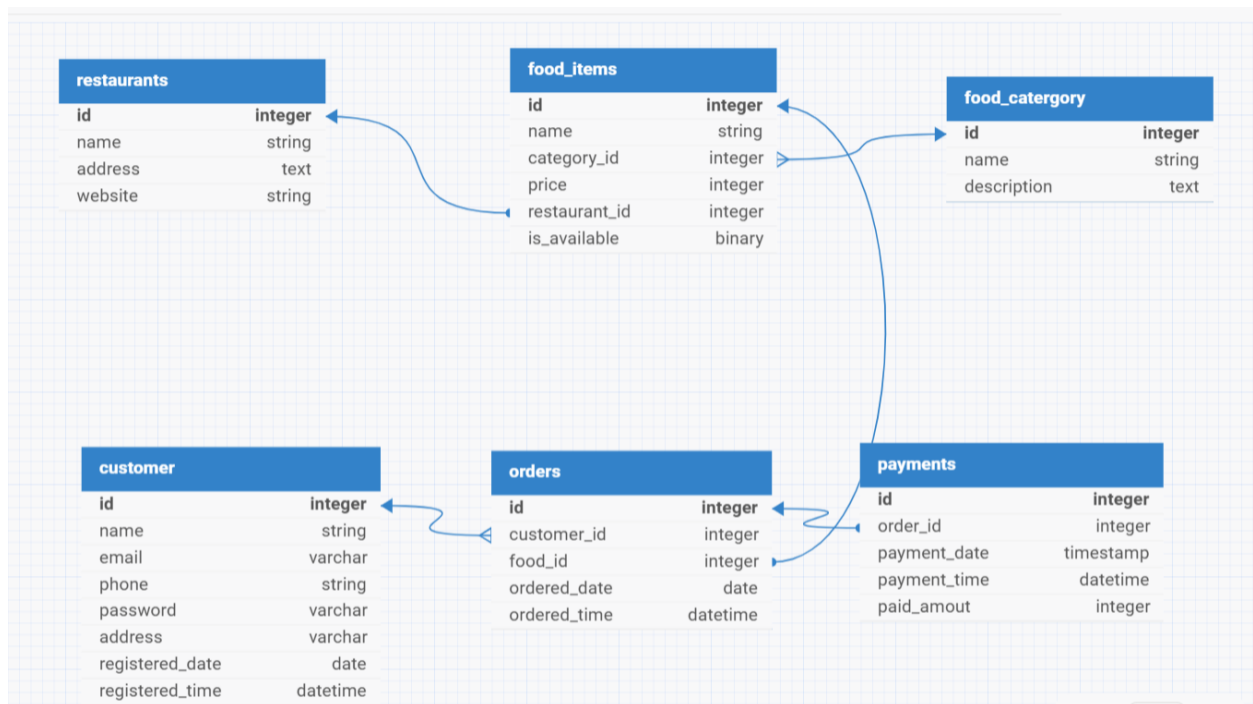
5. ORDERS - ORDERS_ITEMS

- **1 relationship:** An order can contain multiple order items, and each order item is part of one order. This relationship details the specific items within an order.

6. ORDERS_ITEMS - MENU_ITEMS

- **M relationship:** Order items can reference multiple menu items, and menu items can be part of multiple order items. This relationship shows the connection between items in orders and the menu items offered.

CONVERTING ER DIAGRAM TO RELATIONS



Restaurants Table Schema

```
CREATE TABLE IF NOT EXISTS `restaurants` (  
  `id` int AUTO_INCREMENT NOT NULL UNIQUE,  
  `name` varchar(255) NOT NULL,  
  `address` text NOT NULL,  
  `website` varchar(255) NOT NULL,  
  PRIMARY KEY (`id`)  
);
```

Food Items Table Schema

```
CREATE TABLE IF NOT EXISTS `food_items` (  
  `id` int AUTO_INCREMENT NOT NULL UNIQUE,  
  `name` varchar(255) NOT NULL,
```

```
`category_id` int NOT NULL,  
`price` int NOT NULL,  
`restaurant_id` int NOT NULL,  
`is_available` binary(1) NOT NULL,  
PRIMARY KEY (`id`)  
);
```

Food Categories Table Schema

```
CREATE TABLE IF NOT EXISTS `food_category` (  
`id` int AUTO_INCREMENT NOT NULL UNIQUE,  
`name` varchar(255) NOT NULL,  
`description` text NOT NULL,  
PRIMARY KEY (`id`)  
);
```

Customer Table Schema

```
CREATE TABLE IF NOT EXISTS `customer` (  
`id` int AUTO_INCREMENT NOT NULL UNIQUE,  
`name` varchar(255) NOT NULL,  
`email` varchar(255) NOT NULL,  
`phone` varchar(255) NOT NULL,  
`password` varchar(255) NOT NULL,  
`address` varchar(255) NOT NULL,  
`registered_date` date NOT NULL,  
`registered_time` datetime NOT NULL,  
PRIMARY KEY (`id`)  
);
```


Orders Table Schema

```
CREATE TABLE IF NOT EXISTS `orders` (  
  `id` int AUTO_INCREMENT NOT NULL UNIQUE,  
  `customer_id` int NOT NULL,  
  `food_id` int NOT NULL,  
  `ordered_date` date NOT NULL,  
  `ordered_time` datetime NOT NULL,  
  PRIMARY KEY (`id`)  
);
```

Payments Table Schema

```
CREATE TABLE IF NOT EXISTS `payments` (  
  `id` int AUTO_INCREMENT NOT NULL UNIQUE,  
  `order_id` int NOT NULL,  
  `payment_date` timestamp NOT NULL,  
  `payment_time` datetime NOT NULL,  
  `paid_amout` int NOT NULL,  
  PRIMARY KEY (`id`)  
);
```

IDENTIFYING THE INTEGRITY CONSTRAINTS

Create a constraint:

- At the same time as the table is created.
- After the table has been created.

Define a constraints at the column or table level.

Syntax:

- CREATE TABLE [schema.]table
 (column datatype [DEFAULT expr]
 [column_constraint],
 ...
 [table_constraint][,...]);

Column constraint level:

- Column [constraint constraint_name] constraint_type,

Constraint types:-

NOT NULL:

- Ensures that null values are not permitted for the column.
- Example:
 Create table users (
 User_name varchar2(20) not null);

UNIQUE:

- The value should be unique.
- Example:
 Create table restaurants(
 Restaurant_id number unique);

PRIMARY KEY:

- The value should be unique. and not null for a column.
- Example:
 Create table menu_items(
 Item_id number primary key);

FOREIGN KEY:

- defines the column in the child table at the table constraint level.
- Example:
Create table orders(
Order_id number primary key,
number,
User_id
Foreign key(user_id) references users(user_id));

CHECK :

- defines a condition that each row must satisfy.
- Example:
Create table menu_items(
Status varchar2(14)
Check(status='available' or status='not available'));

CREATING TABLES WITH ALL INTEGRITY CONSTRAINTS

Restaurants Table

- **Query:** desc restaurants;

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
name	varchar	NO		NULL	
open_time	time	YES		NULL	
close_time	time	YES		NULL	
address	text	NO		NULL	
website	varchar	NO		NULL	
phone	varchar	YES		NULL	

Food Items Table

- **Query:** desc food_items;

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
name	varchar	NO		NULL	
type	varchar	YES		NULL	
Category_id	int	NO	MUL	NULL	
price	int	NO		NULL	
restaurant_id	int	NO	MUL	NULL	

Food Categories Table

- **Query:** desc food_category;

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
name	varchar	NO		NULL	
description	text	NO		NULL	

Customer Table

- **Query:** desc customers;

Field	Type	Null	Key	Default	Extra
id	Int	NO	PRI	NULL	auto_increment
name	Varchar	NO		NULL	
email	Varchar	NO		NULL	
phone	Varchar	NO		NULL	
password	Varchar	NO		NULL	
registered_date	date	YES		NULL	
Registered_time	time	YES		NULL	

Orders Table

- **Query:** desc orders;

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
Customer_id	int	NO	MUL	NULL	
Food_id	int	NO	MUL	NULL	

restaurant_id	int	YES		NULL	
ordered_date	date	YES		NULL	
Ordered_time	Time	YES		NULL	

Payments Table

- **Query:** desc payments;

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
order_id	int	NO	MUL	NULL	
payment_date	timestamp	NO		NULL	
paid_amout	int	NO		NULL	

DML OPERATIONS

DATA MANIPULATION LANGUAGE:

A DML statement is executed when you:

- Add new rows to a table o Modify existing rows in a table
- Remove existing rows from a table A transaction consists of a collection of DML statements that form a logical unit of work.

ADDING A NEW ROW IN A TABLE:

INSERT STATEMENT:

Insert command writes new rows of data into a table.

- Only one row is inserted at a time with this syntax.
- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally list the columns in the INSERT clause

UPDATE STATEMENT:

- Modify existing rows with the UPDATE statement.
- Update more than one row at a time, if required.
SYNTAX: UPDATE table SET column = value [, column = value, ...] [WHERE condition];
- Updating Rows in a Table
 - Specific row or rows are modified when you specify the WHERE clause.
 - All rows in the table are modified if you omit the WHERE clause.

DELETE STATEMENT:

You can remove existing rows from a table by using the DELETE statement.

SYNTAX: DELETE [FROM] table [WHERE condition];

Deleting Rows from a Table

- Specific rows are deleted when you specify the WHERE clause.
- All rows in the table are deleted if you omit the WHERE clause.
- Delete a row from the table

DATABASE TRANSACTIONS:

Consist of one of the following statements:

- Begin when the first executable SQL statement is executed
- End with one of the following events:
 - COMMIT or ROLLBACK is issued
 - DDL or DCL statement executes (automatic commit)
 - User exits
 - System crashes
- Advantages of COMMIT and ROLLBACK Statements
- Ensure data consistency
- Preview data changes before making changes permanent • Group logically related operations

COMMIT:

- Makes all pending changes permanent.
- Commit command is used to permanently save any transaction into the database.

SAVEPOINT:

- Allows a rollback to the savepoint marker.
- Create a marker in a current transaction by using the SAVEPOINT statement.
- Roll back to that marker by using the ROLLBACK TO SAVEPOINT statement.

ROLLBACK:

- Discards all pending data changes.
- This command restores the database to last committed state.

IMPLEMENTING QURIES

1. retrieve data of a user (name, mobile number) with user_id=1

```
SQL> select username, phone_number  
      from users  
      where user_id=1;  
retrieve data of a user (name, mobile number) with user_id=1
```

2. retrieving data from orders table based upon the condition that price is null.

```
SQL>select order_id,total_price  
      from orders  
      where total_price is null;
```

3.retrieve data from orders based on condition order_id<>4 and <5.

```
SQL> select total_price,order_id  
      from payment  
      where order_id<5 and order_id<>4;
```

4.retrieve data from orders where price is in 450,550 values.

```
SQL> select order_id,total_price  
      from orders  
      where total_price in (450,550);
```

5.retrieve data from menu_items(item_name,price) upto item_id less than 5 in descending order.

```
SQL>select item_name,price from menu_items  
      where item_id<5  
      order by price desc;
```

6.data from users where name starts with letter i.

```
SQL>select username  
      from users  
      where username like 'I%';
```

7.data from users where name ends with letter i.

```
SQL>select username  
      from users  
      where username like '%i';
```

8.retrieve total_price and 10 times of total price with order_id<4 and display in asc order.

```
SQL>select total_price, total_price*10 as tp
      from orders
      where order_id<4
      order by tp;
```

9.retrieve data in upper case with user_id=10.

```
SQL> select upper(address)
      from users
      where user_id=10;
```

10.retrieve data in lower case with user_id=10.

```
SQL> select lower(address)
      from users
      where user_id=10;
```

11.data from menu_items with item_id=5 using initcap function.

```
SQL>select initcap(item_name)
      from menu_items
      where item_id=5;
```

12.total money payed by user with id=1.

```
SQL> select sum(total_price)
      from payment
      where order_id=1;
```

13.retrieve name from restaurants with id=5

```
SQL> select name
      from restaurants
      where restaurant_id=5;
```

14.price from orders where id=4.

```
SQL>select total_price
      from orders
      where user_id=4;
```

15.using count function-to count no. of items from restaurant id=3.

```
SQL> select count(item_id)
from menu_items
where restaurant_id=3;
```

16.by using distinct keyword to find without duplicates.

```
SQL> select distinct item_name,price
from menu_items
where price between 400 and 500;
```

17.adding values with NULL.

```
SQL>select total_price,total_price+null
      from orders where order_id=1;
```

18.Retrieve the total revenue generated by each restaurant:

```
SQL> SELECT restaurant_id, SUM(total_price) AS total_revenue
FROM Orders
GROUP BY restaurant_id;
```

19.What is the price after formatting(as specified)of the menu item with item_id 16?

```
SQL>select to_char(price,'$99,999') price
from menu_items
where item_id=16;
```

20.Retrieve the total number of orders for each user

```
SQL>SELECT user_id, COUNT(*) AS total_orders
FROM Orders
GROUP BY user_id
order by user_id;
```

21.combing 2 columns using concatenation symbol.

```
SQL>select item_name || ' ' || price as item_price
      from menu_items
      where item_id<5;
```

22.retrieve item name from menu_items on condition.

```
SQL> select item_name
      from menu_items
      where item_id<5 or status='available' and restaurant_id in (13,23,3,16,48);
```

23.Retrieve the usernames of users who have not provided their usernames, displaying 'not mentioned' for those users whose usernames are null

```
SQL> select username,nvl(to_char(username),'not mentioned')  
      from users  
      where username is null;
```

24.query for getting distinct price for items that are having item_id<10

```
SQL> select distinct price  
      from menu_items  
      where item_id<10;
```

25.display with price and price with adding 50 to it.

```
SQL> select total_price as tp,total_price+50  
      from orders  
      where order_id=4;
```

26.display with price and price with adding 50 after multiplying with 2 to it.

```
SQL> select total_price,2*total_price+50  
      from orders  
      where order_id=4;
```

28.using substring function to get part of a string.

```
SQL> select substr(username,1,4) as substring  
      from users  
      where user_id=1;
```

29.Retrieve the latest order date upto 19 users

```
SQL>SELECT user_id, MAX(order_date) AS latest_order_date  
      FROM Order_Date  
      where user_id <20  
      GROUP BY user_id  
      order by user_id ;
```

30.example to show usage of mod function.

```
SQL> select item_id,item_name,price,mod(price,15)  
      from menu_items  
      where item_id=10;
```

31.example to show usage of lpad function.

```
SQL> select (phone_number),lpad(phone_number,12,'*')
       from users
       where user_id=9;
```

32.example for showing usage of concat,length,instr function.

```
SQL> select username,
       concat(user_id,username),length(username),instr(username,'a')
       from users
       where user_id=1;
```

34.Retrieve all orders with a total price greater than \$100 and status as 'pending':

```
SQL> SELECT * FROM Orders
       WHERE total_price > 100 AND status = 'pending';
```

35.retrieve data from menu_items with id=2,3,4,6,7,9.

```
SQL> select ITEM_Name
       from menu_items
       where item_id not in (1,5,8) and item_id<10;
```

36.Count the total number of orders placed.

```
SQL>select count(*) as total_orders
       from order_date;
```

37.details of the first order placed by a user

```
SQL> select orders.order_id,users.username,restaurants.name
       as restaurant_name,orders.total_price,orders.status
       from orders, users, restaurants
       where orders.user_id = users.user_id
       and orders.restaurant_id = restaurants.restaurant_id and rownum =1;
```

38. What are the menu items that are offered at both Restaurant ID 14 and Restaurant ID 39?

```
SQL>select item_name
       from menu_items
       where restaurant_id=14 intersect select item_name from menu_items       where
```

```
restaurant_id=39;
```

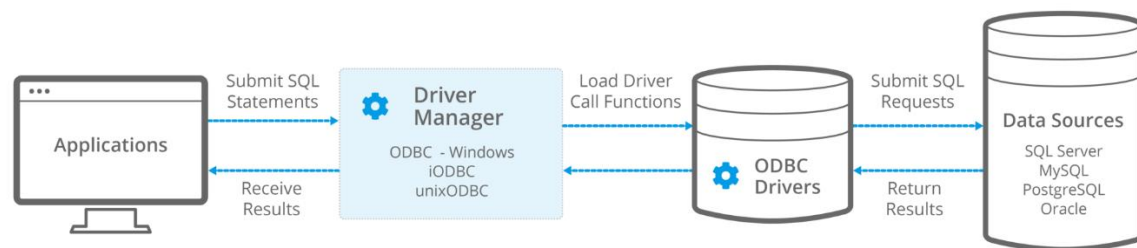
39. retrieve the total number of orders placed at each restaurant

```
SQL> select restaurant_id, count(*) as total_orders  
      from orders  
      group by restaurant_id  
      order by restaurant_id;
```

40. What are the menu items that are offered at both Restaurant ID 14 and Restaurant ID 39?

```
SQL> select item_name  
      from menu_items  
      where restaurant_id in (select restaurant_id from restaurants where  
Bazaar'); name='Biryani
```

DATABASE CONNECTIVITY



Database connectivity is a critical aspect of modern application development. It enables applications to interact with databases to store, retrieve, and manipulate data. Effective database connectivity ensures that applications can access and manage data efficiently, securely, and reliably.

Importance of Database Connectivity

Database connectivity is essential for various reasons:

- **Data Management:** Applications often need to perform CRUD (Create, Read, Update, Delete) operations on data. Database connectivity facilitates these operations.
- **Performance:** Efficient connectivity ensures that data access is fast and responsive, which is crucial for user experience.
- **Security:** Secure database connections protect sensitive data from unauthorized access and breaches.
- **Scalability:** Properly managed connectivity allows applications to handle increased loads and scale effectively.

Types of Database Connectivity

1. Direct Connectivity:

- **ODBC (Open Database Connectivity):** A standard API for accessing database management systems (DBMS). It is independent of DBMS and provides a consistent interface.
- **JDBC (Java Database Connectivity):** An API that allows Java applications to interact with databases. It is part of the Java Standard Edition platform.

2. Middleware:

- **Application Servers:** Middleware like Apache Tomcat, JBoss, or IBM WebSphere can manage database connections for applications.
- **Database Connection Pooling:** Middleware can manage a pool of database connections, improving performance and resource utilization.

3. Web APIs:

- **RESTful APIs:** Applications can interact with databases via web services using HTTP requests.
- **GraphQL:** An alternative to REST that allows clients to request specific data, reducing the amount of data transferred.

Database Connectivity Methods

1. Connection Strings:

A connection string is used to specify the details needed to connect to a database, including server address, database name, user credentials, and other parameters.

2. Connection Pools:

A connection pool maintains a pool of database connections and reuses them, reducing the overhead of establishing new connections.

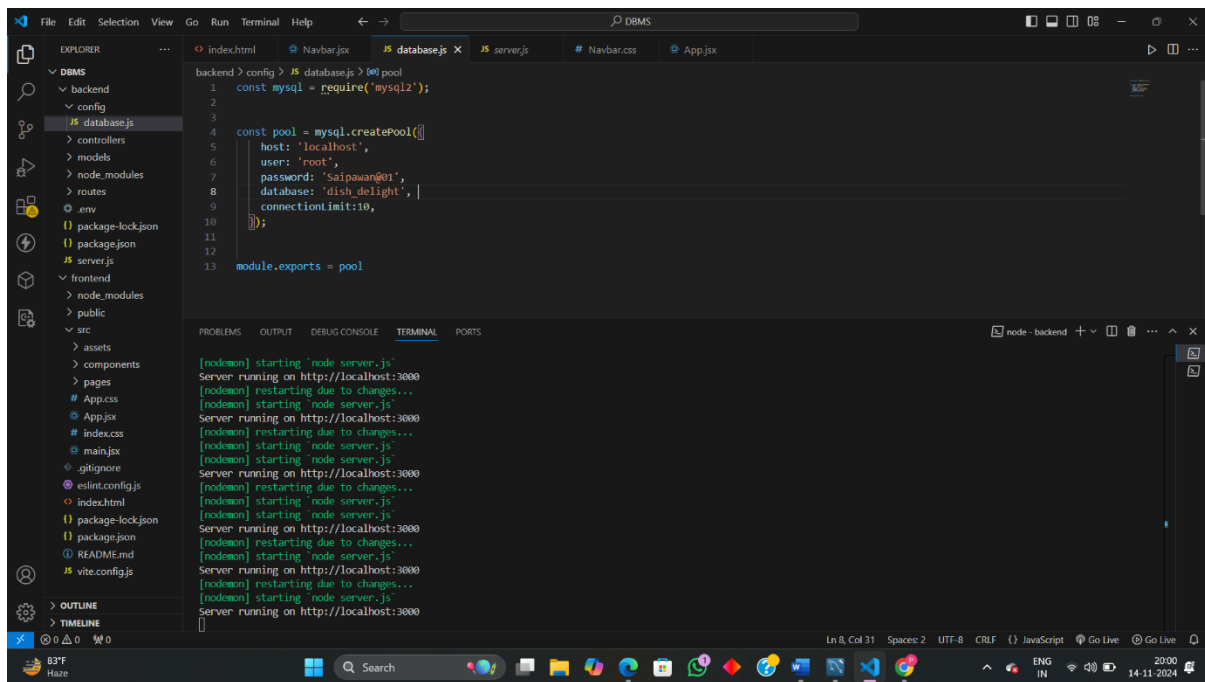
3. ORM (Object-Relational Mapping):

ORMs like Hibernate, Entity Framework, or Django ORM abstract database interactions, allowing developers to work with data as objects rather than SQL queries.

Database connectivity is a foundational element of modern application development, affecting performance, security, and scalability. By understanding the various methods and best practices for database connectivity, developers can create robust, efficient, and secure applications. Ensuring proper implementation of connectivity mechanisms and security measures is crucial for the successful operation of any application that relies on database interactions.

PROJECT OVERVIEW

we have developed a restaurant management system to demonstrate our proficiency in database management. This system is designed to handle various aspects of restaurant operations, providing a seamless interface for managing menu items, customer orders, and other essential tasks.



Frontend: ReactJS

The frontend of the application is built using ReactJS, a popular JavaScript library for building user interfaces. ReactJS offers a component-based architecture that allows for the creation of reusable UI components, making the development process more efficient and scalable. In this project, ReactJS is used to create an intuitive and responsive user interface that provides restaurant staff with easy access to the system's functionalities. Key components include:

- **Menu Management:** Allows users to add, edit, and delete menu items. Each menu item includes details such as name, description, price, and category.
- **Order Management:** Enables users to take new orders, view existing orders, and update order statuses. This feature ensures that the restaurant can efficiently manage customer orders from creation to fulfilment.
- **Dashboard:** Provides an overview of the restaurant's operations, including sales statistics, popular menu items, and current orders. This helps in making informed business decisions.

Backend: Node.js and MySQL

The backend of the application is developed using Node.js, a runtime environment that allows for the execution of JavaScript code server-side. Node.js is chosen for its non-blocking, event-driven architecture, which makes it ideal for handling multiple requests

simultaneously. The backend server is responsible for processing requests from the frontend, interacting with the database, and returning the appropriate responses.

MySQL is used as the database management system for storing all the data related to the restaurant, including menu items, orders, and user information. The connection to the MySQL database is managed using the `mysql2` library in Node.js, which provides a straightforward API for executing SQL queries and managing database connections.

The backend code is organized into several modules, each responsible for a specific aspect of the system:

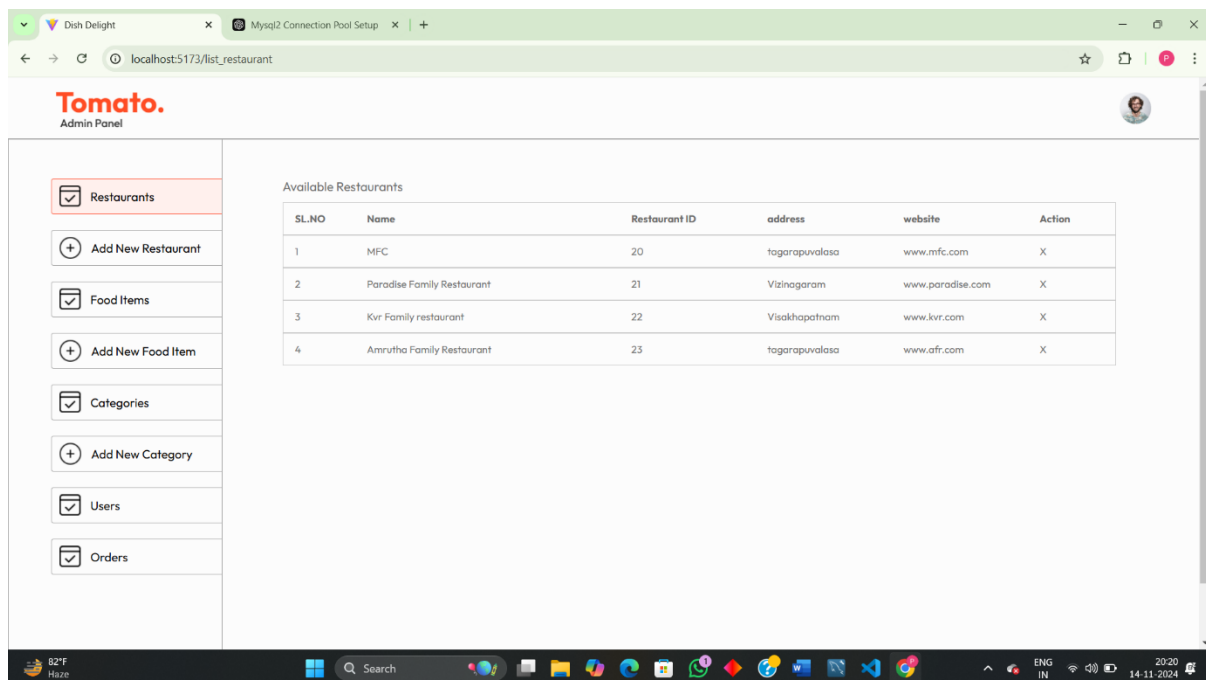
- **Controllers:** Handle the business logic and respond to HTTP requests from the frontend.
- **Models:** Define the structure of the database tables and provide methods for querying and manipulating data.
- **Routes:** Define the endpoints for the RESTful API, mapping HTTP requests to the appropriate controller functions.
- **Config:** Contains configuration settings for the database connection, including the creation of a connection pool to improve performance and resource management.

The integration between the frontend and backend is achieved through a RESTful API, where the frontend makes HTTP requests to the backend to perform CRUD operations on the data.

Frontend Images:

Restaurants Table:

- Retrieving all available restaurants through which user can order food from database.



Add Restaurant Option:

- A simplified interface to register a new restaurant to the database.

The screenshot shows the 'Tomato. Admin Panel' with a sidebar on the left containing navigation links: Restaurants, Add New Restaurant (highlighted), Food Items, Add New Food Item, Categories, Add New Category, Users, and Orders. The main content area displays a form for adding a new restaurant with the following fields: Restaurant Name, Open Time, Close Time, Restaurant Address, Website, and Contact NO. Each field has a placeholder text 'type here'. An 'Add' button is located at the bottom of the form.

Food Items Table:

- Retrieving all available food items which user can order from database.
- Added option to delete the food item from the database.

The screenshot shows the 'Tomato. Admin Panel' with the 'Food Items' section selected in the sidebar. The main content area displays a table titled 'Available Food Items' with the following data:

SL.NO	Name	Item Id	price	type	category	restaurant_id	Action
1	Hyderabad Spl Dum Biryani	21	260	non-veg	Biryani	20	X
2	Prawns Biryani	22	343	non-veg	Biryani	20	X
3	Veg Biryani	23	218	veg	Biryani	20	X
4	Fry Piece Biryani	24	269	non-veg	Biryani	20	X
5	Chicken Mughlai Biryani	25	322	non-veg	Biryani	20	X
6	Mutton Pot Biryani	33	339	non-veg	Biryani	21	X
7	Special Mandi	34	619	non-veg	Biryani	21	X
8	Fish Biryani	39	299	non-veg	Biryani	21	X
9	Kaju Biryani	52	230	veg	Biryani	23	X
10	Egg Manchurian	26	218	non-veg	Starters	20	X
11	Chicken Roast	27	266	non-veg	Starters	20	X
12	Chicken Majestic	40	299	non-veg	Starters	21	X
13	Chicken Roast Boneless	41	189	non-veg	Starters	21	X

Add Food Item Option:

- A simplified interface to add new food items to the database.
- Added option to delete restaurant from the database.

The screenshot shows the 'Tomato Admin Panel' interface. On the left is a sidebar menu with options: Restaurants, Add New Restaurant, Food Items, Add New Food Item (highlighted), Categories, Add New Category, Users, and Orders. The main content area is titled 'Add New Food Item' and contains a form with the following fields: Item Name, Type, Category Id, Price, and Restaurant Id. Each field has a placeholder text 'type here'. At the bottom of the form is a black 'Add' button. The browser's address bar shows 'localhost:5173/add_foods'.

Categories Table:

- Retrieving all available categories from database.
- Added option to delete the category from the database.

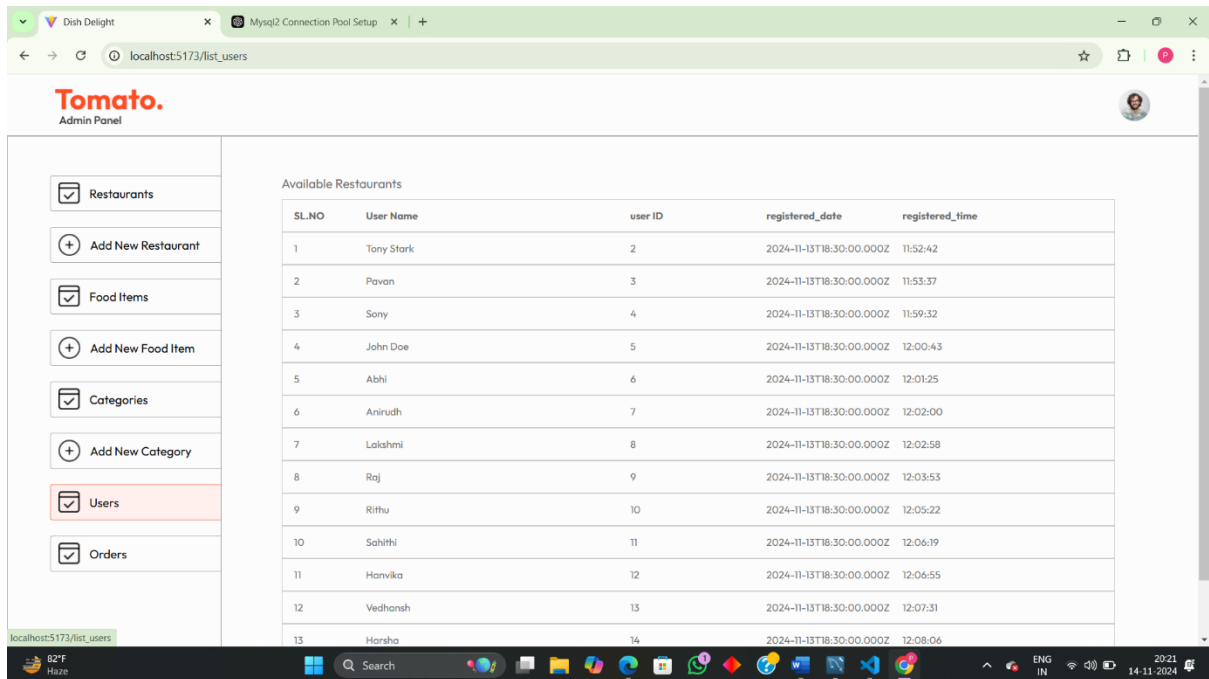
The screenshot shows the 'Tomato Admin Panel' interface. On the left is a sidebar menu with options: Restaurants, Add New Restaurant, Food Items, Add New Food Item, Categories (highlighted), Add New Category, Users, and Orders. The main content area is titled 'Available Categories' and displays a table with the following data:

SL.NO	Name	Description	Category ID	Action
1	Biryani	Delicious to eat	12	X
2	Starters	Start with spicy to eat more	13	X
3	Fried Rice	Light to take	14	X
4	Curry	To experience Best Como try it Now!	15	X
5	Breakfast	good food lasts good thoughts	16	X
6	Bevarages	Cool and Pleasant to drink	17	X
7	FastFoods	Evening with delightful flavors	18	X
8	Meals	Fulfillment of eating	19	X
9	Pizzas	Hot and succulent at the same time	20	X

The browser's address bar shows 'localhost:5173/list_category'.

User Table:

- Retrieving all available users from database.

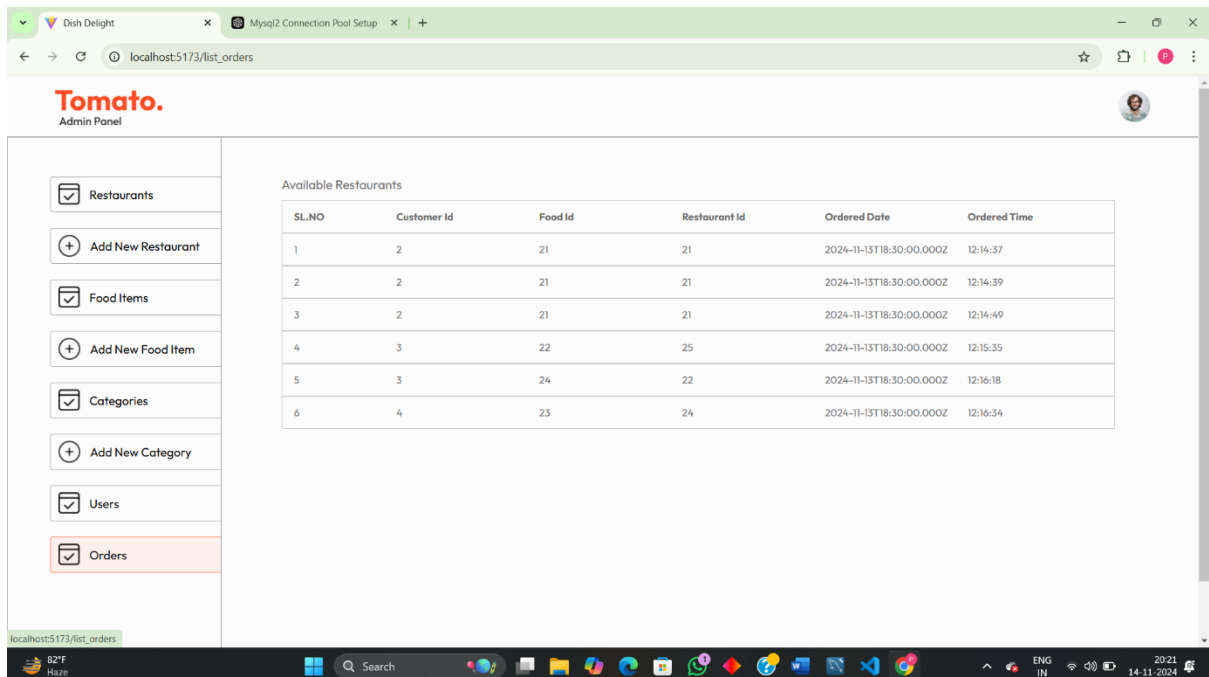


The screenshot shows the Tomato Admin Panel interface. On the left is a sidebar with navigation links: Restaurants, Add New Restaurant, Food Items, Add New Food Item, Categories, Add New Category, Users (highlighted), and Orders. The main content area is titled 'Available Restaurants' and displays a table with user information.

SL.NO	User Name	user ID	registered_date	registered_time
1	Tony Stark	2	2024-11-13T18:30:00.000Z	11:52:42
2	Pavan	3	2024-11-13T18:30:00.000Z	11:53:37
3	Sony	4	2024-11-13T18:30:00.000Z	11:59:32
4	John Doe	5	2024-11-13T18:30:00.000Z	12:00:43
5	Abhi	6	2024-11-13T18:30:00.000Z	12:01:25
6	Anirudh	7	2024-11-13T18:30:00.000Z	12:02:00
7	Lakshmi	8	2024-11-13T18:30:00.000Z	12:02:58
8	Raj	9	2024-11-13T18:30:00.000Z	12:03:53
9	Rithu	10	2024-11-13T18:30:00.000Z	12:05:22
10	Sahithi	11	2024-11-13T18:30:00.000Z	12:06:19
11	Hanvika	12	2024-11-13T18:30:00.000Z	12:06:55
12	Vedhansh	13	2024-11-13T18:30:00.000Z	12:07:31
13	Harsha	14	2024-11-13T18:30:00.000Z	12:08:06

Orders Table:

- Retrieving all available users from database.



The screenshot shows the Tomato Admin Panel interface with the 'Orders' link highlighted in the sidebar. The main content area is titled 'Available Restaurants' and displays a table with order information.

SL.NO	Customer Id	Food Id	Restaurant Id	Ordered Date	Ordered Time
1	2	21	21	2024-11-13T18:30:00.000Z	12:14:37
2	2	21	21	2024-11-13T18:30:00.000Z	12:14:39
3	2	21	21	2024-11-13T18:30:00.000Z	12:14:49
4	3	22	25	2024-11-13T18:30:00.000Z	12:15:35
5	3	24	22	2024-11-13T18:30:00.000Z	12:16:18
6	4	23	24	2024-11-13T18:30:00.000Z	12:16:34

CONCLUSION

This restaurant management system project exemplifies my comprehensive skill set in full-stack web development, integrating ReactJS for the frontend, Node.js for the backend, and MySQL as the database. By leveraging ReactJS, I have created a dynamic and responsive user interface that enhances the user experience, allowing restaurant staff to manage menu items, orders, and view operational statistics efficiently. The backend, powered by Node.js, ensures robust and scalable server-side processing, while the MySQL database securely handles the data storage and retrieval operations.