PROJECT REPORT

TOPIC: E-Commerce Bakery Website

Submitted by -

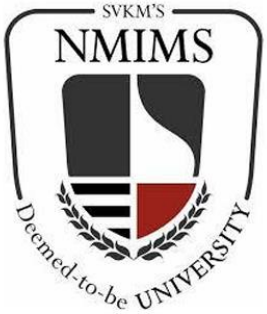
Aayush Patwa (Roll number: D004)

Abhijeet Shahapurkar (Roll number: T002)

Bhushan Harde (Roll number: D023)

Submitted To-

Dr. Shruti Sharma Ma’am



Course: DBMS

Academic Year: 2024-25

Table of Contents

I. Storyline

Sweet Bites Bakery is a local bakery that has been delighting customers with freshly baked goods for over a decade. As the bakery's popularity grows, the owner wants to expand its reach by launching an online platform where customers can browse the menu, place orders, and get their favorite treats delivered to their doorstep. To achieve this, a robust Database Management System (DBMS) is essential to store, manage, and retrieve data efficiently.

The system will require a well-structured relational database to handle multiple entities such as customers, products, orders, payments, and deliveries. Each entity will be represented as a table with appropriate attributes to capture all necessary details. The DBMS will use primary keys to uniquely identify records and foreign keys to establish relationships between tables, ensuring referential integrity.

Additionally, the system should support key functionalities like data insertion, updating, deletion, and retrieval (CRUD operations), enabling smooth management of customer information, product inventory, order tracking, and payment processing. Implementing such a DBMS will enhance data consistency, minimize redundancy through proper normalization, and ensure that the bakery can efficiently scale its online operations while maintaining a seamless customer experience.

II. Components of Database Design

Entities & Attributes:

Users

- user_id (PK)
- name
- email
- password

Products

- product_id (PK)
- product_name
- description
- price
- stock_quantity

Orders

- order_id (PK)
- user_id (FK to Users)
- order_date
- total_cost
- status (e.g., "pending", "shipped", "delivered", "cancelled")

Order_Items

- order_item_id (PK)
- order_id (FK to Orders)
- product_id (FK to Products)
- quantity
- subtotal

Payments

- payment_id (PK)
- order_id (FK to Orders)
- payment_method (e.g., "credit card", "paypal", "bank transfer")
- payment_date
- amount

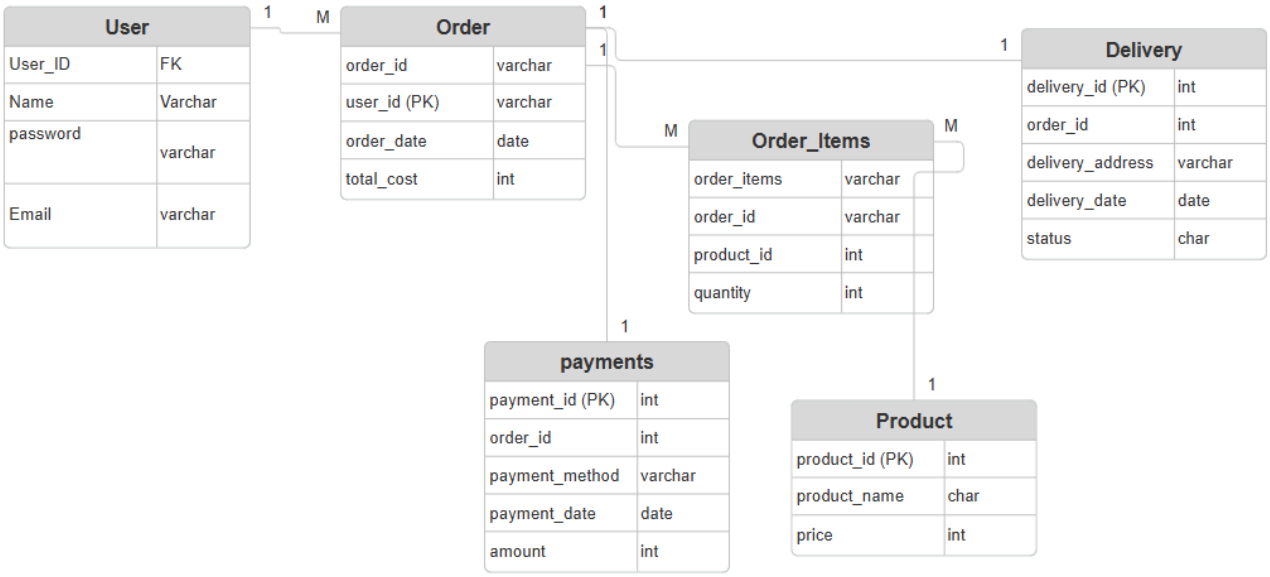
Deliveries (Optional)

- delivery_id (PK)
- order_id (FK to Orders)
- delivery_address
- delivery_date
- status (e.g., "pending", "shipped", "delivered")
- **Contact_Submission**
  - `id` (PK)
  - `name`
  - `email`
  - `message`
  - `submission_date`

Relationships:

1. A user can place many orders (one-to-many).
2. An order is placed by one user (many-to-one).
3. An order can have many order items (one-to-many).
4. An order item belongs to one order (many-to-one).
5. An order item is associated with one product (many-to-one).
6. A product can be part of many order items (one-to-many).
7. An order can have one payment (one-to-one).
8. A payment is associated with one order (one-to-one).
9. An order can have one delivery (one-to-one).
10. A delivery is associated with one order (one-to-one).

III. Entity Relationship Diagram



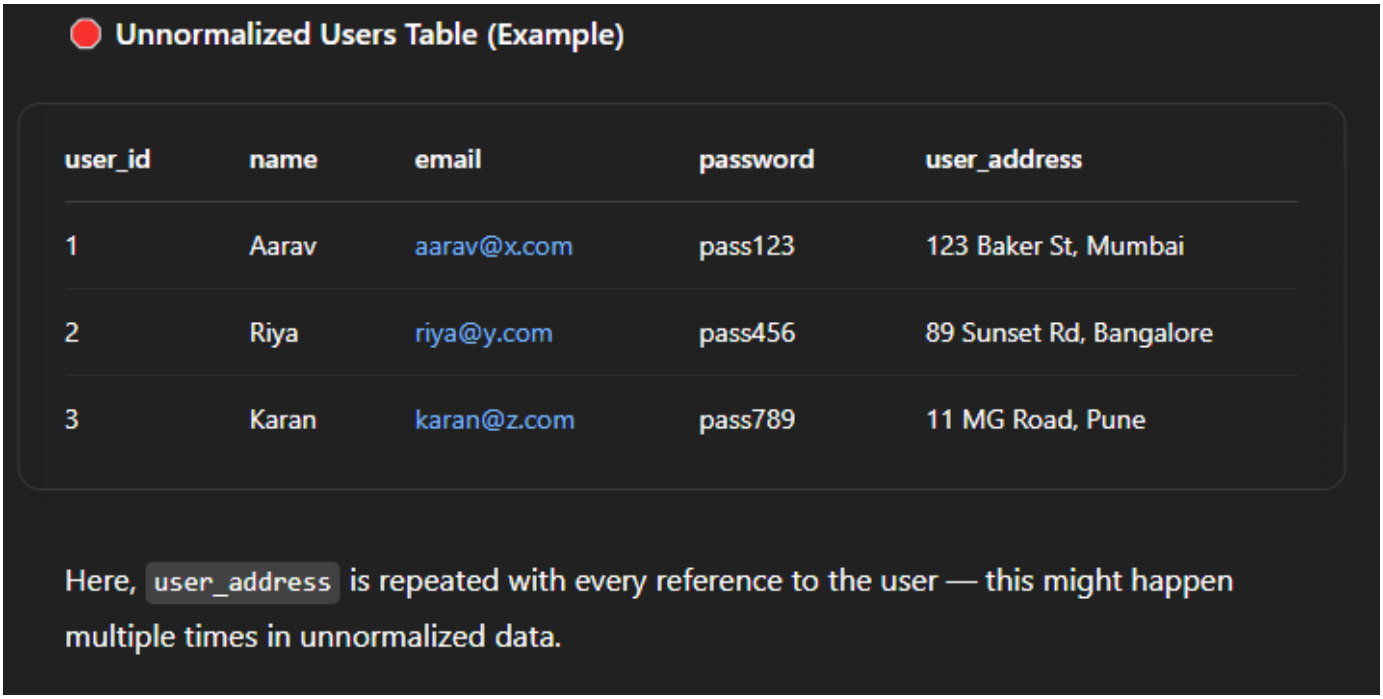
IV. Relational Model

Table Name	Primary Key (PK)	Foreign Key (FK) & relationships	Attributes
Users	UserID	None	Name, Email (Unique), Phone Number, Address

Products	ProductID	None	Product Name, Description, Price, Stock Quantity
Orders	OrderID	UserID → Users(UserID) (M:1)	Order Date, Total Cost, Status (*Pending/Shipped/Delivered/Cancelled*)
Order Items	OrderItemID	OrderID → Orders(OrderID) (M:1), ProductID → Products(ProductID) (M:1)	Quantity, Subtotal
Payments	PaymentID	OrderID → Orders(OrderID) (1:1)	Payment Method (*Credit Card/PayPal/Bank Transfer*), Payment Date, Amount
Deliveries	DeliveryID	OrderID → Orders(OrderID) (1:1)	Delivery Address, Delivery Date, Status (*Pending/Shipped/Delivered*)

V. Normalization

1. **Users**



After normalization-

- User ID (Primary Key)
- Name
- Email
- Password

Similarly we did this for all the further tables

2. **Products**

🔁 Unnormalized Products Table (with redundant baker info)

product_id	product_name	description	price	stock_quantity	baker_name	baker_email
1	Chocolate Cake	Rich chocolate taste	499	10	Aarav	aarav@x.com
2	Vanilla Cupcake	Creamy vanilla base	149	20	Aarav	aarav@x.com

✅ Normalized Structure:

Since every product is tied to a **single home-run baker (who is also the user)**, we **remove redundancy** by storing baker info in the `Users` table.

Final Products Table:

product_id	product_name	description	price	stock_quantity
1	Chocolate Cake	Rich chocolate taste	499	10

- Product ID (Primary Key)
- Product Name
- Description
- Price
- Stock Quantity

3. **Orders**

Unnormalized Orders Table (with repeated user info)

order_id	user_name	user_email	order_date	total_cost	status
101	Aarav	aarav@x.com	2025-04-01	797	Delivered
102	Riya	riya@y.com	2025-04-02	597	Pending

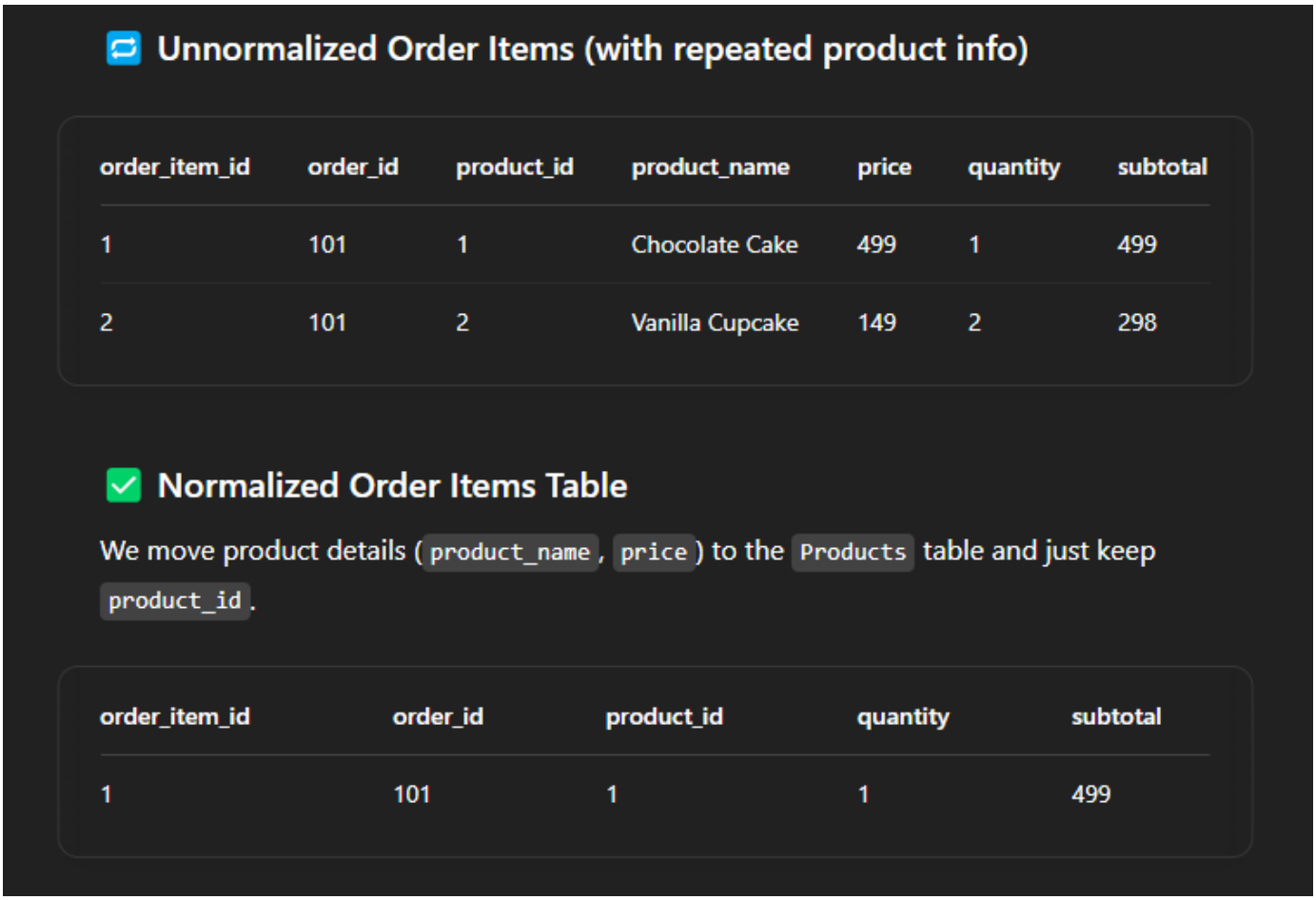
Normalized Orders Table

User info moved to the `Users` table (already exists), only store `user_id`.

order_id	user_id	order_date	total_cost	status
101	1	2025-04-01	797	Delivered

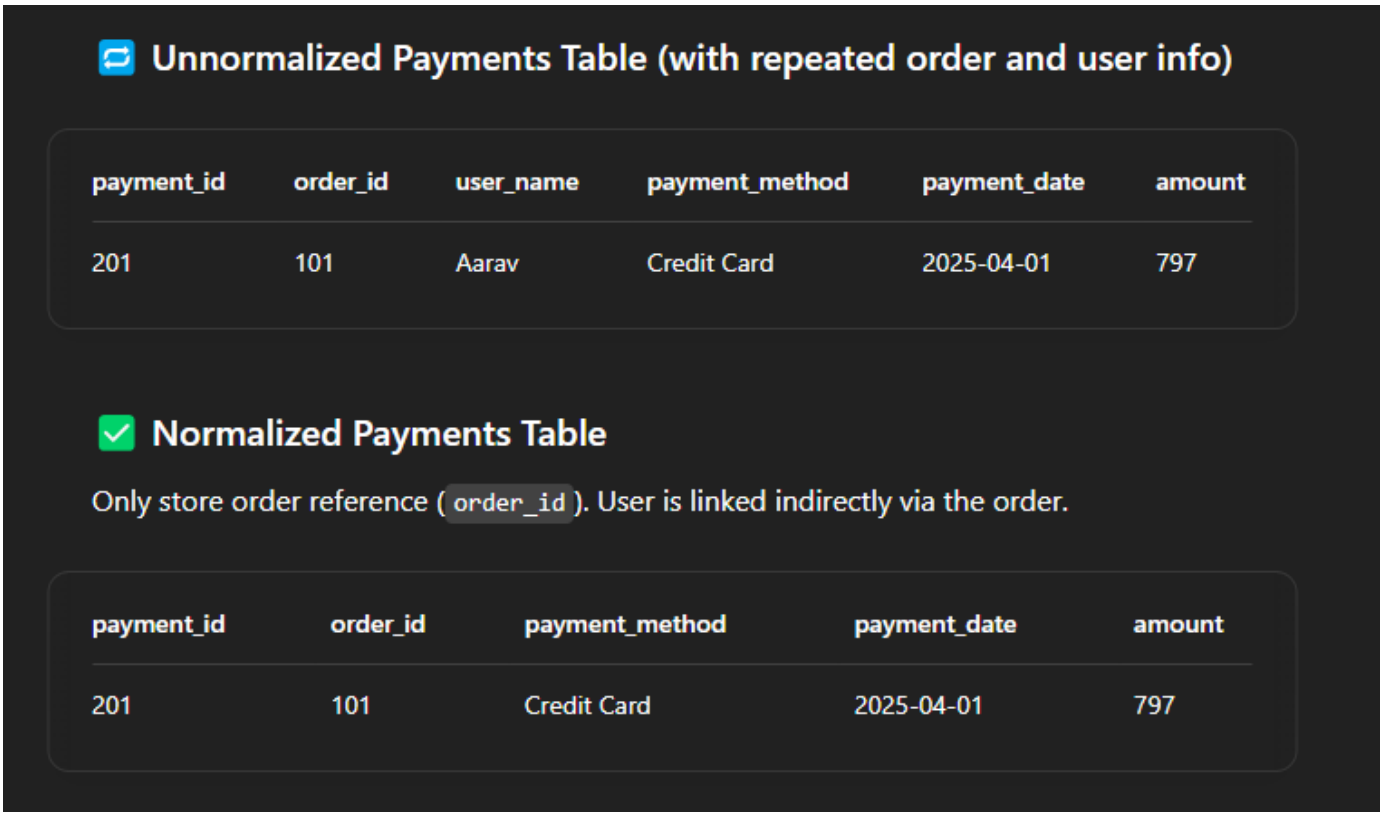
- Order ID (Primary Key)
- User ID (Foreign Key referencing Users)
- Order Date
- Total Cost
- Status (e.g., "pending", "shipped", "delivered", "cancelled")

4. **Order Items**



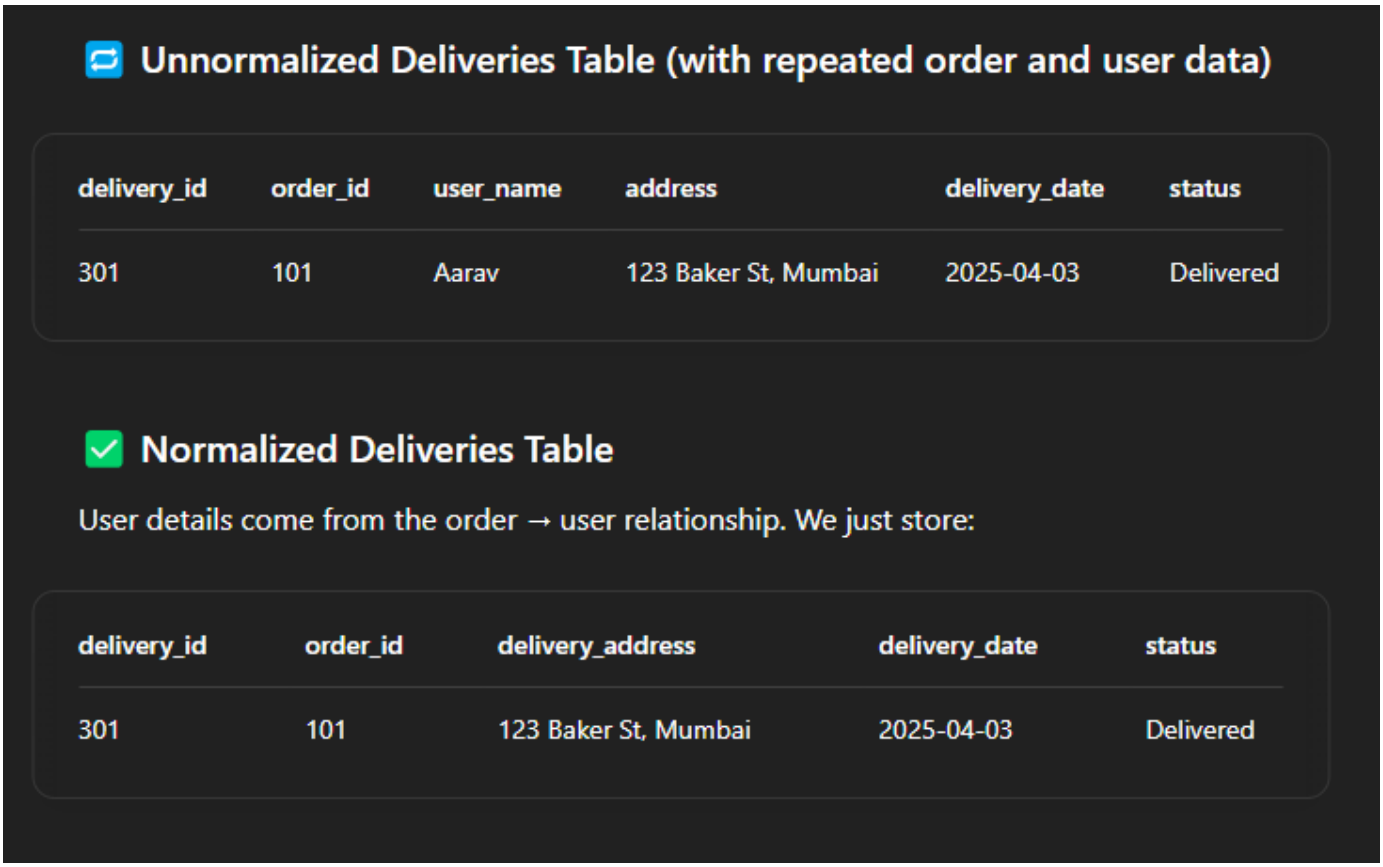
- Order Item ID (Primary Key)
- Order ID (Foreign Key referencing Orders)
- Product ID (Foreign Key referencing Products)
- Quantity
- Subtotal

5. **Payments**



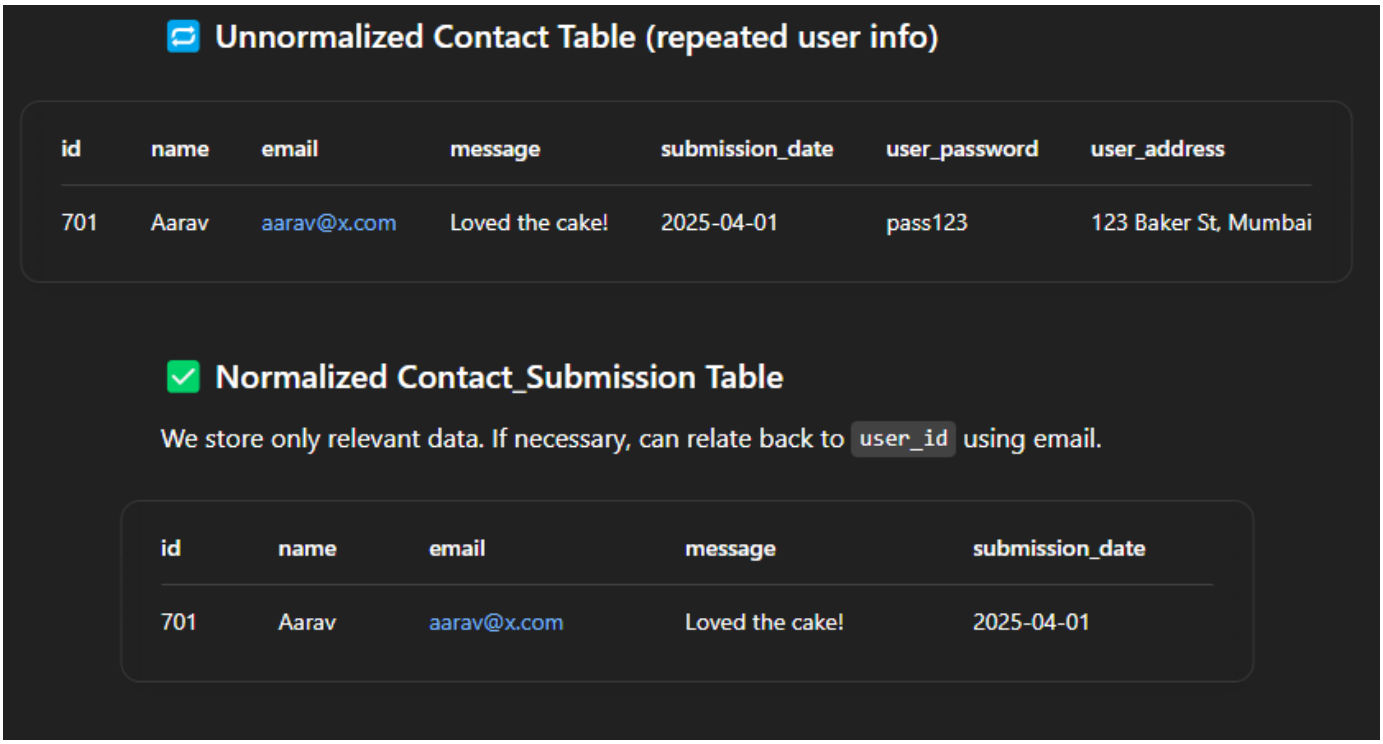
- Payment ID (Primary Key)
- Order ID (Foreign Key referencing Orders)
- Payment Method (e.g., "credit card", "paypal", "bank transfer")
- Payment Date
- Amount

6. **Deliveries**



- Delivery ID (Primary Key)
- Order ID (Foreign Key referencing Orders)
- Delivery Address
- Delivery Date
- Status (e.g., "pending", "shipped", "delivered")

7. **Contact Submission**



- Name
- Email
- Message
- Id (primary key)
- Submission date

VI. SQL Queries

Queries to create tables :

1. User Table-

```
CREATE TABLE user_login (
  id int NOT NULL AUTO_INCREMENT,
  User_Name varchar(45) NOT NULL,
  Password varchar(45) NOT NULL,
  Email varchar(45) NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY User_Name_UNIQUE (User_Name),
  UNIQUE KEY Password_UNIQUE (Password),
  UNIQUE KEY Email_UNIQUE (Email)
)
```

2. Order_item:

```
CREATE TABLE order_items (
  id int NOT NULL AUTO_INCREMENT,
  order_id varchar(50) NOT NULL,
  product_name varchar(100) NOT NULL,
  price decimal(10,2) NOT NULL,
  quantity int NOT NULL,
  created_at timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  KEY fk_order_items_order_id (order_id),
  CONSTRAINT fk_order_items_order_id FOREIGN KEY (order_id) REFERENCES orders
(order_id) ON DELETE CASCADE
)
```

3. Orders :

```
CREATE TABLE orders (
  id int NOT NULL AUTO_INCREMENT,
  order_id varchar(50) NOT NULL,
  user_id int DEFAULT NULL,
  amount decimal(10,2) NOT NULL,
  payment_method varchar(20) NOT NULL DEFAULT 'UPI',
  transaction_id varchar(255) DEFAULT NULL,
  payment_id varchar(100) DEFAULT NULL,
  status varchar(20) NOT NULL,
```

```
  created_at timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  UNIQUE KEY order_id (order_id)
)
```

4. Contact submission:

```
CREATE TABLE contact_submissions (
  id int NOT NULL AUTO_INCREMENT,
  name varchar(100) NOT NULL,
  email varchar(100) NOT NULL,
  message text NOT NULL,
  submission_date timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (id)
)
```

5. Product :

```
CREATE TABLE Products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    product_name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10, 2) NOT NULL,
    stock_quantity INT NOT NULL
);
```

6. Payment :

```
CREATE TABLE Payments (
    payment_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    payment_method ENUM('credit card', 'paypal', 'bank transfer') NOT NULL,
    payment_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    amount DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

7. Delivery:

```
CREATE TABLE Deliveries (
    delivery_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    delivery_address TEXT NOT NULL,
```

```
    delivery_date DATETIME,
    status ENUM('pending', 'shipped', 'delivered') DEFAULT 'pending',
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
      ON DELETE CASCADE
       ON UPDATE CASCADE
);
```

Queries to insert tables:

1. Users:
   ```
   INSERT INTO Users (name, email, password) VALUES
   ('Aarav Mehta', 'aarav@example.com', 'pass123'),
   ('Priya Sharma', 'priya@example.com', 'secure456'),
   ('Ravi Kapoor', 'ravi@example.com', 'mypwd789'),
   ('Anjali Singh', 'anjali@example.com', 'cake321'),
   ('Karan Das', 'karan@example.com', 'bakeit987'),
   ('Neha Joshi', 'neha@example.com', 'sweet456'),
   ('Vikram Rao', 'vikram@example.com', 'passcake'),
   ('Meera Nair', 'meera@example.com', 'meera123'),
   ('Rohit Verma', 'rohit@example.com', 'rohitpass'),
   ('Sneha Kulkarni', 'sneha@example.com', 'kul1234');
   ```

2. Order :
   ```
   INSERT INTO Orders (user_id, order_date, total_cost, status) VALUES
   (1, '2025-04-01 10:00:00', 499.00, 'delivered'),
   (2, '2025-04-02 10:30:00', 299.00, 'delivered'),
   (3, '2025-04-03 11:00:00', 599.00, 'shipped'),
   (4, '2025-04-04 12:00:00', 199.00, 'shipped'),
   (5, '2025-04-05 14:00:00', 249.00, 'pending'),
   (6, '2025-04-06 15:00:00', 329.00, 'delivered'),
   (7, '2025-04-07 16:00:00', 349.00, 'pending'),
   (8, '2025-04-08 17:00:00', 199.00, 'pending'),
   (9, '2025-04-09 18:00:00', 449.00, 'pending'),
   (10, '2025-04-10 19:00:00', 179.00, 'pending');
   ```

3. Order Items:
   ```
   INSERT INTO Order_Items (order_id, product_id, quantity, subtotal) VALUES
   (1, 1, 1, 499.00),
   (2, 2, 1, 299.00),
   ```

(3, 3, 1, 599.00),
(4, 4, 1, 199.00),
(5, 5, 1, 249.00),
(6, 6, 1, 329.00),
(7, 7, 1, 349.00),
(8, 8, 1, 199.00),
(9, 9, 1, 449.00),
(10, 10, 1, 179.00);

4. Contact Submissions:
   INSERT INTO Contact_Submission (name, email, message, submission_date) VALUES
   ('Aarav Mehta', 'aarav@example.com', 'Loved the cake quality!', '2025-04-01'),
   ('Priya Sharma', 'priya@example.com', 'Do you deliver in Delhi?', '2025-04-02'),
   ('Ravi Kapoor', 'ravi@example.com', 'Is there a gluten-free option?', '2025-04-03'),
   ('Anjali Singh', 'anjali@example.com', 'Amazing packaging and taste!', '2025-04-04'),
   ('Karan Das', 'karan@example.com', 'I'd like to customize a cake.', '2025-04-05'),
   ('Neha Joshi', 'neha@example.com', 'Thanks for fast delivery!', '2025-04-06'),
   ('Vikram Rao', 'vikram@example.com', 'Do you have eggless items?', '2025-04-07'),
   ('Meera Nair', 'meera@example.com', 'How long do cakes stay fresh?', '2025-04-08'),
   ('Rohit Verma', 'rohit@example.com', 'Do you take bulk orders?', '2025-04-09'),
   ('Sneha Kulkarni', 'sneha@example.com', 'Everything was perfect!', '2025-04-10');

5. Product :
   INSERT INTO Products (product_name, description, price, stock_quantity) VALUES
   ('Chocolate Cake', 'Rich chocolate layered cake with ganache', 499.00, 10),
   ('Vanilla Cupcakes', 'Pack of 6 vanilla cupcakes with buttercream', 299.00, 20),
   ('Red Velvet Cake', 'Cream cheese frosted red velvet cake', 599.00, 8),
   ('Banana Bread', 'Moist and healthy banana bread loaf', 199.00, 15),
   ('Brownies', 'Fudgy brownies with walnuts (Pack of 4)', 249.00, 25),
   ('Blueberry Muffins', 'Pack of 6 freshly baked blueberry muffins', 329.00, 18),
   ('Strawberry Tart', 'Tart with creamy custard and fresh strawberries', 349.00, 12),
   ('Cheesecake Slice', 'Classic baked New York cheesecake slice', 199.00, 30),
   ('Lemon Cake', 'Tangy lemon-flavored sponge cake', 449.00, 9),
   ('Oatmeal Cookies', 'Pack of 10 homemade oatmeal cookies', 179.00, 22);

6. Payment :
   INSERT INTO Payments (order_id, payment_method, payment_date, amount) VALUES

(1, 'credit card', '2025-04-01 10:30:00', 499.00),
(2, 'paypal', '2025-04-02 11:15:00', 299.00),
(3, 'bank transfer', '2025-04-03 14:45:00', 599.00),
(4, 'credit card', '2025-04-04 16:20:00', 199.00),
(5, 'paypal', '2025-04-05 12:00:00', 249.00),
(6, 'credit card', '2025-04-06 09:00:00', 329.00),
(7, 'bank transfer', '2025-04-06 17:30:00', 349.00),
(8, 'paypal', '2025-04-07 13:00:00', 199.00),
(9, 'credit card', '2025-04-08 15:45:00', 449.00),
(10, 'bank transfer', '2025-04-09 10:10:00', 179.00);

7. Delivery:
   INSERT INTO Deliveries (order_id, delivery_address, delivery_date, status) VALUES
   (1, '123 MG Road, Bangalore', '2025-04-03 17:00:00', 'delivered'),
   (2, '45 Park Street, Kolkata', '2025-04-04 18:30:00', 'delivered'),
   (3, '78 Carter Road, Mumbai', '2025-04-05 14:00:00', 'shipped'),
   (4, '12 Anna Nagar, Chennai', '2025-04-06 16:00:00', 'shipped'),
   (5, '88 Sector 15, Noida', '2025-04-06 19:00:00', 'pending'),
   (6, '50 Rajouri Garden, Delhi', '2025-04-07 11:00:00', 'delivered'),
   (7, '27 Laxmi Road, Pune', '2025-04-08 13:30:00', 'pending'),
   (8, '36 Banjara Hills, Hyderabad', '2025-04-09 15:00:00', 'pending'),
   (9, '91 Salt Lake, Kolkata', '2025-04-10 17:00:00', 'pending'),
   (10, '22 Civil Lines, Jaipur', '2025-04-10 19:30:00', 'pending');

Basic SQL Queries Additional Basic Operations (Filtering, Ordering, Limiting)

1. List all users and their orders (if any)

SELECT u.user_id, u.name, o.order_id, o.total_cost, o.status

FROM Users u

LEFT OUTER JOIN Orders o ON u.user_id = o.user_id;

2. View order and payment details

SELECT order_id, total_cost, payment_method, amount

FROM Orders

NATURAL JOIN Payments;

3. List products in each order with quantity and subtotal

```
SELECT order_id, product_name, quantity, subtotal
FROM Order_Items
NATURAL JOIN Products;
```

4. Show total quantity ordered per product

```
SELECT p.product_id, p.product_name, SUM(oi.quantity) AS total_ordered
FROM Products p
LEFT OUTER JOIN Order_Items oi ON p.product_id = oi.product_id
GROUP BY p.product_id, p.product_name;
```

5. Order status and delivery info

```
SELECT o.order_id, o.status AS order_status, d.delivery_date, d.status AS delivery_status
FROM Orders o
LEFT OUTER JOIN Deliveries d ON o.order_id = d.order_id;
```

6. Payments made by users

```
SELECT p.payment_id, p.amount, u.name
FROM Payments p
RIGHT OUTER JOIN Orders o ON p.order_id = o.order_id
RIGHT OUTER JOIN Users u ON o.user_id = u.user_id;
```

7. List users who never ordered anything

```
SELECT u.user_id, u.name
FROM Users u
LEFT OUTER JOIN Orders o ON u.user_id = o.user_id
WHERE o.order_id IS NULL;
```

8. Find unpaid orders

```
SELECT o.order_id, o.total_cost
FROM Orders o
LEFT OUTER JOIN Payments p ON o.order_id = p.order_id
WHERE p.payment_id IS NULL;
```

9. Match contact form submissions to registered users

```
SELECT cs.name, cs.email, cs.message, u.user_id
FROM Contact_Submission cs
LEFT OUTER JOIN Users u ON cs.email = u.email;
```

10. List out-of-stock products

```
SELECT product_id, product_name, stock_quantity
FROM Products
WHERE stock_quantity = 0;
```

11. Get names and emails from contact submissions

```
SELECT name, email FROM Contact_Submission;
```

12. Get all orders placed in April 2025

```
SELECT * FROM Orders
WHERE order_date BETWEEN '2025-04-01' AND '2025-04-30';
```

13. Get orders and corresponding payment details

```
SELECT o.order_id, o.total_cost, p.payment_method, p.amount
FROM Orders o
JOIN Payments p ON o.order_id = p.order_id;
```

14. Calculate total items sold per product

```
SELECT p.product_name, SUM(oi.quantity) AS total_sold
FROM Order_Items oi
JOIN Products p ON oi.product_id = p.product_id
```

GROUP BY p.product_name;

15. Orders with customer names

SELECT order_id, name

FROM Orders

NATURAL JOIN Users;

16. Get orders with more than 1 item

SELECT order_id

FROM Order_Items

GROUP BY order_id

HAVING COUNT(order_item_id) > 1;

17. Show the 5 most recent payments

SELECT * FROM Payments

ORDER BY payment_date DESC

LIMIT 5;

18. Find users who placed orders

SELECT DISTINCT u.user_id, u.name

FROM Users u

JOIN Orders o ON u.user_id = o.user_id;

19. Delete a product from catalog

DELETE FROM Products WHERE product_id = 9;

20. Payment and corresponding product names

SELECT payment_id, amount, product_name

FROM Payments

NATURAL JOIN Orders

NATURAL JOIN Order_Items

NATURAL JOIN Products;

VII. Project Demonstration

The development of Crave Bites required the usage of a multitude of varied tools and software to ensure that our project becomes what we had dreamt of, i.e. providing scalability, responsiveness and a user-friendly UI/UX design

Visual Studio Code served as the base of our project whereupon we built this entire project. VS Code offers a systematic and efficient platform for organizing all the various languages and tools we used, all in one place.

HTML was used to create the skeletal structure of our web application offering a reliable and friendly user coding experience

JavaScript was used to make our website come to life with it's capability for stylized animations and beautiful UI/UX designs.

MySQL workbench was used for backend development, with it's capabilities to work on, it proved extremely helpful on our conquest to make this project come true.

Finally node.js was used to connect the backend databases and the frontend web application.

VIII. Learning from the Project

Through the development of this E-commerce platform for home-run bakeries, I gained a thorough understanding of how to design and implement a structured, database-driven application tailored specifically for small-scale, single-person businesses. The project began with identifying the unique needs of such bakeries, which operate without any outsourcing, and designing a system that supports both the business owner and customer within a single platform. I learned how to create a well-organized, normalized database schema using DBMS principles, ensuring efficiency and eliminating redundancy by applying 1NF, 2NF, and 3NF. The schema included essential tables like Users, Products, Orders, Order_Items, Payments, Payment_Methods, and an optional Delivery table, all connected through well-defined primary and foreign keys. I wrote precise SQL CREATE TABLE scripts using MySQL,

establishing a strong backend structure capable of handling core functionalities such as user management, product listings, order processing, and payment tracking. Additionally, I explored how such a database could be connected to a Java-based backend, setting the groundwork for future integration, and gained insight into how data could flow between backend systems and a frontend website. The project also involved the creation and interpretation of a detailed Entity-Relationship Diagram, helping visualize the structure and relationships between different entities. Overall, this project deepened my understanding of database design, normalization, and backend planning for real-world applications.

IX. Challenges Faced

Some of the key challenges faced during the development of this E-commerce platform for home-run bakeries included carefully designing a database structure that caters to both the baker and the customer within a single unified user model. Since the platform is meant for single-person businesses, it was important to avoid unnecessary complexity like managing multiple bakers or employees, which challenged the typical E-commerce schema patterns. Another challenge was ensuring proper normalization of the database without breaking essential relationships—structuring the data in 3NF while maintaining usability and performance required careful planning, especially when handling multiple items in a single order and linking payments to orders and methods. Creating a clear and professional Entity-Relationship Diagram that accurately reflected all entities, attributes, and cardinalities was also tricky, especially when using tools that sometimes misrendered text or connections, requiring regeneration and correction. Overall, balancing simplicity with functionality while maintaining clean design principles was a recurring challenge throughout the project.

X. Conclusion

This project allowed us to explore the complete lifecycle of building a small-scale E-commerce platform tailored specifically for home-run bakers, focusing on both functionality and proper data management. Through the course of development, we learned how to design and structure a relational database using DBMS principles, starting from an unnormalized form and methodically applying normalization to eliminate redundancy, improve data integrity, and create scalable structures. We created seven essential tables—Users, Products, Orders, Order Items, Payments, Deliveries, and Contact Submissions—ensuring that each one was logically connected through primary and foreign key relationships. In the process, we understood the importance of minimizing data repetition, how to break down composite data into atomic fields, and how normalized schemas directly support a clean, efficient backend for any web application. Though we didn’t implement RESTful APIs or full Java-based integration in this project, we still gained significant insights into backend planning, SQL scripting, and how frontend platforms can be connected to

databases for dynamic and user-driven experiences. Overall, this project served as a strong foundation in both database design and E-commerce logic for single-person business operations.