BIKE STORE MANAGEMENT SYSTEM

OVERVIEW: The Bike Store Management System , a Web-based application that serves as a pivotal tool for the staff, accessible only through secure login credentials. The team gains access to a comprehensive suite of features designed to streamline our day-to-day activities.

- Within the system, the staff can efficiently monitor and manage customer orders, allowing for a seamless tracking process. Additionally, the platform provides insights into our customer database, enabling us to better understand and cater to their needs. Furthermore, the system offers real-time visibility into our current stock of items. This system ensures smooth and precise management of the bike store operations.

PRINCIPAL FEATURES:

- Staff Management: The system comprehensively manages staff data, encompassing essential information including names, email addresses, and phone numbers.
- Bike Management: The system diligently monitors motor-bike details, maintaining a seamlessly synchronized record that includes customer orders and the current stock of these items in the stores.

DATABASE DESIGN:

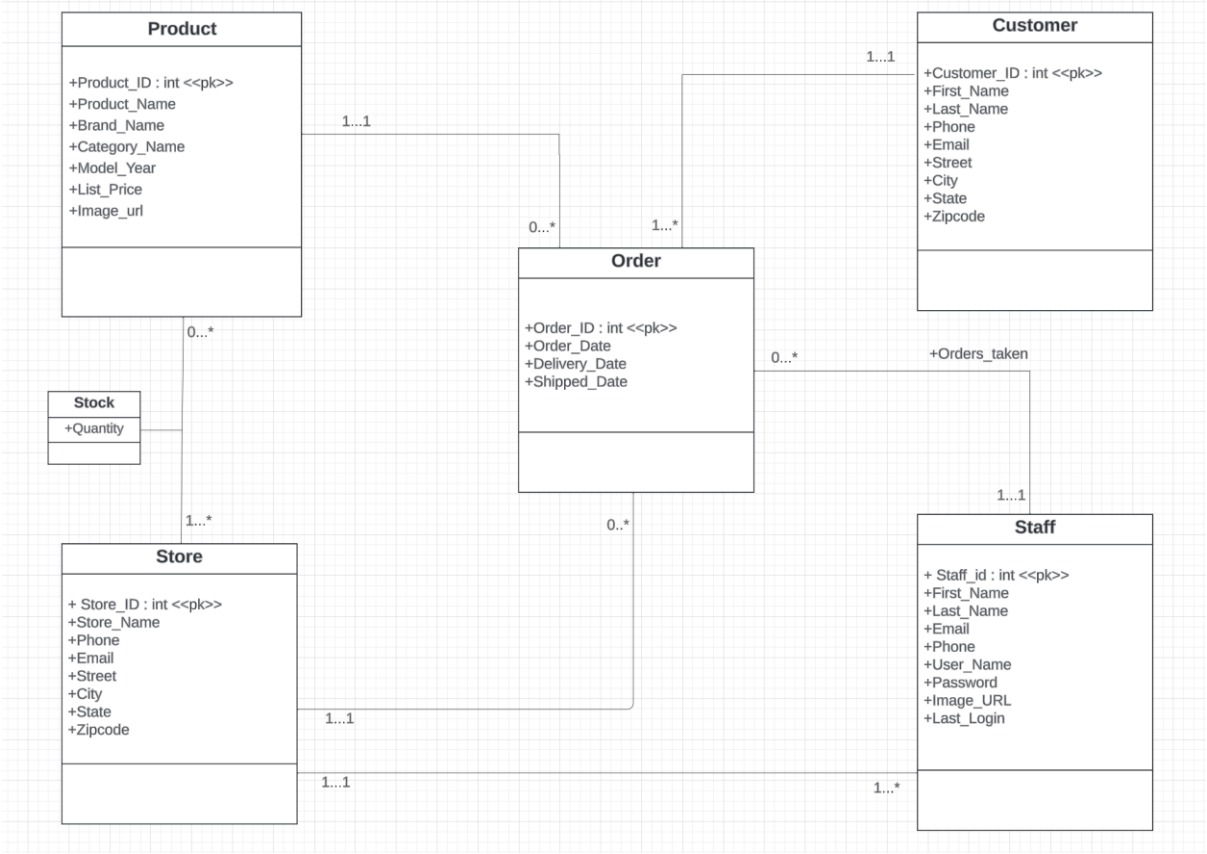
- *Classes*: The database incorporates distinct classes for staff, orders, customers, products and stores.
- *Attributes:* Within each class, specific attributes are defined. For instance, the staff class includes attributes such as name, email, phone etc, while the orders class encompasses details like customer_ id, product_id, delivery date etc.

Relationships: The database intricately maps relationships between classes, capturing vital connections. For instance, it manages associations between staff and the orders they have taken etc.

SOFTWARE COMPONENTS:

- User Interface (UI): The UI is developed using Django's template engine, which allows for the creation of dynamic web pages. Django templates enable the seamless integration of Python code within HTML, facilitating the presentation of data retrieved from the server side.The UI is designed to be responsive, ensuring a consistent and user-friendly experience across various devices and screen sizes. This enhances accessibility and usability for both customers and staff interacting with the application.
- Database (DB):The MySQL database is structured to efficiently manage and store crucial data related to the Bike Store. The schema includes tables for managing stocks, customer details, order information, branch details, and staff records. The database design adheres to normalization principles to eliminate data redundancy and maintain data integrity. Indexing is implemented strategically to optimize query performance, ensuring quick and efficient retrieval of information.
- Business Logic:The core business logic is implemented using Python through the Django framework. Python's readability and modular design principles are leveraged to create a maintainable and extensible codebase.
- CRUD Operations: The application supports essential CRUD (Create, Read, Update, Delete) operations for managing customers, products, staff, stores, and stocks. These operations are seamlessly integrated into the Django framework, ensuring data consistency and reliability.
- Security Measures: Security is a priority in the business logic implementation. Django's built-in security features, such as protection against common web vulnerabilities, contribute to a robust and secure application.

UML Design:



This UML (Unified Modeling Language) diagram represents the data model of a bike store management system. The UML includes five entities: Product, Stock, Store, Order, Customer, and Staff. Each entity has a set of attributes, and the relationships between these entities are represented by lines with cardinality indicators.

Here are the details of each entity and its relationships:

- Product:
  - Attributes: Product_ID, Product_Name, Brand_Name, Category_Name, Model_Year, List_Price, Image_url.
  - Has a one-to-many relationship with Stock, indicating each product can have multiple stock entries.
- Stock:
  - Attribute: Quantity.
  - Has a many-to-one relationship with Store, suggesting that stock is held in specific stores.
- Store:
  - Attributes: Store_ID, Store_Name, Phone, Email, Street, City, State, Zipcode.
  - Has a one-to-one relationship with Staff, meaning each store has one staff member responsible for it.
- Order:
  - Attributes: Order_ID, Order_Date, Delivery_Date, Shipped_Date.
  - Has a one-to-many relationship with Customer, indicating a customer can place multiple orders.
  - Has a one-to-many relationship with Product, suggesting an order can contain multiple products.
- Customer:
  - Attributes: Customer_ID, First_Name, Last_Name, Phone, Email, Street, City, State, Zipcode.
  - Has a one-to-many relationship with Staff, shown by the Orders_taken attribute, which implies a staff member can take orders from many customers.

- Staff:
  - Attributes: Staff_id, First_Name, Last_Name, Email, Phone, User_Name, Password, Image_URL, Last_Login.
  - Is linked to Store and Customer through a one-to-many relationship, indicating their role in managing stores and taking orders from customers.

The relationships between the entities are essential for understanding how data is interconnected in the system. For example, the ERD shows that customers can have multiple orders, orders can include multiple products, and stores manage their stock and are assigned to specific staff members. This type of diagram is crucial for designing and understanding database schemas.

BCNF: BCNF is designed to eliminate certain types of redundancy in a relational database by addressing functional dependencies. Following the steps, BCNF resulted in proper tables. Thus it favorable design for this system. Performs well in avoiding anomalies and Maintains losslessness (preservation of information) in the normalization process.

3NF: 3NF aims to reduce redundancy by eliminating transitive dependencies in addition to addressing those handled by 2NF. Like BCNF, 3NF helps in avoiding anomalies related to insertion, update, and deletion.3NF, when applied preserved lossless decomposition for this database.3NF ensures dependency preservation up to a certain level, but it may not preserve all dependencies compared to BCNF.

COMPARISION OF SCHEMAS:

	BCNF	3NF	UML
Reduction in redundancy	Same	Same	Same
Avoidance of anomalies	Good	Good	Good
Losslessness	Yes	Yes	Yes
Dependency Preservation	Yes	Yes	Yes
Overall quality	Best	Best	Best

CONCLUSION:

If it is to design a relational database with a focus on normalization, BCNF and 3NF are relevant concepts. If in the early stages of system design and need visual representation, UML is a suitable choice, but it doesn't replace normalization in the context of relational databases. In this case, BCNF, 3NF, and UML all exhibit the same levels of reduction in redundancy, avoidance of anomalies, losslessness, and dependency preservation. BCNF, 3NF, and UML are all considered to be of the "best" overall quality according to the given criteria. There is no best design as all three give similar results.