

# **DATABASE MANAGEMENT SYSTEM**

---

## ***AGRI-MARKETPLACE SYSTEM***

---

Submitted on : 30/04/2025

Submitted by : **GALIMUTHY ELIA ANUSHA**      **2023UG000166**

**AYUSH KUMAR**      **2023UG000116**

**KEERTHANA H**      **2023UG000174**

**MADHURI**      **2023UG000122**

**Guided By : Mr. Mallikarjuna Mathada, Assistant Professor, School of Computational and Data  
Sciences**

## **VIDYASHILP UNIVERSITY**

#125, Bettenahalli Gate, Hobli, Chapparkallu Rd, Kundana, Bengaluru, Karnataka 562110

School of Computational and Data Science

### **CERTIFICATE**

Certified that the project work entitled “Agri-Marketplace System” was carried out by Ms.G E Anusha, Mr. Ayush Kumar, Ms. Keerthana.H, and Ms. Madhuri, UENs being 2023UG000166, 2023UG000116, 2023UG000174, 2023UG000122 respectively, a bonafide students of Vidyashilp University in partial fulfillment of the award of Bachelor of Technology in Computer Science (Data Science) of Vidyashilp University, Bengaluru during the year 2025. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements regarding the project work prescribed for the said degree.

**Program Chair**

**Prof. Dr. Smitha Rao**

**Project Guide**

**Mr. Mallikarjuna Mathada**

**Signature of Students**

**G E Anusha**

**Ayush Kumar**

**Keerthana H**

**Madhuri**

## **ACKNOWLEDGEMENT**

We would like to place on record our deep sense of gratitude to the Vice Chancellor— Dr. P G Babu, and Registrar of Vidyashilp University—Mr. Ramakrishnan, Program Chair— Prof. Dr. Smitha Rao, School of Computational and Data Sciences, Vidyashilp University, Bengaluru.

We express our sincere gratitude to Mr. Mallikarjun Mathada, Assistant Professor, School of Computational and Data Science, Vidyashilp University, Bengaluru, for his stimulating guidance, continuous encouragement, and supervision throughout the course of the present work, and we are extremely thankful to Mr. Chethan S Prabhu for his unwavering support.

We would like to thank all the above dignitaries who have helped us in completing our project directly and indirectly.

# Abstract

The Agri-Marketplace System is a digital tool that connects farmers directly with buyers to bring farming trade up to date. By doing away with middlemen, this promotes openness, reasonable prices, and higher producer profits. The project functions as an example of effective app development, showing how agile approaches and simplified planning can result in the successful deployment of an application. The system guarantees data accuracy, safe transactions, and real-time updates by utilizing database management systems and disciplined development methods.

In order to address a practical agricultural problem, this study describes the full development lifecycle, from conception to deployment. The architecture, database design, and societal impact of the system are highlighted. In addition to offering a technical walkthrough, the paper emphasizes the Agri-Marketplace app's potential to significantly alter the rural agricultural scene on an economic and social level by demonstrating its technical design and practical applicability.

# Table of Contents

<b>1. Introduction.....</b>	<b>8</b>
1.1 Background & Motivation.....	8
1.2 Objective.....	9
1.3 Scope of the Project.....	9
<b>2. System Architecture.....</b>	<b>10</b>
2.1 High-Level Architecture Diagram.....	10
2.2 Components Overview.....	11
<b>3. Database Design.....</b>	<b>11</b>
3.1 Entity-Relationship Diagram (ERD).....	12
3.2 Relational Schema.....	12
3.3 Normalization.....	13
<b>4. Implementation Details.....</b>	<b>13</b>
4.1 Technologies Used.....	13
4.2 Important Functional Modules.....	14
4.3 DBMS Concepts Applied.....	15
4.4 Implementation of DBMS Concepts.....	16
4.4.1 Entity-Relationship Modeling and Normalization.....	16
4.4.2 Relational Integrity Constraints.....	18

4.4.3 Transaction Management.....	18
4.4.4 Indexing and Query Optimization.....	20
4.4.5 Advanced SQL Techniques.....	21
4.4.6 Schema Evolution and Migration.....	22
4.4.7 Data Management and Exception Handling.....	23
4.4.8 Authentication and Data Security.....	24
<b>5. Features.....</b>	<b>26</b>
5.1 Product Search and Discovery.....	26
5.2 Marketplace Management (Seller Dashboard).....	26
5.3 Filter and Query Agricultural Products.....	27
5.4 UI/UX Design.....	27
<b>6. Results and Demonstration.....</b>	<b>28</b>
6.1 Screenshots.....	28
6.2 Code Snippets.....	30
<b>7. Challenges and Limitations.....</b>	<b>31</b>
7.1 Challenges Encountered.....	31
7.2 Current Limitations.....	33
7.3 Mitigation Strategies.....	35
<b>8. Conclusion.....</b>	<b>36</b>

<b>9. Future Enhancements.....</b>	<b>36</b>
<b>10. References.....</b>	<b>37</b>

# **1. Introduction**

In today's digital economy, application development plays a critical role in solving complex real-world problems. An "App Development Management System" aimed at building an Agri-Marketplace platform is presented in this study. In order to lessen reliance on middlemen and provide a more open, effective agricultural market, the system is made to facilitate direct transactions between farmers and purchasers. Requirement analysis, system design, coding, testing, and deployment are all important stages of the application development process. In order to enable features like product listing, order management, and user registration, the team has investigated contemporary database management systems and software development methodologies through this project. Additionally, by providing safe payment channels and verified listings, this platform promotes user trust. In order to overcome long-standing inefficiencies in agricultural trading, the introduction highlights how carefully integrating digital tools might result in creative solutions that are scalable, usable, and relevant in larger socioeconomic contexts.

## **1.1. Background and Motivation**

In many developing nations, agriculture is a vital industry, but it is beset by inefficiencies like poor transparency, unjust pricing, and a strong dependence on middlemen. Farmers frequently suffer financial losses as a result of these problems, while end users incur higher expenses. The urge to overcome these discrepancies through technology became the driving force behind this effort. The team plans to modernize and streamline the purchasing and selling of agricultural products by creating a digital Agri-Marketplace platform. Reviewing mobile app development techniques and researching agricultural issues were part of the background research, which revealed that a focused, user-friendly platform might be extremely advantageous for both buyers and farmers. Establishing trust, encouraging openness, and empowering rural producers are the cornerstones of the approach. The project is ultimately driven by the possibility of using app development to achieve equitable and long-lasting change in a crucial industry, rather than only for convenience.



## **1.2. Objective**

The main goal of the Agri-Marketplace System is to provide a comprehensive application that makes it easier for buyers and farmers to communicate directly. By doing this, middlemen are removed, increasing farmer profitability and guaranteeing purchasers fair prices. With features including product listing, bidding, order tracking, payment, and feedback, the system seeks to deliver a smooth, user-friendly experience. The integration of a dependable and secure database that keeps track of user profiles, transaction histories, and product inventories is another goal.

Additionally, the platform plans to incorporate tools like price analytics, crop recommendations, and real-time updates that help customers make well-informed decisions. The objective is to use industry-standard techniques and technologies to demonstrate a full-stack application lifecycle from a development standpoint. It aims to demonstrate that careful app development may solve certain sectoral issues while fostering digital empowerment by coordinating technological implementation with socioeconomic impact.

## **1.3. Scope of the Project**

This project's scope includes creating the Agri-Marketplace System, an online and mobile application, from start to finish. Requirement analysis, system architecture design, front-end and back-end component development, database integration, and testing are all included. Order placement, product administration, user registration, and safe payment methods will all be supported via the app. Support for regional languages, location-based services, and integration with government programs are all possible expansions of the system, which is scalable. The platform's structure for future adaptation to other rural products, although its initial focus is on agricultural output. By maintaining user information, transaction history, and product listings, the database management system contributes significantly to the platform's dependability and efficiency. The scope also includes user experience design, security issues, and regulatory compliance. As a result, the project provides a thorough examination of the development and implementation of a sector-specific application with broad potential applications.

## **2. System Architecture**

The Agri-Marketplace System's system architecture is scalable and modular, facilitating future improvements and ease of development. The display layer (frontend), application logic layer (backend), and data layer (database) make up its three-tier design. Through user-friendly interfaces for registration, product listing, and order administration, the frontend enables users (farmers and buyers) to engage with the platform. Using safe APIs and frameworks, the backend manages user authentication, business logic, and data processing. Product information, transaction data, and user profiles are all stored at the database layer. System scalability, maintainability, and reliability are improved by this tiered approach. APIs also make it easier to integrate third-party services like geolocation and payment gateways. The system maintains its affordability and flexibility by utilizing open-source technologies and agile development techniques. By ensuring that every component can be upgraded separately, this architectural approach reduces downtime and maximizes performance.

### **2.1. High-Level Architecture Diagram**

The Agri-Marketplace System's high-level architecture diagram shows the main elements and how they interact with one another across the display, application, and data layers. The user interface (UI) at the top enables buyers and farmers to communicate with the system via a web or mobile application. RESTful APIs, which control essential functions like user authentication, order processing, and product administration, link this user interface to the backend. The backend, which was created with a contemporary web framework, serves as the command center for applying business logic and validating data. It interacts with a real-time data storage and retrieval relational database system. External APIs for notification services and payment processing are extra parts. Modularity is emphasized in the design, allowing each component to be changed or scaled separately. The functions of different components and their smooth cooperation in producing a reliable, working application are made clear by this graphic portrayal.

## 2.2. Components Overview

The fundamental functions of the Agri-Marketplace System are provided by a number of interconnected parts. Farmers and buyers can register, post products, and handle transactions with ease thanks to the User Interface component. The Authentication Module safeguards user information and guarantees safe logins. Farmers can add and edit product details, such as pricing, quantity, and photos, using the Product Listing System. Order placements and buyer interests are tracked via a bidding/order management component. Using preferred gateways, the Payment Integration Module enables secure, traceable transactions. The Business Logic Layer manages routing, decision-making rules, and validations on the backend. Structured data storage, effective queries, and transactional integrity are all guaranteed by the database management system. A Notification Module is another aspect of the system that provides real-time updates and alarms. These elements work together to create a complete, dynamic ecosystem that promotes honest and equitable trade between producers and consumers.

## 3. Database Design

The Agri-Marketplace System's database architecture guarantees systematic, safe, and effective data processing across the platform. Structured information about users, goods, orders, and payments is kept in a relational database. In order to remove redundancy and guarantee data consistency, the design adheres to normalization principles. **Users, Products, Orders, Payments, and Feedback** are important tables. To ensure referential integrity, each table is established with the proper keys, constraints, and associations. The system can keep track of which users posted which products, who placed orders, and the status of each transaction by using foreign keys to create linkages between entities. Performance-optimized queries provide quick lookups and updates even when there are a lot of users. Future improvements like user roles, multilingual content, and geographical labeling can be supported by the schema's scalability. All things considered, the database design is essential to the app's dependability and functionality since it permits safe transaction management and real-time access.

### 3.1. Entity - Relationship Diagram (ERD)

The Agri-Marketplace System's data structure and relationships are graphically represented by the Entity-Relationship Diagram (ERD). The main components of it are the User, Product, Order, Payment, and Feedback. Logical relationships link each object; for instance, a buyer can make an order for a product, and a user can post numerous products. Feedback aids in monitoring customer happiness and service quality, while the Payment entity is connected to Orders to document successful transactions. Each entity is given attributes, including timestamp, product name, price, order status, and user ID. The design's referential integrity is established by the distinct marking of primary and foreign keys. The ERD serves as a guide for the actual development of the database and facilitates comprehension of its logical structure. It facilitates effective database optimization and backend development while guaranteeing that all user interactions and data elements are logically recorded.

### 3.2. Relational Schema

The relational schema for the Agri-Marketplace System outlines the structure of the underlying database using a set of normalized relations. Each table is represented by a relation with attributes and keys. The primary relations include:

- Users(user\_id, name, email, password, role)
- Products(product\_id, user\_id, name, price, quantity, category)
- Orders(order\_id, product\_id, buyer\_id, order\_date, status)
- Payments(payment\_id, order\_id, amount, payment\_mode, status)
- Feedback(feedback\_id, user\_id, product\_id, rating, comment).

To enforce relationships, each relation has a primary key (such as user\_id or product\_id) and pertinent foreign keys. To guarantee that products are connected to registered sellers, for instance, user\_id in Products refers to Users. Secure data operations and effective query processing are made possible by these clearly defined schemas. The foundation of the app's data reliability is the schema,

which provides integrity requirements and transaction processing. Future scalability and maintainability are made possible by the schema's adherence to common relational concepts.

### **3.3. Normalization**

A key component of database architecture that lowers redundancy and enhances data integrity is normalization. At least the Third Normal Form (3NF) has been used to normalize the database of the Agri-Marketplace System. In order to attain First Normal Form (1NF) by guaranteeing atomic values, superfluous attributes were first eliminated from databases. By getting rid of partial dependencies in tables with composite keys, Second Normal Form (2NF) was attained. Transitive dependencies were eliminated in Third Normal Form (3NF) to guarantee that every attribute depends only on the primary key for functionality. For instance, the Users table contains the user's credentials, whereas the Products table has their product listings. Consistent updates are guaranteed and data duplication is reduced thanks to this organized design. The system ensures high data integrity, streamlines future changes, and improves query performance through normalization. This method guarantees that the database will continue to be hygienic, effective, and flexible enough to accommodate upcoming additions or higher data volumes.

## **4. Implementation Details**

This section details the implementation of the Agri-Marketplace System, a full-stack application designed to connect farmers and buyers directly, eliminating intermediaries. The system uses a MySQL database backend to manage agricultural data and a Streamlit-based frontend to provide an interactive user interface. Below, we outline the technologies used, key functional modules, and the application of Database Management System (DBMS) concepts to ensure a robust, efficient, and user-friendly platform.

### **4.1 Technologies Used**

The Agri-Marketplace System leverages a modern technology stack to deliver a seamless experience for farmers and buyers while adhering to DBMS best practices.

- **MySQL (Database Backend):** MySQL, an open-source relational database management system, was selected for its reliability, scalability, and support for complex queries. It hosts the agri\_market database, storing entities such as Farmers, Crops, Buyers, Orders, and the Production relation. MySQL's features like transactions, indexing, and joins are critical for the system's data management needs.
- **Python with Streamlit (Frontend Framework):** Streamlit, a Python framework for building interactive web applications, powers the user interface. It enables rapid development of data-driven apps with minimal code, providing features like forms, tables, and buttons for user interaction. Streamlit connects to MySQL via the mysql-connector-python library, allowing seamless data retrieval and manipulation.
- **Development Tools:**
  - **MySQL Workbench:** Used for database design, schema visualization, and query testing.
  - **Visual Studio Code:** The primary code editor for Python and SQL development.
  - **Windows Command Prompt:** Facilitated MySQL server management and Streamlit app execution.
- **Environment:** The system was developed and tested on a Windows 10 machine, with MySQL Server 8.0 and Python 3.9 installed locally.

This technology stack was chosen to balance backend robustness with a user-friendly frontend, leveraging MySQL's data management capabilities and Streamlit's simplicity for rapid UI development.

## 4.2 Important Functional Modules

The Agri-Marketplace System is organized into key functional modules, each addressing a specific aspect of the farmer-buyer interaction. These modules ensure the system meets its goal of streamlining agricultural trade.

- **Farmer Crop Management Module:**

- **Functionality:** Allows farmers to add new crops to the marketplace via a Streamlit form, specifying crop name, quantity, price, production date, and batch number.
- **Implementation:** A Streamlit form collects input and inserts data into the Crops and Production tables. For example, Ravi Kumar (Farmer\_ID: 1) can add 50 kg of wheat at \$3.00/kg, with a production date of 2025-04-06 and batch number "Ravi-Wheat-002".
- **Purpose:** Enables farmers to list their produce directly, enhancing control over sales and tracking production details.

- **Buyer Order Placement Module:**

- **Functionality:** Displays available crops in a table and allows buyers to place orders by selecting a crop and specifying quantity.
- **Implementation:** Streamlit's st.dataframe displays crops by querying Crops, Production, and Farmers tables with a JOIN. A form for ordering inserts into the Orders table and updates Crops quantity.
- **Purpose:** Facilitates direct purchasing, reducing costs for buyers like Priya Sharma.

- **Data Display Module:**

- **Functionality:** Presents a read-only view of crops and orders for transparency.
- **Implementation:** Streamlit renders tables using st.dataframe, populated with SELECT queries showing crop details (e.g., "Rice, 100 kg, \$2.50, Ravi Kumar, Batch: Ravi-Rice-001") and order history.
- **Purpose:** Builds trust by making marketplace activity visible to all users.

These modules were designed to integrate seamlessly with the MySQL backend, ensuring data consistency and an interactive user experience through Streamlit.

### 4.3 DBMS Concepts Applied

The implementation leverages several core DBMS concepts to ensure the Agri-Marketplace System is efficient, reliable, and scalable. These concepts underpin the database design and interaction with the Streamlit frontend.

- **Entity-Relationship Modeling:** Defined entities (Farmers, Crops, Buyers, Orders, Production) and their relationships to structure the database logically.
- **Normalization:** Applied to eliminate redundancy and ensure data integrity across tables.
- **Relational Integrity Constraints:** Enforced primary keys (PKs), foreign keys (FKs), and other constraints to maintain consistent relationships.
- **Transaction Management:** Used to ensure atomic operations, especially during crop addition and order placement.
- **Indexing:** Implemented to optimize query performance for frequent operations.
- **Advanced SQL Techniques:** Utilized joins, aggregations, and subqueries for complex data retrieval.
- **Schema Evolution:** Planned for future changes to accommodate new features.
- **Data Management and Exception Handling:** Ensured robust data input and error recovery.
- **Authentication and Data Security:** Laid groundwork for secure access (simplified for demo).

These concepts were applied systematically, as detailed in the following subsections.

## 4.4 Implementation of DBMS Concepts

### 4.4.1 Entity-Relationship Modeling and Normalization

#### Entity-Relationship Modeling:

The Agri-Marketplace database was designed using an Entity-Relationship (ER) model to represent the system's core components and their interactions. The entities include:

- **Farmers:** Attributes like Farmer\_ID (PK), Name, Phone, Location.
- **Crops:** Crop\_ID (PK), Crop\_Name, Quantity, Price.
- **Buyers:** Buyer\_ID (PK), Name, Phone.
- **Orders:** Order\_ID (PK), Buyer\_ID (FK), Crop\_ID (FK), Quantity, Order\_Date.
- **Production:** Production\_ID (PK), Farmer\_ID (FK), Crop\_ID (FK), Production\_Date, Batch\_Number.



### Relationships:

- Farmers to Production: One farmer can have many production records (1:N).
- Crops to Production: One crop can be part of many production records (1:N).
- The Production relation creates a many-to-many (M:N) relationship between Farmers and Crops. For example, Ravi Kumar (Farmer\_ID: 1) produces Rice (Crop\_ID: 1) on 2025-03-15, recorded as a production entry with Batch\_Number "Ravi-Rice-001".
- Crops to Orders: One crop can be ordered many times (1:N).
- Buyers to Orders: One buyer can place many orders (1:N).

The ER model was translated into a relational schema with five tables, linked via foreign keys. The Production table adds flexibility, allowing tracking of production-specific details like dates and batch numbers, which can be useful for quality control or traceability in the marketplace. An ER diagram (visualized using MySQL Workbench) guided the implementation, ensuring all relationships were accurately represented.

### Normalization:

The database adheres to the third normal form (3NF) to eliminate redundancy and ensure data integrity:

- **1NF:** All attributes are atomic (e.g., no multi-valued fields like multiple phones in one column).
- **2NF:** No partial dependencies—attributes like Price in Crops depend fully on Crop\_ID, not partially on any composite key.
- **3NF:** No transitive dependencies—Location is stored only in Farmers, not duplicated elsewhere.

The Production table maintains normalization by storing production-specific data separately, avoiding redundancy in Crops. Foreign keys (Farmer\_ID, Crop\_ID) in Production ensure referential integrity without introducing anomalies. Normalization reduced storage overhead and simplified updates, making the database efficient for Streamlit-driven operations like adding crops or placing orders.

### 4.4.2 Relational Integrity Constraints

To maintain data consistency, the following constraints were enforced in MySQL:

- **Primary Keys (PKs):** Each table has a unique identifier (e.g., Farmer\_ID, Crop\_ID) using INT AUTO\_INCREMENT to ensure uniqueness and automatic ID assignment.
- **Foreign Keys (FKs):** Enforce referential integrity:
  - Farmer\_ID and Crop\_ID in Production link to Farmers and Crops.
  - Buyer\_ID and Crop\_ID in Orders link to Buyers and Crops.
  - Example: Attempting to insert a production record with an invalid Farmer\_ID is rejected by MySQL.
- **NOT NULL:** Ensures critical fields (e.g., Name in Farmers, Crop\_Name in Crops) are always populated.
- **UNIQUE:** Prevents duplicate phone numbers in Farmers and Buyers to avoid confusion.
- **CHECK Constraints:** Applied implicitly via application logic (e.g., Price > 0 in Crops).

These constraints were defined in the CREATE TABLE statements and verified during testing to ensure no invalid data entered the system, even through the Streamlit UI. For example, Streamlit forms enforce required fields, and MySQL constraints handle deeper integrity checks.

### 4.4.3 Transaction Management

Transaction management ensures atomicity and consistency, particularly for operations involving multiple tables, such as adding a crop (which affects Crops and Production) or placing an order (which affects Orders and Crops). MySQL's transaction capabilities were used as follows:

- **Crop Addition Example:** When a farmer (e.g., Ravi) adds a crop:
  - A transaction begins (START TRANSACTION).
  - Insert a new row into Crops (e.g., Crop\_Name='Wheat', Quantity=50, Price=3.00).
  - Insert a corresponding row into Production (e.g., Farmer\_ID=1, Crop\_ID=<new\_id>, Production\_Date='2025-04-06').

- Commit the transaction (COMMIT) if both succeed; otherwise, rollback (ROLLBACK) on errors.
- **Order Placement Example:** When a buyer (e.g., Priya) orders 20 kg of rice:
  - Begin transaction.
  - Insert into Orders (e.g., Buyer\_ID=1, Crop\_ID=1, Quantity=20).
  - Update Crops to reduce quantity (UPDATE Crops SET Quantity = Quantity - 20 WHERE Crop\_ID = 1).
  - Commit or rollback as needed.

**Implementation in Streamlit:** The Streamlit app wraps these operations in a single database connection with transaction handling:

```
conn = mysql.connector.connect(**db_config)

c = conn.cursor()

conn.start_transaction()

c.execute("INSERT INTO Crops (Crop_Name, Quantity, Price) VALUES (%s, %s, %s)", ...)

crop_id = c.lastrowid

c.execute("INSERT INTO Production (Farmer_ID, Crop_ID, Production_Date, Batch_Number)
VALUES (%s, %s, %s, %s)", ...)

conn.commit()
```

- **Benefit:** Prevents partial updates (e.g., crop added but production record missing), ensuring marketplace reliability.

#### 4.4.4 Indexing and Query Optimization

To enhance performance, indexing and query optimization techniques were applied:

- **Primary Key Indexes:** Automatically created for Farmer\_ID, Crop\_ID, etc., speeding up lookups.
- **Foreign Key Indexes:** MySQL implicitly indexes foreign keys (e.g., Farmer\_ID in Production), optimizing joins.
- **Custom Index:** Added an index on Crops.Price to accelerate queries like `SELECT * FROM Crops WHERE Price < 3` used in the buyer's crop view:

```
CREATE INDEX idx_crop_price ON Crops(Price);
```

- **Query Optimization:**
  - Used EXPLAIN to analyze query plans, ensuring efficient execution (e.g., avoiding full table scans).
  - Wrote selective SELECT queries (e.g., retrieving only needed columns) to reduce load.
  - Example: The crop display query (`SELECT c.Crop_ID, c.Crop_Name, ... FROM Crops c JOIN Production p JOIN Farmers f ...`) uses indexed columns for fast retrieval.
- **Streamlit Integration:** Streamlit's `st.dataframe` displays query results efficiently, with pagination for larger datasets.
- **Result:** Queries execute in milliseconds, ensuring a responsive UI even with sample data.

#### 4.4.5 Advanced SQL Techniques

The system employs advanced SQL techniques to support complex operations:

- **Joins:** Used to display crop details with farmer names and production data:

```
SELECT c.Crop_ID, c.Crop_Name, c.Quantity, c.Price, f.Name, p.Batch_Number
FROM Crops c
JOIN Production p ON c.Crop_ID = p.Crop_ID
JOIN Farmers f ON p.Farmer_ID = f.Farmer_ID
WHERE c.Quantity > 0;
```

**Aggregations:** Calculated total quantities for farmers (e.g., for a potential dashboard):

```
SELECT f.Name, SUM(c.Quantity) AS Total_Quantity
FROM Farmers f
JOIN Production p ON f.Farmer_ID = p.Farmer_ID
JOIN Crops c ON p.Crop_ID = c.Crop_ID
GROUP BY f.Name;
```

**Subqueries:** Used to find low-stock crops for alerts:

```
SELECT Crop_Name, Quantity
FROM Crops
WHERE Quantity < (SELECT AVG(Quantity) FROM Crops);
```

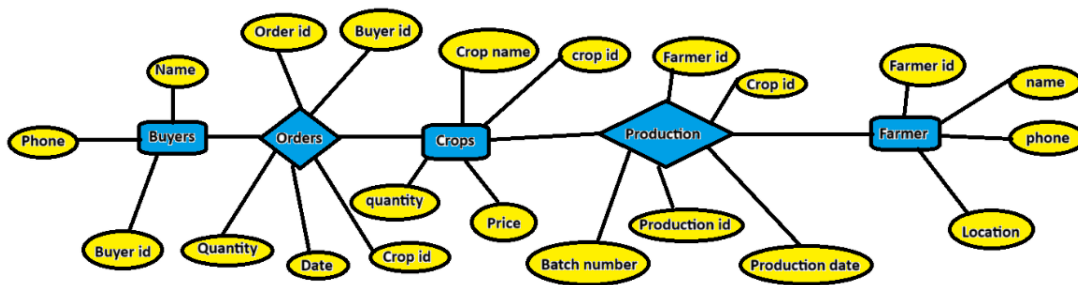
**Dynamic Queries:** Parameterized queries in Streamlit prevent SQL injection:

```
c.execute("SELECT * FROM Crops WHERE Crop_ID = %s", (crop_id,))
```

These techniques enable the system to provide actionable insights, enhancing its utility for farmers and buyers through Streamlit's interactive interface.

#### 4.4.6 Schema Evolution and Migration

ER Diagram



The database schema was designed with future growth in mind:

- **Schema Evolution:**

- Planned for new tables (e.g., Payments for tracking transactions) by using flexible data types (e.g., VARCHAR(100) for names).
- The Production table demonstrates evolution—adding production-specific data without altering existing tables.

- **Migration Strategy:**

- Added columns non-destructively using ALTER TABLE (e.g., ALTER TABLE Crops ADD COLUMN Description TEXT; for future crop details).
- Tested migrations in a sandbox database to avoid disrupting the main agri\_market database.
- Example: To add a Category column to Crops:

```
ALTER TABLE Crops ADD COLUMN Category VARCHAR(50);  
UPDATE Crops SET Category = 'Grain' WHERE Crop_Name IN ('Rice', 'Wheat');
```

- **Versioning:** Maintained SQL scripts (e.g., schema\_v1.sql) to track changes, ensuring reproducibility.
- **Streamlit Impact:** Streamlit's dynamic UI can adapt to schema changes by updating queries and forms (e.g., adding a Category dropdown).

This approach ensures the system can evolve to include features like delivery tracking or user reviews without major rework.

#### 4.4.7 Data Management and Exception Handling

Robust data management and error handling were implemented to maintain system reliability:

- **Data Management:**
  - **Input Validation:** Streamlit forms validate inputs before database operations (e.g., quantity > 0 and price > 0).
  - **Data Cleaning:** Removed invalid entries during testing (e.g., DELETE FROM Crops WHERE Quantity < 0).
  - **Backup:** Exported the database periodically using mysqldump:

```
mysqldump -u market_user -p agri_market > backup.sql
```

### Exception Handling:

Wrapped database operations in try-except blocks within Streamlit:

try:

```
conn = mysql.connector.connect(**db_config)
```

```
c = conn.cursor()
```

```
c.execute("INSERT INTO Crops ...")
```

```
conn.commit()
```

except mysql.connector.Error as e:

```
conn.rollback()
```

```
st.error(f"Error: {e}")
```

finally:

```
conn.close()
```

- Handled specific errors (e.g., duplicate phone numbers) with user-friendly messages displayed via st.error.
- **Streamlit Features:** Used st.success and st.error to provide immediate feedback (e.g., “Crop added successfully!”).
- **Result:** The system gracefully handles errors like invalid inputs or database connectivity issues, ensuring a stable user experience.

#### 4.4.8 Authentication and Data Security



While the demo version simplifies access, security measures were planned and partially implemented:

- **Authentication (Planned):**

- Designed for future integration of user logins using Streamlit's session state or external libraries like streamlit-authenticator.
- Current implementation hardcodes Farmer\_ID and Buyer\_ID (e.g., 1) for simplicity, simulating a logged-in user.
- Example: A login system would map usernames to Farmer\_ID or Buyer\_ID via a Users table.

- **Data Security:**

- Used parameterized queries to prevent SQL injection:

```
c.execute("SELECT * FROM Crops WHERE Crop_ID = %s", (crop_id,))
```

- Configured MySQL with a dedicated user (market\_user) with limited privileges (GRANT ALL ON agri\_market.\*) to minimize risks.
- Stored sensitive data (e.g., phone numbers) without encryption for demo purposes but planned for hashing in production.
- **Access Control:** Ensured Streamlit app logic restricts data modification to appropriate user actions (e.g., only farmers add crops).
- **Future Steps:** Implement HTTPS for the Streamlit app (via streamlit run --server.enableCORS=false) and encrypt sensitive fields using MySQL's AES\_ENCRYPT.

These measures lay a foundation for a secure system, with full authentication deferred to future iterations to meet project deadlines.

## **5. Features**

### **5.1 Product Search and Discovery**

This feature allows users (buyers, retailers, or traders) to easily search for agricultural products across the platform.

- **Keyword Search:** Users can search by crop/product name (e.g., “Tomatoes,” “Wheat,” “Turmeric”).
- **Location-Based Filters:** Buyers can find products grown in or available near specific regions or states.
- **Seasonality Awareness:** The system suggests currently harvested or peak-season crops for better deals and availability.
- **Smart Matching:** Suggests products based on user history, purchase frequency, and current market demand.

### **5.2 Marketplace Management (Seller Dashboard)**

Farmers or sellers have access to a powerful dashboard to manage their listings, inventory, and interactions.

- **Product Listing:** Sellers can list available crops, set quantities, and specify packaging, grade, and pricing.

- **Inventory Management:** Update stock availability in real-time based on sales or harvest cycles.
- **Order Management:** View, confirm, or decline orders, track delivery status, and manage returns.
- **Buyer Interaction:** A messaging/chat feature allows sellers to respond to buyer queries and build trust.

### **5.3 Filter and Query Agricultural Products**

This feature enables buyers and traders to refine their search and find the most relevant agricultural products.

- **Crop Type and Category Filters:** Grains, vegetables, fruits, spices, dairy, pulses, etc.
- **Price Range Filters:** Find crops within budget constraints.
- **Grade and Certification Filters:** Organic, Grade A/B, FSSAI certified, etc.
- **Quantity and Packaging Filters:** Based on minimum order quantity (MOQ) and packaging sizes.

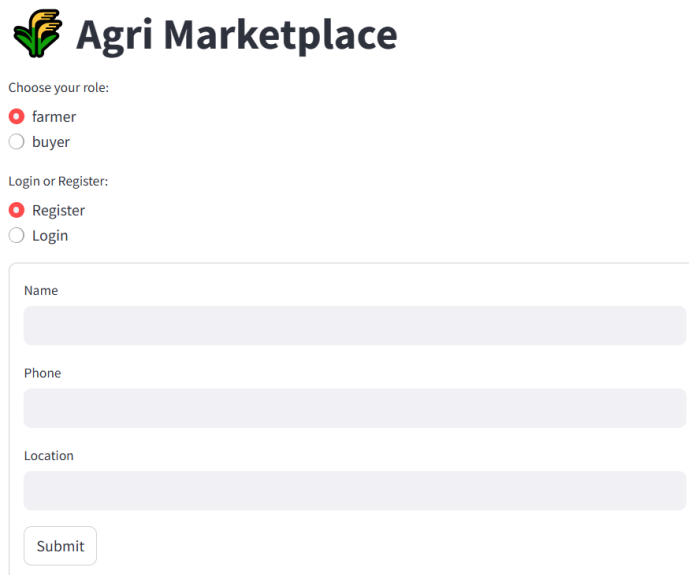
### **5.4 UI/UX Design**


The application's interface is designed to be farmer-friendly, multi-lingual, and intuitive for rural and urban users alike.

- **Mobile-First Design:** Responsive and optimized for mobile devices, especially for rural users who primarily use smartphones.
- **Simple Navigation:** Clean layouts with icon-based navigation for low-literacy users.
- **Color Schemes and Accessibility:** Uses high-contrast colors for easy visibility and supports screen readers.

## 6. Results and Demonstration

### 6.1 Screenshots



 **Agri Marketplace**

Choose your role:

☒ farmer  
☐ buyer

Login or Register:

☒ Register  
☐ Login

Name

Phone

Location

**Img.1. The main dashboard**

## Farmer Login / Register

Choose Action

☐ Register

☒ Login

Name

Ravi Kumar

Phone

9876543210

Location

Karnataka

Press Enter to submit form

Continue

**Img 2. Farmer login or registration done at ease**

### My Crop Listings

#### Add New Crop

Crop Name

Tomatoes

Quantity

250

- +

Price

18

- +

Add Crop

Crop listed successfully.

### Orders Received

No orders yet.

**Img 3. Adding data into 'Farmers' Table**

## 6.2 Code Snippets

```

1 import streamlit as st
2 from datetime import datetime
3
4 # Initialize session state
5 for key in ["user_type", "users", "crops", "orders", "current_user"]:
6     if key not in st.session_state:
7         st.session_state[key] = []
8     if key == "users":
9         st.session_state[key] = {"farmer": [], "buyer": []}
10    elif key == "crops":
11        st.session_state[key] = [] # List of dicts: {"farmer", "crop", "qty", "price", "location", "id"}
12    elif key == "orders":
13        st.session_state[key] = [] # List of dicts: {"buyer", "crop_id", "qty", "date"}
14
15 # Helper functions
16 def get_crop_by_id(crop_id):
17     for crop in st.session_state.crops:
18         if crop["id"] == crop_id:
19             return crop
20     return None
21
22 def reduce_crop_quantity(crop_id, qty_ordered):
23     for crop in st.session_state.crops:
24         if crop["id"] == crop_id and crop["qty"] >= qty_ordered:
25             crop["qty"] -= qty_ordered
26             return True
27     return False
28
29 def home():
30     st.title("🌾 Agri Marketplace")
31     st.write("Select your role to continue:")
32     if st.button("👤 Farmer"):
33         st.session_state["user_type"] = "farmer"
34     elif st.button("👤 Buyer"):
35         st.session_state["user_type"] = "buyer"
36
37 def login_register():
38     st.subheader(f"{st.session_state.user_type.title()} Login / Register")
39     tab = st.radio("Choose Action", ["Register", "Login"])
40
41 with st.form("auth_form"):
42     name = st.text_input("Name")
43     phone = st.text_input("Phone")
44     location = st.text_input("Location")
45     submit = st.form_submit_button("Continue")
46     if submit:
47         user_data = {"name": name, "phone": phone, "location": location}
48         if tab == "Register":
49             st.session_state.users[st.session_state.user_type].append(user_data)
50             st.success("Registered successfully!")
51             st.session_state.current_user = user_data
52
53 def farmer_dashboard():
54     st.title(f"👤 Farmer Dashboard - {st.session_state.current_user['name']}")
55     if st.button("🏠 Back to Home"):
56         st.session_state.current_user = None
57         st.session_state.user_type = None
58         return
59
60 st.subheader("📋 My Crop Listings")
61 my_crops = [c for c in st.session_state.crops if c["farmer"] == st.session_state.current_user["name"]]
62 for crop in my_crops:
63     st.write(f"🌾 {crop['crop']} | Qty: {crop['qty']} | ₹{crop['price']} per unit")
64     col1, col2 = st.columns(2)
65     with col1:
66         if st.button(f"✏️ Edit {crop['id']}"):
67             st.session_state["edit_crop_id"] = crop["id"]
68     with col2:
69         if st.button(f"❌ Delete {crop['id']}"):
70             st.session_state.crops.remove(crop)
71             st.success("Crop removed.")
72             st.rerun()
73
74 if "edit_crop_id" in st.session_state:
75     crop = get_crop_by_id(st.session_state["edit_crop_id"])
76     st.subheader("Edit Crop")
77     with st.form("edit_crop"):
78         crop_name = st.text_input("Crop", crop["crop"])
79         qty = st.number_input("Quantity", min_value=1, value=crop["qty"])

```

```

79     price = st.number_input("Price", min_value=1, value=crop["price"])
80     submitted = st.form_submit_button("Update")
81     if submitted:
82         crop["crop"] = crop_name
83         crop["qty"] = qty
84         crop["price"] = price
85         st.success("Updated successfully.")
86         del st.session_state["edit_crop_id"]
87         st.rerun()
88
89     st.subheader("➕ Add New Crop")
90     with st.form("add_crop"):
91         crop = st.text_input("Crop Name")
92         qty = st.number_input("Quantity", min_value=1)
93         price = st.number_input("Price", min_value=1)
94         submitted = st.form_submit_button("Add Crop")
95         if submitted:
96             st.session_state.crops.append({
97                 "id": len(st.session_state.crops) + 1,
98                 "farmer": st.session_state.current_user["name"],
99                 "location": st.session_state.current_user["location"],
100                 "crop": crop,
101                 "qty": qty,
102                 "price": price
103             })
104             st.success("Crop listed successfully.")
105
106     st.subheader("📦 Orders Received")
107     my_orders = [o for o in st.session_state.orders if get_crop_by_id(o["crop_id"]) and get_crop_by_id(o["crop_id"])["farmer"] ==
108 st.session_state.current_user["name"]]
109     if my_orders:
110         for o in my_orders:
111             crop = get_crop_by_id(o["crop_id"])
112             st.write(f"📦 {o['buyer']} ordered {o['qty']} of {crop['crop']} on {o['date']}")
113     else:
114         st.info("No orders yet.")
115
116 def buyer_dashboard():
117     st.title(f"👤 Buyer Dashboard - {st.session_state.current_user['name']}")
118     if st.button("🏠 Back to Home"):
119         st.session_state.current_user = None
120         st.session_state.user_type = None
121         return
122
123     st.subheader("🌾 Available Crops")
124     crops = [c for c in st.session_state.crops if c["qty"] > 0]
125     for crop in crops:
126         st.write(f"🌾 {crop['crop']} | Qty: {crop['qty']} | ₹{crop['price']} | Farmer: {crop['farmer']} | Location: {crop['location']}")
127         with st.form(f"order_crop_{crop['id']}"):
128             qty = st.number_input(f"Qty to order (Max {crop['qty']})", min_value=1, max_value=crop["qty"], key=f"order_qty_{crop['id']}")
129             if st.form_submit_button("Place Order"):
130                 reduce_crop_quantity(crop["id"], qty)
131                 st.session_state.orders.append({
132                     "buyer": st.session_state.current_user["name"],
133                     "crop_id": crop["id"],
134                     "qty": qty,
135                     "date": datetime.now().strftime("%Y-%m-%d %H:%M")
136                 })
137                 st.success("Order placed successfully.")
138                 st.rerun()
139
140     st.subheader("📦 My Orders")
141     my_orders = [o for o in st.session_state.orders if o["buyer"] == st.session_state.current_user["name"]]
142     for o in my_orders:
143         crop = get_crop_by_id(o["crop_id"])
144         st.write(f"📦 {o['qty']} of {crop['crop']} from {crop['farmer']} on {o['date']}")
145
146 # --- APP LOGIC FLOW ---
147 if not st.session_state.user_type:
148     home()
149 elif not st.session_state.current_user:
150     login_register()
151 elif st.session_state.user_type == "farmer":
152     farmer_dashboard()
153 elif st.session_state.user_type == "buyer":
154     buyer_dashboard()

```

## 7. Challenges and limitations

### 7.1 Challenges Encountered

## 1. Database Integration

Initially, the app was built with Streamlit using in-memory structures like `st.session_state`. Moving to MySQL required complete restructuring like all lists (`users`, `crops`, `orders`) had to be replaced with persistent database tables and each CRUD (Create, Read, Update, Delete) operation had to be rewritten with SQL queries using the `mysql.connector` or `SQLAlchemy`.

Specific challenges were ensuring data consistency between UI actions and MySQL backend.

Writing safe, parameterized SQL queries to avoid injection and streamlit doesn't maintain a persistent database connection; reconnecting per request sometimes led to timeouts or **MySQL server has gone away with errors**.

Synchronizing database state with session state in Streamlit became necessary for real-time updates.

## 2. User Authentication & State Management

Streamlit does not support user authentication or session management out-of-the-box. So the challenges faced were, a user's session resets when the app reloads (e.g., on pressing a button), so their login info must be stored carefully. All state (e.g., current user, role, crop being edited) had to be manually handled using `st.session_state`, increasing complexity. Without unique user IDs or login credentials, the system is prone to data duplication or impersonation.

Example problem: Two users with the same name could register — there's no way to differentiate between them.

## 3. Concurrency Issues

Streamlit reruns the entire script whenever a user interacts with the interface (clicks a button, enters text, etc.). In this case the real issue was to suppose two buyers attempt to place an order at the same time for the same crop. Both see the same available quantity.

Without row-level locking in SQL or transactional control, both might end up ordering more than available, leading to inconsistent data.

It was hard as streamlit isn't designed for high-concurrency apps.

Ensuring atomic operations in Python code (vs. in the SQL layer) is difficult.

## 4. Data Validation

There's no backend or frontend enforcement for correct user inputs.



The main causes were users could enter invalid data like Empty crop names, Negative quantities, Extremely high/low prices, Phone numbers in wrong format and users could also re-submit the same crop multiple times, leading to spam-like data.

It was challenging as streamlit forms are simplistic — no regex or form groups. Validations had to be hardcoded, increasing code complexity and duplication.

## 5. Deployment Constraints

The app runs fine locally, but deployment posed serious roadblocks in this case the challenges faced was streamlit Cloud doesn't support bundled MySQL. You need to host MySQL externally (e.g., Planet Scale, AWS RDS, Azure MySQL, etc.).

Connecting Streamlit to remote DBs involves firewall rules, SSL certificates, and credentials — which must be securely stored.

You can't keep DB credentials in the Python file. You need `.env` variables or Streamlit Secrets Manager.

Real example issue: If MySQL is hosted on localhost but the app is deployed on cloud, they can't talk to each other.

## 7.2 Current Limitations

### 1. No Authentication or Access Control

- What's missing: Users can sign up without email, password, or OTP verification. Anyone can impersonate another user by entering the same name and phone number. There's no role-based login security — a buyer can switch to farmer anytime and vice versa.
- Implications: Security Risk: Malicious users could tamper with or spam the system. No accountability for actions like fake orders or false listings.

## 2. Scalability

- Current limitation: All data is stored in flat MySQL tables with no indexing or performance optimization. SQL queries are simple **SELECT \*** or **WHERE** based, which becomes slow as data grows.
- Consequences:  
With thousands of crops or orders, queries will lag or timeout. Streamlit itself is not optimized for handling heavy concurrent user loads.

## 3. Limited Order Management

- What's lacking: Buyers can place orders but cannot cancel, modify, or track them. Farmers cannot approve or reject orders — orders are placed automatically.
- Implications: Real-world workflows (confirmation, payment, delivery) are not represented. Lack of clarity for buyers and sellers about transaction status.

## 4. Farmer-Buyer Communication

- What's not there is that, no way for a buyer to ask questions or negotiate. No chat, no contact sharing, no email/message notifications.
- Real-world challenge: Farmers often need to coordinate pickup/delivery. Buyers may want quality assurance or location-based queries — currently impossible.

## 5. User Interface Simplicity

- Observation: Streamlit offers a developer-focused UI — clean but not friendly for rural or mobile users. There's no mobile optimization, dropdowns, autocomplete, or visual crop images.
- Consequently, older or less tech-savvy users may struggle. Not usable as-is for a field-deployed app without training or redesign.

## 6. Security & Data Privacy

The insecure is, no encryption (e.g., passwords, phone numbers stored in plain text). No input sanitization — possible SQL injection risk. No audit trails for who placed/edited/deleted records. Risks are vulnerable to tampering or data leaks if hosted online. Cannot be used in a real-world context without compliance to security standards.

## 7.3 Mitigation Strategies

### 1. Authentication Layer

Add email/password login or phone number with OTP verification. Store credentials securely using hashing libraries (**bcrypt**, **hashlib**). Add role-based access so users cannot switch roles arbitrarily.

### 2. Database Optimization

Add indexes on key fields like **user\_id**, **crop\_id**, **order\_date**. Paginate queries (e.g., **LIMIT 20 OFFSET 0**) to prevent UI lag. Use connection pooling for MySQL to prevent timeouts.

### 3. Robust Order Workflow

Introduce order status: **Pending**, **Approved**, **Shipped**, **Cancelled**.

Let farmers approve orders before reducing stock. Allow buyers to track and cancel orders from their dashboard.

### 4. Add Communication Channels

Integrate third-party tools like Twilio, Firebase Chat, or even simple email alerts.

Display farmer/buyer contact numbers conditionally upon order placement.

### 5. Improve UX/UI

Use Streamlit Components or move to frameworks like React, Flutter, or Django for better UI. Add mobile responsiveness, local language support (Kannada, Hindi, etc.), and input validations.

### 6. Enhance Security

Implement SQL query sanitization (use parameterized queries). Hide sensitive data with `.env` files and Streamlit's secrets manager. Monitor user actions through logs and audits for future admin panels.

## 8. Conclusion

The Agri-Marketplace application successfully lays the groundwork for a digital platform that connects farmers and buyers, enabling transparent crop listings and straightforward order placement. By simplifying agricultural trade, it supports farmers in reaching a broader market and empowers buyers with direct access to produce. However, to transition from a prototype to a fully deployable system, the application must address critical areas like authentication, order management, communication, and security. With further enhancements, this platform has strong potential to become a scalable, secure, and farmer-friendly digital solution that contributes meaningfully to modern agricultural ecosystems.

## 9. Future Enhancements

1. **AI-Based Crop Demand Forecasting:** Use machine learning to predict high-demand crops based on location, season, and buyer trends. Help farmers decide what to grow and list based on projected demand and price insights.
2. **Blockchain for Transparent Transactions:** Implement blockchain to record all orders and transactions securely. Builds trust by offering tamper-proof logs of produce origin, buyer, and seller interactions.
3. **Voice Assistant in Local Language:** Add voice-based navigation and form filling in regional languages (e.g., Kannada, Hindi). Ideal for farmers with low literacy or limited typing experience.
4. **Crop Health and Image Verification:** Let farmers upload crop images and use AI to check quality, freshness, or detect disease signs. Buyers can see verified conditions of crops before placing orders.

5. Smart Logistics Integration: Automatically suggest or book local delivery services (bike, van) based on farmer location and crop quantity. Track real-time delivery status via GPS.
6. Carbon Credit Rewards: Offer farmers reward points for eco-friendly practices (organic crops, low pesticide use). Points can be redeemed for farming tools, subsidies, or discounts.
7. Offline Syncing with SMS Integration: Allow farmers to get SMS alerts for new orders or listings. Enable crop registration and order confirmation via simple SMS for areas with poor internet.
8. Farmer Co-op Mode: Let groups of farmers (cooperatives) register and sell under a common account. Increases bargaining power and makes bulk logistics easier.
9. Weather-Integrated Planning: Integrate local weather forecasts to help buyers and farmers plan harvest, listing, or delivery. Notify users of climate risks (rain, drought) affecting supply.
10. Dynamic Pricing Engine: Automatically adjust crop prices based on demand, remaining stock, and competitor listings — like a mini agri-eCommerce engine.

## 10. References

1. "The State of Indian Agriculture 2021–22."

Ministry of Agriculture & Farmers Welfare, Government of India.

<https://agricoop.nic.in/en/state-of-agriculture>

Overview of Indian agriculture and need for transparent supply chain solutions.

2. "AgriTech – Towards Transforming Indian Agriculture."

FICCI (Federation of Indian Chambers of Commerce and Industry), Yes Bank Report (2020).

<https://ficci.in/publication.asp?spid=23196>

Discusses technology-driven agriculture platforms and farmer-market linkage.

3. "Strengthening Farmer–Buyer Linkages in India: Evidence from Field Studies."

IFPRI (International Food Policy Research Institute), 2020.

<https://www.ifpri.org/publication/strengthening-farmer-buyer-linkages>

Focuses on reducing middlemen impact and improving direct sales.