PROJECT REPORT

ONLINE AUCTION SYSTEM

CONTRIBUTORS:

NAME: MANAS C BAVASKAR

REG. NO: 221080004

CLASS: S.Y. B.Tech. I.T.

MOBILE: +91 87796 21594

EMAIL: mcbavaskar_b22@it.vjti.ac.in

NAME: AARYA A BODAS

REG. NO: 221080008

CLASS: S.Y. B.Tech. I.T.

MOBILE: +91 98206 73543

EMAIL: aabodas_b22@it.vjti.ac.in

PROBLEM DEFINITIONS AND DATA MODELLING

Problem Statement:

Online auction systems have revolutionized the way people buy and sell goods, providing a convenient platform for users to engage in bidding activities remotely. Building a successful online auction platform entails careful consideration of the underlying database structure to facilitate efficient data management, scalability, and security. Our project delves into the design and implementation of a comprehensive database schema for an online auction system, with a focus on managing users, items, bids, auctions, and transactions effectively.

At the heart of the database structure lies the Users table, serving as the central repository for user information. The Users table establishes crucial relationships with other database entities, including Items, Bids, and Transactions, facilitating seamless data integration and user interaction throughout the auction process.

The Items table stores comprehensive information about items available for auction, enabling users to list their products and services. Real-time bidding functionality will be implemented to track the current highest bid for each item, ensuring users have access to the latest bidding activity.

When users place bids on items, the Bids table records and tracks bidding activity in real-time. The Bids table establishes relationships with the Users and Items tables, facilitating seamless data retrieval and bid management throughout the auction process.

The Auctions table manages the auction process for items, providing essential details about each auction event. The Auctions table is linked to the Items table, ensuring synchronization between item listings and auction events, with bid updates and auction status changes reflected in real-time.

Completed transactions resulting from winning bids are recorded in the Transactions table. The Transactions table establishes relationships with the Users and Items tables, facilitating seamless integration of transaction data with user and item profiles.

ER Diagram:

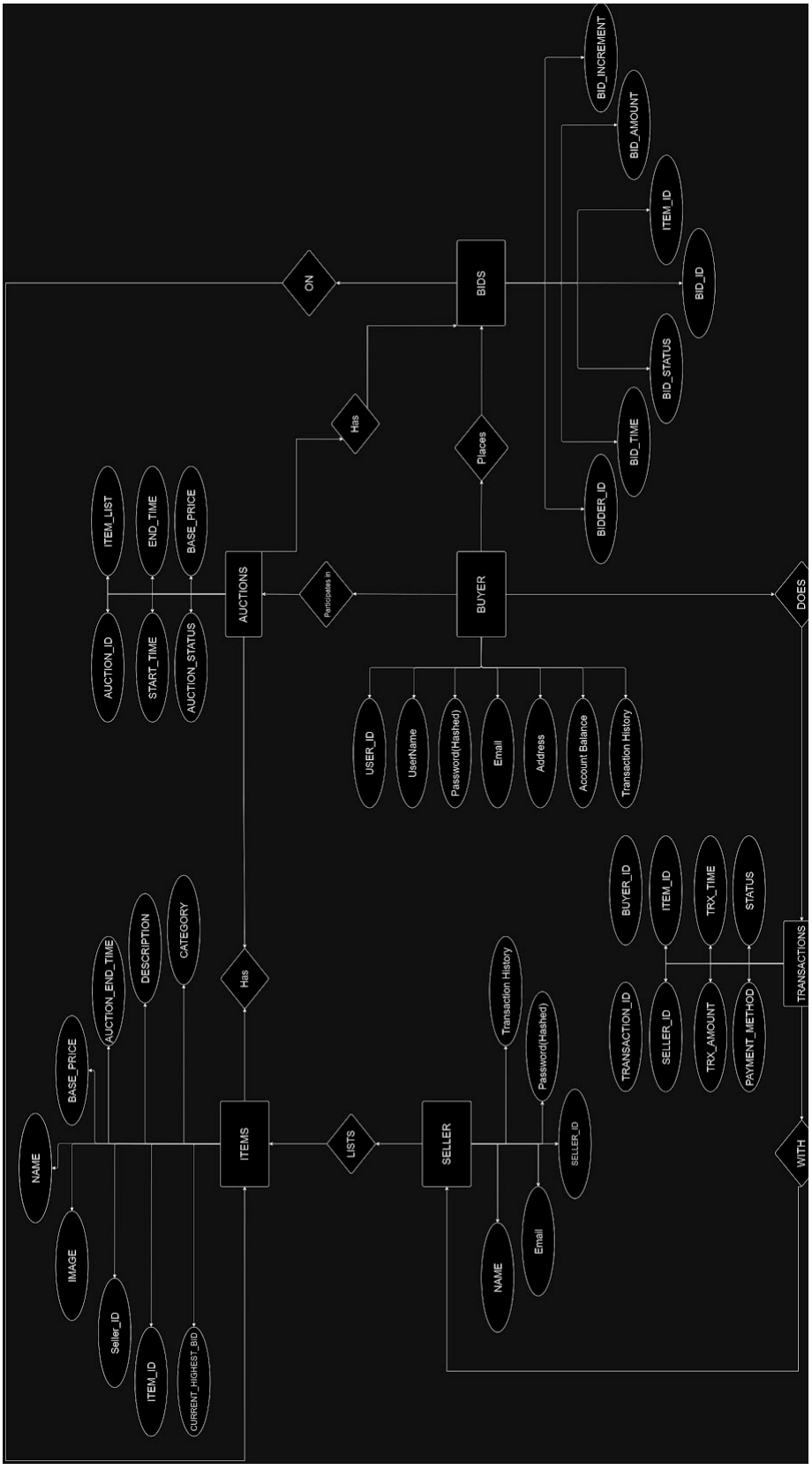


FIGURE 1.1

Revised Problem Statement:

This project delves into the design and implementation of a comprehensive database schema for an online auction system, with a focus on managing users, items, bids, auctions, and transactions effectively.

At the heart of the database structure lies the Users table, serving as the central repository for user information. Each user is assigned a unique User_ID, and essential details such as Username, Password, Email, and Address are recorded. Additional fields for preferred payment methods and transaction history enhance user profiles. The Users table establishes crucial relationships with other database entities, including Items, Bids, and Transactions, facilitating seamless data integration and user interaction throughout the auction process.

The Items table stores comprehensive information about items available for auction, enabling users to list their products and services. Each item is assigned a unique Item_ID, and attributes such as Seller_ID, Item_Name, Description, Starting_Price, and Auction_End_Time are recorded. Additional fields for item condition, category, and images enrich item listings, making them more attractive to potential buyers. Real-time bidding functionality will be implemented to track the current highest bid for each item, ensuring users have access to the latest bidding activity.

When users place bids on items, the Bids table records and tracks bidding activity in real-time. Each bid is assigned a unique Bid_ID, and details such as Bidder_ID, Item_ID, Bid_Amount, and Bid_Time are captured. Supplementary fields for bid status and bid increment provide additional insights into bid progress and auction dynamics. The Bids table establishes relationships with the Users and Items tables, facilitating seamless data retrieval and bid management throughout the auction process.

The Auctions table manages the auction process for items, providing essential details about each auction event. Attributes such as Auction_ID, Item_ID, Auction_Start_Time, and Auction_End_Time are recorded to track auction timelines and status. Additional fields for auction status and reserve price enhance auction management and decision-making. The Auctions table is linked to the Items table, ensuring synchronization between item listings and auction events, with bid updates and auction status changes reflected in real-time.

Completed transactions resulting from winning bids are recorded in the Transactions table. Each transaction is assigned a unique Transaction_ID, and details such as Buyer_ID, Seller_ID, Item_ID, Transaction_Amount, and Transaction_Time are captured. Additional fields for payment method and transaction status streamline transaction processing and management. The Transactions table establishes relationships with the Users and Items tables, facilitating seamless integration of transaction data with user and item profiles.

The project will be implemented using the MERN stack, with the only modification being that MariaDB will be used as an SQL database instead of MongoDB, which is a non-SQL database. The buyer will be able to view items listed on various auctions. The seller will be able to view his/her listed products and will also be able to list more products. The admin will be able to view the history of auctions, transactions, and bids placed on the web app. Data Visualization and Analysis will also be done using Power BI, which will be connected to the MariaDB server, facilitating real-time updates to it.

Revised ER Diagram:

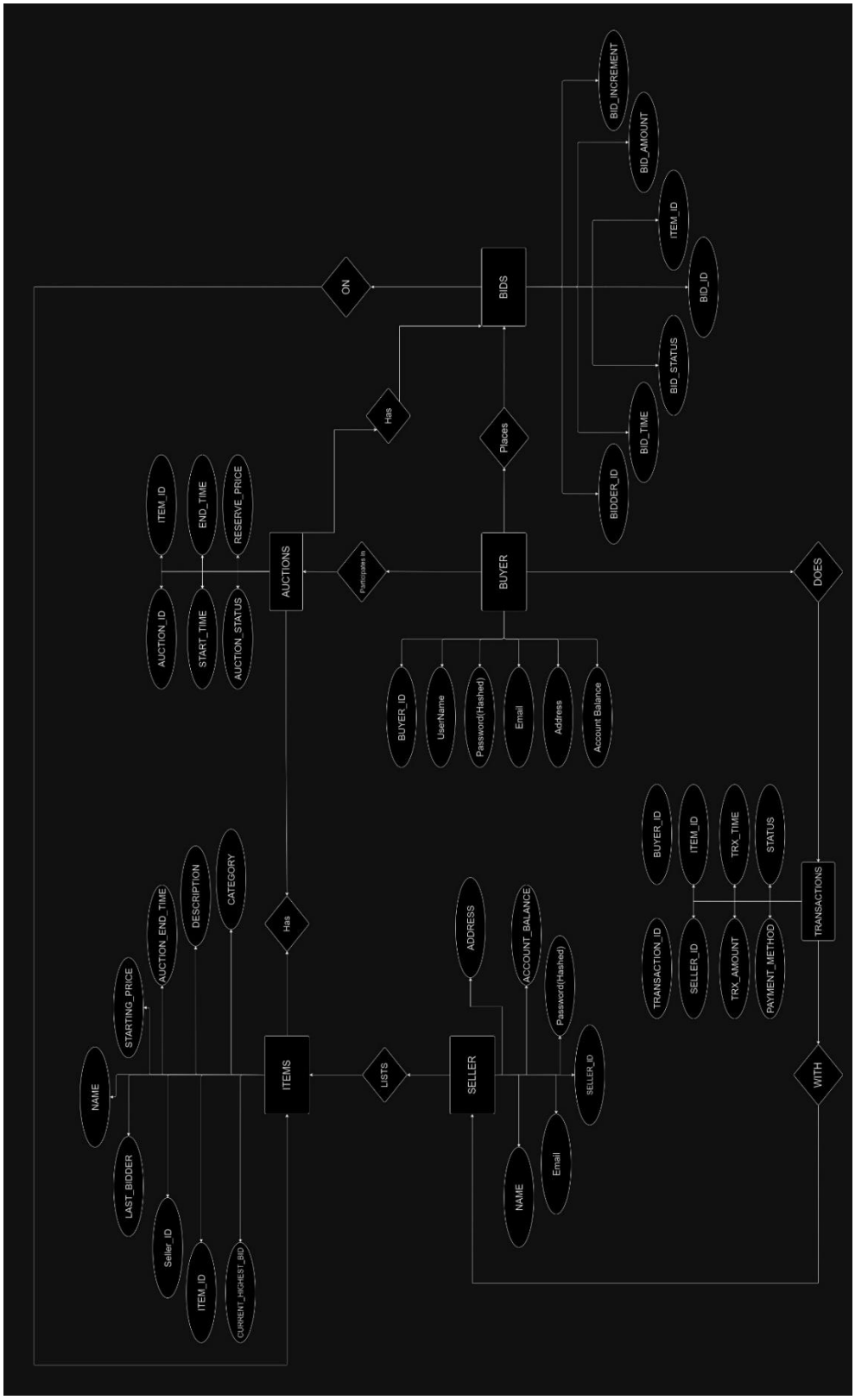


FIGURE 1.2

DATABASE DESIGN, DATA DICTIONARY AND QUERY LANGUAGE

Schema Design:

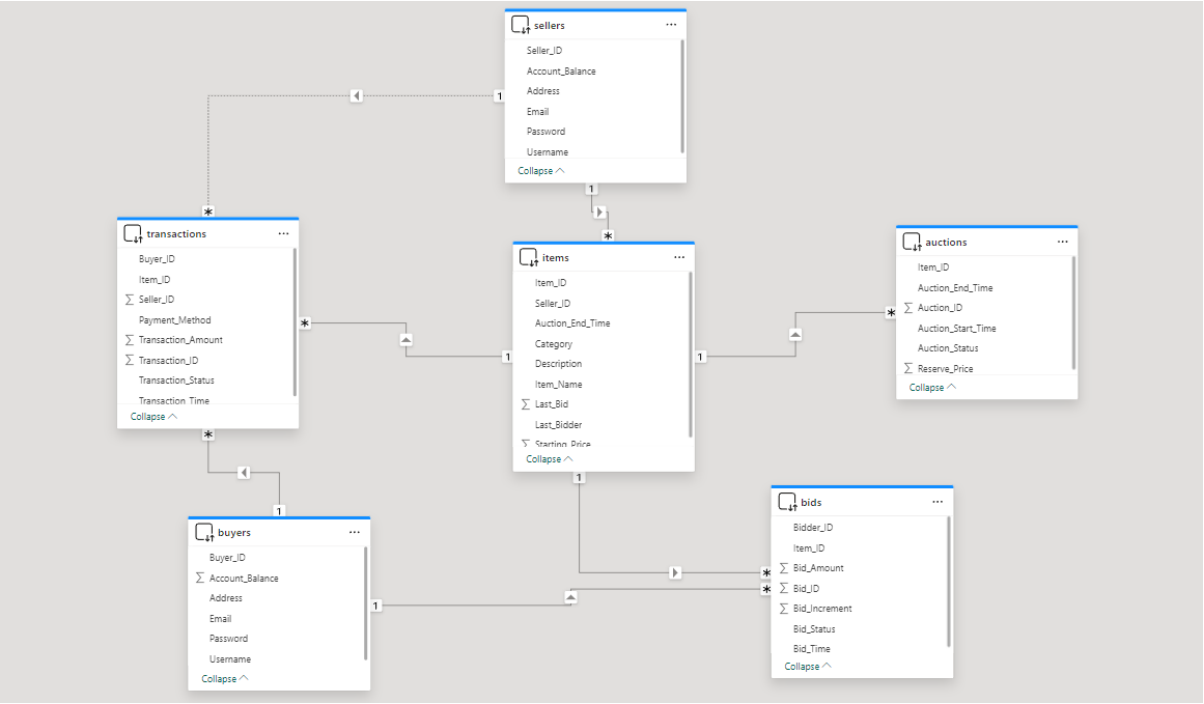


FIGURE 2.0

Data Dictionary:

Database	Table	Column	Column Type	Is Nullable	Default Value	Key	Extra	Comment
dbms	auctions	Auction_ID	int(11)	NO	NULL	PRI		
dbms	auctions	Item_ID	int(11)	NO	NULL	MUL		
dbms	auctions	Auction_Start_Time	datetime	NO	NULL	MUL		
dbms	auctions	Auction_End_Time	datetime	NO	NULL	MUL		
dbms	auctions	Auction_Status	varchar(20)	NO	NULL			
dbms	auctions	Reserve_Price	decimal(10,2)	YES	0.00			
dbms	bids	Bid_ID	int(11)	NO	NULL	PRI		
dbms	bids	Bidder_ID	int(11)	NO	NULL	MUL		
dbms	bids	Item_ID	int(11)	NO	NULL	MUL		
dbms	bids	Bid_Amount	decimal(10,2)	NO	NULL			
dbms	bids	Bid_Time	datetime	NO	NULL	MUL		
dbms	bids	Bid_Status	varchar(20)	NO	NULL			
dbms	bids	Bid_Increment	decimal(10,2)	YES	0.00			
dbms	buyers	Buyer_ID	int(11)	NO	NULL	PRI		
dbms	buyers	Username	varchar(50)	NO	NULL	MUL		
dbms	buyers	Password	varchar(255)	NO	NULL			
dbms	buyers	Email	varchar(100)	NO	NULL	MUL		
dbms	buyers	Address	varchar(255)	NO	NULL			
dbms	buyers	Account_Balance	decimal(65,20)	YES	0.00000000000000000000			
dbms	items	Item_ID	int(11)	NO	NULL	PRI		
dbms	items	Seller_ID	int(11)	NO	NULL	MUL		
dbms	items	Item_Name	varchar(255)	NO	NULL			
dbms	items	Description	text	YES	NULL			
dbms	items	Starting_Price	decimal(10,2)	NO	0.00			
dbms	items	Auction_End_Time	datetime	NO	NULL	MUL		
dbms	items	Category	varchar(50)	NO	NULL	MUL		
dbms	items	Last_Bidder	varchar(255)	YES	NULL			
dbms	items	Last_bid	decimal(10,2)	YES	NULL			
dbms	sellers	Seller_ID	int(11)	NO	NULL	PRI		
dbms	sellers	Username	varchar(50)	NO	NULL	MUL		
dbms	sellers	Password	varchar(255)	NO	NULL			
dbms	sellers	Email	varchar(100)	NO	NULL			
dbms	sellers	Address	varchar(255)	NO	NULL			
dbms	sellers	Account_Balance	decimal(65,20)	YES	0.00000000000000000000			
dbms	transactions	Transaction_ID	int(11)	NO	NULL	PRI		
dbms	transactions	Buyer_ID	int(11)	NO	NULL	MUL		
dbms	transactions	Seller_ID	int(11)	NO	NULL	MUL		
dbms	transactions	Item_ID	int(11)	NO	NULL	MUL		
dbms	transactions	Transaction_Amount	decimal(10,2)	NO	NULL			
dbms	transactions	Transaction_Time	datetime	NO	NULL	MUL		
dbms	transactions	Payment_Method	varchar(50)	NO	NULL			
dbms	transactions	Transaction_Status	varchar(20)	NO	NULL			

42 rows in set (0.009 sec)

Fig 2.1: Data Dictionary

TABLE_NAME	COLUMN_NAME
auctions	Auction_ID
bids	Bid_ID
buyers	Buyer_ID
items	Item_ID
sellers	Seller_ID
transactions	Transaction_ID

6 rows in set (0.001 sec)

Fig 2.2: Unique Keys

TABLE_NAME	COLUMN_NAME	CONSTRAINT_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
auctions	Item_ID	auctions_ibfk_1	items	Item_ID
bids	Bidder_ID	bids_ibfk_1	buyers	Buyer_ID
bids	Item_ID	bids_ibfk_2	items	Item_ID
items	Seller_ID	items_ibfk_1	sellers	Seller_ID
transactions	Buyer_ID	transactions_ibfk_1	buyers	Buyer_ID
transactions	Seller_ID	transactions_ibfk_2	sellers	Seller_ID
transactions	Item_ID	transactions_ibfk_3	items	Item_ID

7 rows in set (0.001 sec)

Fig 2.3: Foreign Keys

DDL for the Data Dictionary:

```
CREATE TABLE Buyers (
    Buyer_ID INT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL,
    Password VARCHAR(255) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    Address VARCHAR(255) NOT NULL,
    Account_Balance DECIMAL(10, 2) DEFAULT 0
);

CREATE TABLE Sellers (
    Seller_ID INT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL,
    Password VARCHAR(255) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    Address VARCHAR(255) NOT NULL,
    Account_Balance DECIMAL(10, 2) DEFAULT 0
);
```

```
CREATE TABLE Items (
    Item_ID INT PRIMARY KEY,
    Seller_ID INT NOT NULL,
    Item_Name VARCHAR(255) NOT NULL,
    Description TEXT,
    Starting_Price DECIMAL(10, 2) NOT NULL,
    Auction_End_Time DATETIME NOT NULL,
    Category VARCHAR(50) NOT NULL,
    Last_Bidder VARCHAR(255) DEFAULT NULL,
    Last_Bid DECIMAL(10, 2) DEFAULT NULL,
    FOREIGN KEY (Seller_ID) REFERENCES Sellers(Seller_ID)
);

CREATE TABLE Bids (
    Bid_ID INT PRIMARY KEY,
    Bidder_ID INT NOT NULL,
    Item_ID INT NOT NULL,
    Bid_Amount DECIMAL(10, 2) NOT NULL,
    Bid_Time DATETIME NOT NULL,
    Bid_Status VARCHAR(20) NOT NULL,
    Bid_Increment DECIMAL(10, 2) DEFAULT 0,
    FOREIGN KEY (Bidder_ID) REFERENCES Buyers(Buyer_ID),
    FOREIGN KEY (Item_ID) REFERENCES Items(Item_ID)
);
```

```
CREATE TABLE Auctions (
    Auction_ID INT PRIMARY KEY,
    Item_ID INT NOT NULL,
    Auction_Start_Time DATETIME NOT NULL,
    Auction_End_Time DATETIME NOT NULL,
    Auction_Status VARCHAR(20) NOT NULL,
    Reserve_Price DECIMAL(10, 2) DEFAULT 0,
    FOREIGN KEY (Item_ID) REFERENCES Items(Item_ID)
);

CREATE TABLE Transactions (
    Transaction_ID INT PRIMARY KEY,
    Buyer_ID INT NOT NULL,
    Seller_ID INT NOT NULL,
    Item_ID INT NOT NULL,
    Transaction_Amount DECIMAL(10, 2) NOT NULL,
    Transaction_Time DATETIME NOT NULL,
    Payment_Method VARCHAR(50) NOT NULL,
    Transaction_Status VARCHAR(20) NOT NULL,
    FOREIGN KEY (Buyer_ID) REFERENCES Buyers(Buyer_ID),
    FOREIGN KEY (Seller_ID) REFERENCES Sellers(Seller_ID),
    FOREIGN KEY (Item_ID) REFERENCES Items(Item_ID)
);
```

Queries that can be asked to the data tables and their solution:

1. How many buyers are there in the database?

```
MariaDB [dbms]> SELECT COUNT(*) AS num_buyers FROM Buyers;
+------------+
| num_buyers |
+------------+
|         50 |
+------------+
1 row in set (0.004 sec)
```

2. How many sellers are there in the database?

```
MariaDB [dbms]> SELECT COUNT(*) AS num_sellers FROM Sellers;
+-------------+
| num_sellers |
+-------------+
|          50 |
+-------------+
1 row in set (0.001 sec)
```

3. What is the total number of items listed for auction?

```
MariaDB [dbms]> SELECT COUNT(*) AS total_items FROM Items;
+-------------+
| total_items |
+-------------+
|          51 |
+-------------+
1 row in set (0.001 sec)
```

4. How many bids have been placed so far?

```
MariaDB [dbms]> SELECT COUNT(*) AS total_bids FROM Bids;
+------------+
| total_bids |
+------------+
|         50 |
+------------+
1 row in set (0.002 sec)
```

5. What is the average account balance of all buyers?

```
MariaDB [dbms]> SELECT AVG(Account_Balance) AS avg_buyer_balance FROM Buyers;
+------------------+
| avg_buyer_balance |
+------------------+
|      5192.000000 |
+------------------+
1 row in set (0.003 sec)
```

6. What is the average account balance of all sellers?

```
MariaDB [dbms]> SELECT AVG(Account_Balance) AS avg_seller_balance FROM Sellers;
+--------------------+
| avg_seller_balance |
+--------------------+
|       9884.000000 |
+--------------------+
1 row in set (0.001 sec)
```

7. How many auctions are currently active?

```
MariaDB [dbms]> SELECT COUNT(*) AS active_auctions
    -> FROM Auctions
    -> WHERE Auction_Status = 'Active';
+-----------------+
| active_auctions |
+-----------------+
|              29 |
+-----------------+
1 row in set (0.001 sec)
```

8. How many items have a starting price greater than $500?

```
MariaDB [dbms]> SELECT COUNT(*) AS items_above_500
    -> FROM Items
    -> WHERE Starting_Price > 500;
+-----------------+
| items_above_500 |
+-----------------+
|              49 |
+-----------------+
1 row in set (0.000 sec)
```

9. What is the highest bid amount placed on any item?

```
MariaDB [dbms]> SELECT MAX(Bid_Amount) AS highest_bid
    -> FROM Bids;
+-------------+
| highest_bid |
+-------------+
|    62000.00 |
+-------------+
1 row in set (0.001 sec)
```

;

10. Which item has received the highest bid amount?

```
MariaDB [dbms]> SELECT Item_ID, MAX(Bid_Amount) AS highest_bid
    -> FROM Bids
    -> GROUP BY Item_ID
    -> ORDER BY highest_bid DESC
    -> LIMIT 1;
+---------+-------------+
| Item_ID | highest_bid |
+---------+-------------+
|       9 |    62000.00 |
+---------+-------------+
1 row in set (0.011 sec)
```

11. How many items belong to the 'Electronics' category?

```
MariaDB [dbms]> SELECT COUNT(*) AS electronics_items
    -> FROM Items
    -> WHERE Category = 'Electronics';
+-------------------+
| electronics_items |
+-------------------+
|                14 |
+-------------------+
1 row in set (0.001 sec)
```

12. What is the total amount of all bids placed?

```
MariaDB [dbms]> SELECT SUM(Bid_Amount) AS total_bids_amount
    -> FROM Bids;
+-------------------+
| total_bids_amount |
+-------------------+
|         712650.00 |
+-------------------+
1 row in set (0.001 sec)
```

13. Which seller has the highest account balance?

```
MariaDB [dbms]> SELECT Seller_ID, MAX(Account_Balance) AS highest_balance
    -> FROM Sellers;
+-----------+-----------------+
| Seller_ID | highest_balance |
+-----------+-----------------+
|         1 |        12000.00 |
+-----------+-----------------+
1 row in set (0.000 sec)
```

14. Which buyer has the highest account balance?

```
MariaDB [dbms]> SELECT Buyer_ID, MAX(Account_Balance) AS highest_balance
    -> FROM Buyers;
+----------+-----------------+
| Buyer_ID | highest_balance |
+----------+-----------------+
|        1 |         7000.00 |
+----------+-----------------+
1 row in set (0.000 sec)
```

15. How many transactions have been completed successfully?

```
MariaDB [dbms]> SELECT COUNT(*) AS successful_transactions
    -> FROM Transactions
    -> WHERE Transaction_Status = 'Completed';
+-------------------------+
| successful_transactions |
+-------------------------+
|                      50 |
+-------------------------+
1 row in set (0.011 sec)
```

16. How many items have a bid status of 'Rejected'?

```
MariaDB [dbms]> SELECT COUNT(*) AS rejected_bids
    -> FROM Bids
    -> WHERE Bid_Status = 'Rejected';
+---------------+
| rejected_bids |
+---------------+
|             0 |
+---------------+
1 row in set (0.001 sec)
```

17. What is the total number of unique categories among all items?

```
MariaDB [dbms]> SELECT COUNT(DISTINCT Category) AS unique_categories
    -> FROM Items;
+-------------------+
| unique_categories |
+-------------------+
|                10 |
+-------------------+
1 row in set (0.002 sec)
```

18. What is the average bid amount placed on items?

```
MariaDB [dbms]> SELECT AVG(Bid_Amount) AS avg_bid_amount
    -> FROM Bids;
+----------------+
| avg_bid_amount |
+----------------+
|   14253.000000 |
+----------------+
1 row in set (0.001 sec)
```

19. How many bids have been placed in the last week?

```
MariaDB [dbms]> SELECT COUNT(*) AS bids_last_week
    -> FROM Bids
    -> WHERE Bid_Time >= CURDATE() - INTERVAL 7 DAY;
+----------------+
| bids_last_week |
+----------------+
|             50 |
+----------------+
1 row in set (0.000 sec)
```

20. How many transactions have been made using credit cards?

```
MariaDB [dbms]> SELECT COUNT(*) AS credit_card_transactions
    -> FROM Transactions
    -> WHERE Payment_Method = 'Credit Card';
+--------------------------+
| credit_card_transactions |
+--------------------------+
|                       11 |
+--------------------------+
1 row in set (0.000 sec)
```

21. What is the average starting price of items?

```
MariaDB [dbms]> SELECT AVG(Starting_Price) AS avg_starting_price
    -> FROM Items;
+--------------------+
| avg_starting_price |
+--------------------+
|       11918.647059 |
+--------------------+
1 row in set (0.000 sec)
```

22. How many items have auction end times within the next 24 hours?

```
MariaDB [dbms]> SELECT COUNT(*) AS items_ending_soon
    -> FROM Items
    -> WHERE Auction_End_Time <= NOW() + INTERVAL 1 DAY;
+-------------------+
| items_ending_soon |
+-------------------+
|                16 |
+-------------------+
1 row in set (0.002 sec)
```

23. How many buyers have an account balance greater than $1000?

```
MariaDB [dbms]> SELECT COUNT(*) AS wealthy_buyers
    -> FROM Buyers
    -> WHERE Account_Balance > 1000;
+---------------+
| wealthy_buyers |
+---------------+
|             50 |
+---------------+
1 row in set (0.001 sec)
```

24. How many sellers have an account balance less than $500?

```
MariaDB [dbms]> SELECT COUNT(*) AS struggling_sellers
    -> FROM Sellers
    -> WHERE Account_Balance < 500;
+--------------------+
| struggling_sellers |
+--------------------+
|                  0 |
+--------------------+
1 row in set (0.000 sec)
```

25. What is the total amount of all successful transactions?

```
MariaDB [dbms]> SELECT SUM(Transaction_Amount) AS total_successful_transactions
    -> FROM Transactions
    -> WHERE Transaction_Status = 'Completed';
+-----------------------------+
| total_successful_transactions |
+-----------------------------+
|                   123050.00 |
+-----------------------------+
1 row in set (0.001 sec)
```

26. How many items have a bid status of 'Accepted'?

```
MariaDB [dbms]> SELECT COUNT(*) AS accepted_bids
    -> FROM Bids
    -> WHERE Bid_Status = 'Accepted';
+---------------+
| accepted_bids |
+---------------+
|             0 |
+---------------+
1 row in set (0.001 sec)
```

27. What is the average transaction amount for payments made through UPI?

```
MariaDB [dbms]> SELECT AVG(Transaction_Amount) AS avg_upi_transaction
    -> FROM Transactions
    -> WHERE Payment_Method = 'UPI';
+---------------------+
| avg_upi_transaction |
+---------------------+
|         1600.000000 |
+---------------------+
1 row in set (0.000 sec)
```

28. What is the average number of bids per item?

```
MariaDB [dbms]> SELECT AVG(num_bids) AS avg_bids_per_item
    -> FROM (
    ->     SELECT Item_ID, COUNT(*) AS num_bids
    ->     FROM Bids
    ->     GROUP BY Item_ID
    -> ) AS bid_counts;
+-------------------+
| avg_bids_per_item |
+-------------------+
|            2.0000 |
+-------------------+
1 row in set (0.011 sec)
```

29. How many items have a description longer than 100 characters?

```
MariaDB [dbms]> SELECT COUNT(*) AS items_long_description
    -> FROM Items
    -> WHERE LENGTH(Description) > 100;
+------------------------+
| items_long_description |
+------------------------+
|                      0 |
+------------------------+
1 row in set (0.000 sec)
```

30. What is the average number of days an auction lasts?

```
MariaDB [dbms]> SELECT AVG(DATEDIFF(Auction_End_Time, Auction_Start_Time)) AS avg_auction_duration
    -> FROM Auctions;
+----------------------+
| avg_auction_duration |
+----------------------+
|               5.0000 |
+----------------------+
1 row in set (0.002 sec)
```

Insertion of Data Into Tables:

SQL-DML to Maniupulate Data for your ER Model

1. Buyers Table

```
MariaDB [dbms]> INSERT INTO Buyers (Buyer_ID, Username, Password, Email, Address, Account_Balance) VALUES
    -> (1, 'ananya_gupta', 'password123', 'ananya.gupta@gmail.com', '123 Main Street, City, Uttar Pradesh, India', 5000.00),
    -> (2, 'raj_sharma', 'p@ssw0rd!', 'raj.sharma@yahoo.com', '456 Elm Street, City, Maharashtra, India', 7000.00),
    -> (3, 'rahul.kumar', 'securepass', 'rahul.kumar@outlook.com', '789 Oak Street, City, West Bengal, India', 3000.00),

    -> (4, 'sangeeta25', 'myp@ssw0rd', 'sangeeta25@gmail.com', '101 Maple Street, City, Tamil Nadu, India', 6000.00),
    -> (5, 'vikram_singh', 'vikram123', 'vikram.singh@yahoo.com', '234 Pine Street, City, Karnataka, India', 4500.00),
    -> (6, 'sonali_89', '89$onali', 'sonali_89@outlook.com', '567 Cedar Street, City, Kerala, India', 5500.00),
    -> (7, 'amit_patil', 'p@t!l2023', 'amit.patil@gmail.com', '890 Birch Street, City, Punjab, India', 4800.00),
    -> (8, 'shreya.m', 'shreya@12345', 'shreya.m@yahoo.com', '1234 Willow Street, City, Uttar Pradesh, India', 6200.00),

    -> (9, 'nikhil_2000', 'nicenikhil', 'nikhil_2000@outlook.com', '5678 Cherry Street, City, Gujarat, India', 5100.00),

    -> (10, 'priya12', 'priyapassword', 'priya12@gmail.com', '9012 Walnut Street, City, Maharashtra, India', 5300.00),
    -> (11, 'arnav.kapoor', 'arnav123!', 'arnav.kapoor@yahoo.com', '3456 Sycamore Street, City, Tamil Nadu, India', 6800.00),
    -> (12, 'megha.sen', 'sen_megha', 'megha.sen@outlook.com', '7890 Ash Street, City, Karnataka, India', 4200.00),
    -> (13, 'vishal32', 'vishal#2024', 'vishal32@gmail.com', '123 Oak Avenue, City, Kerala, India', 5700.00),
    -> (14, 'anjali_11', 'anjali@2023', 'anjali_11@yahoo.com', '456 Elm Avenue, City, Punjab, India', 4900.00),
    -> (15, 'ritika.m', 'mritika!', 'ritika.m@outlook.com', '789 Cedar Avenue, City, West Bengal, India', 5500.00),
    -> (16, 'harshit123', 'harshit2023', 'harshit123@gmail.com', '101 Maple Avenue, City, Tamil Nadu, India', 6200.00),
    -> (17, 'sneha_04', 'snehap@ssw0rd', 'sneha_04@yahoo.com', '234 Pine Avenue, City, Karnataka, India', 4000.00),
    -> (18, 'amar_kumar', 'amar!kumar', 'amar_kumar@outlook.com', '567 Cedar Avenue, City, Kerala, India', 4700.00),
    -> (19, 'divya_09', 'divya09@123', 'divya_09@gmail.com', '890 Birch Avenue, City, Punjab, India', 5400.00),
    -> (20, 'rohit.b', 'rohitrohit', 'rohit.b@yahoo.com', '1234 Willow Avenue, City, Uttar Pradesh, India', 5800.00),
    -> (21, 'anu_1990', '1990anu!', 'anu_1990@outlook.com', '5678 Cherry Avenue, City, Maharashtra, India', 4900.00),
    -> (22, 'aditya31', 'aditya123!', 'aditya31@gmail.com', '9012 Walnut Avenue, City, West Bengal, India', 6500.00),
    -> (23, 'priyanka.m', 'priyankam@22', 'priyanka.m@yahoo.com', '3456 Sycamore Avenue, City, Tamil Nadu, India', 4700.00),
    -> (24, 'rakesh.singh', 'rakesh@12345', 'rakesh.singh@outlook.com', '7890 Ash Avenue, City, Karnataka, India', 5200.00),
    -> (25, 'shivani_g', 'shivani!23', 'shivani_g@gmail.com', '123 Oak Boulevard, City, Kerala, India', 4900.00),
    -> (26, 'ankit_08', 'ankit08@pass', 'ankit_08@yahoo.com', '456 Elm Boulevard, City, Punjab, India', 5800.00),
    -> (27, 'saurabh.k', 'saurabh@123', 'saurabh.k@outlook.com', '789 Cedar Boulevard, City, West Bengal, India', 4600.00),
    -> (28, 'nisha_45', 'nisha123!', 'nisha_45@gmail.com', '101 Maple Boulevard, City, Uttar Pradesh, India', 5300.00),
    -> (29, 'deepak_1987', 'd33pak@1987', 'deepak_1987@yahoo.com', '234 Pine Boulevard, City, Maharashtra, India', 4400.00),
    -> (30, 'mohit.kumar', 'kumarmohit', 'mohit.kumar@outlook.com', '567 Cedar Boulevard, City, Tamil Nadu, India', 5900.00),
    -> (31, 'pooja_18', 'poojapooja', 'pooja_18@gmail.com', '890 Birch Boulevard, City, Kerala, India', 5100.00),
    -> (32, 'vivek.p', 'vp@2023', 'vivek.p@yahoo.com', '1234 Willow Boulevard, City, Punjab, India', 5200.00),
    -> (33, 'tanvi_09', 'tanvi09!', 'tanvi_09@outlook.com', '3456 Sycamore Boulevard, City, West Bengal, India', 4800.00),
    -> (34, 'rohan_gupta', 'rohan!@123', 'rohan_gupta@gmail.com', '7890 Ash Boulevard, City, Uttar Pradesh, India', 4700.00),
    -> (35, 'shikha_34', 'shikha123', 'shikha_34@yahoo.com', '123 Oak Court, City, Maharashtra, India', 5400.00),
    -> (36, 'devansh.s', 'devansh123', 'devansh.s@outlook.com', '456 Elm Court, City, Tamil Nadu, India', 4600.00),
    -> (37, 'priyanka.k', 'kpriyanka', 'priyanka.k@gmail.com', '789 Cedar Court, City, Kerala, India', 5500.00),
    -> (38, 'akash_joshi', 'akash@12345', 'akash_joshi@yahoo.com', '101 Maple Court, City, Punjab, India', 5100.00),
    -> (39, 'neha12', 'nehap@ssw0rd', 'neha12@outlook.com', '234 Pine Court, City, West Bengal, India', 4800.00),
    -> (40, 'amit_23', 'amitamit', 'amit_23@gmail.com', '567 Cedar Court, City, Uttar Pradesh, India', 6000.00),
    -> (41, 'riya.45', 'riyariya', 'riya.45@yahoo.com', '890 Birch Court, City, Maharashtra, India', 5200.00),
    -> (42, 'harshita.s', 'harshitas', 'harshita.s@outlook.com', '1234 Willow Court, City, Tamil Nadu, India', 4700.00),

    -> (43, 'shubham_07', 'shubham07!', 'shubham_07@gmail.com', '3456 Sycamore Court, City, Kerala, India', 5600.00),
    -> (44, 'kritika.k', 'kritika@123', 'kritika.k@yahoo.com', '7890 Ash Court, City, Punjab, India', 5000.00),
    -> (45, 'rahul_1988', 'rahul@123', 'rahul_1988@outlook.com', '123 Oak Drive, City, Uttar Pradesh, India', 4900.00),
    -> (46, 'meena.sharma', 'meena@2023', 'meena.sharma@gmail.com', '456 Elm Drive, City, Maharashtra, India', 5100.00),

    -> (47, 'shivamg', 'shivam123!', 'shivamg@yahoo.com', '789 Cedar Drive, City, West Bengal, India', 5400.00),
    -> (48, 'kavya_123', 'kavya123!', 'kavya_123@outlook.com', '101 Maple Drive, City, Tamil Nadu, India', 4700.00),
    -> (49, 'anurag_86', 'anurag@2023', 'anurag_86@gmail.com', '234 Pine Drive, City, Kerala, India', 4800.00),
    -> (50, 'jyoti.11', 'jyoti@123', 'jyoti.11@yahoo.com', '567 Cedar Drive, City, Punjab, India', 5200.00);
Query OK, 50 rows affected (0.007 sec)
Records: 50  Duplicates: 0  Warnings: 0
```

2. Sellers Table

```
MariaDB [dbms]> INSERT INTO Sellers (Seller_ID, Username, Password, Email, Address, Account_Balance) VALUES
    -> (1, 'indian_crafts', 'crafty@123!', 'indian_crafts@gmail.com', '123 Main Street, City, Uttar Pradesh, India', 10000.00),
    -> (2, 'organic_farm', 'organic4life!', 'organic_farm@yahoo.com', '456 Elm Street, City, Maharashtra, India', 8500.00),
    -> (3, 'fashion_hub', 'fashionista2023', 'fashion_hub@outlook.com', '789 Oak Street, City, West Bengal, India', 12000.00),
    -> (4, 'spices_galore', 'spiceitup!', 'spices_galore@gmail.com', '101 Maple Street, City, Tamil Nadu, India', 9500.00),
    -> (5, 'tech_store', 'techgeek2023!', 'tech_store@yahoo.com', '234 Pine Street, City, Karnataka, India', 11000.00),
    -> (6, 'aroma_world', 'aroma2023world', 'aroma_world@outlook.com', '567 Cedar Street, City, Kerala, India', 8000.00),
    -> (7, 'greenery_decor', 'greendecor21', 'greenery_decor@gmail.com', '890 Birch Street, City, Punjab, India', 10500.00),
    -> (8, 'desi_dresses', 'ethnicchic!', 'desi_dresses@yahoo.com', '1234 Willow Street, City, Uttar Pradesh, India', 9000.00),
    -> (9, 'bookworm_books', 'booklover23', 'bookworm_books@outlook.com', '5678 Cherry Street, City, Gujarat, India', 11500.00),
    -> (10, 'home_appliances', 'homegadget123!', 'home_appliances@gmail.com', '9012 Walnut Street, City, Maharashtra, India', 9500.00),
    -> (11, 'crafty_creations', 'createcraft2023', 'crafty_creations@yahoo.com', '3456 Sycamore Street, City, Tamil Nadu, India', 10200.00),
    -> (12, 'wellness_corner', 'wellnessliving!', 'wellness_corner@outlook.com', '7890 Ash Street, City, Karnataka, India', 8800.00),
    -> (13, 'organic_beauty', 'naturalbeauty1', 'organic_beauty@gmail.com', '123 Oak Avenue, City, Kerala, India', 10500.00),
    -> (14, 'garden_oasis', 'greenoasis!', 'garden_oasis@yahoo.com', '456 Elm Avenue, City, Punjab, India', 9200.00),
    -> (15, 'ethnic_attire', 'ethnicfashion21', 'ethnic_attire@outlook.com', '789 Cedar Avenue, City, West Bengal, India', 10000.00),
    -> (16, 'gadget_galaxy', 'techwiz!2023', 'gadget_galaxy@gmail.com', '101 Maple Avenue, City, Tamil Nadu, India', 11200.00),
    -> (17, 'fitness_freaks', 'fitnessfan123!', 'fitness_freaks@yahoo.com', '234 Pine Avenue, City, Karnataka, India', 9700.00),
    -> (18, 'kitchen_king', 'kingofkitchen!', 'kitchen_king@outlook.com', '567 Cedar Avenue, City, Kerala, India', 8800.00),
    -> (19, 'home_decor', 'decorstyle21!', 'home_decor@gmail.com', '890 Birch Avenue, City, Punjab, India', 9900.00),
    -> (20, 'beauty_boutique', 'beautyqueen2023', 'beauty_boutique@yahoo.com', '1234 Willow Avenue, City, Uttar Pradesh, India', 10800.00),
    -> (21, 'grocery_galore', 'freshgroceries!', 'grocery_galore@outlook.com', '5678 Cherry Avenue, City, Maharashtra, India', 9200.00),
    -> (22, 'gadget_hub', 'gadgetgenius21!', 'gadget_hub@gmail.com', '9012 Walnut Avenue, City, West Bengal, India', 9800.00),
    -> (23, 'fashionista', 'fashionforward!', 'fashionista@yahoo.com', '3456 Sycamore Avenue, City, Tamil Nadu, India', 10300.00),
    -> (24, 'handmade_hub', 'handmadeunique', 'handmade_hub@outlook.com', '7890 Ash Avenue, City, Karnataka, India', 9300.00),
    -> (25, 'pet_paradise', 'petloverforever!', 'pet_paradise@gmail.com', '123 Oak Boulevard, City, Kerala, India', 10600.00),
    -> (26, 'sports_corner', 'sportyathome!', 'sports_corner@yahoo.com', '456 Elm Boulevard, City, Punjab, India', 9400.00),
    -> (27, 'stationary_stop', 'writewell2023!', 'stationary_stop@outlook.com', '789 Cedar Boulevard, City, West Bengal, India', 9700.00),
    -> (28, 'fashion_fusion', 'fashionfusion21', 'fashion_fusion@gmail.com', '101 Maple Boulevard, City, Uttar Pradesh, India', 10100.00),
    -> (29, 'home_furnishings', 'homedecorlove!', 'home_furnishings@yahoo.com', '234 Pine Boulevard, City, Maharashtra, India', 8900.00),
    -> (30, 'organic_eats', 'eathealthy!2023', 'organic_eats@outlook.com', '567 Cedar Boulevard, City, Karnataka, India', 10900.00),
    -> (31, 'smart_gadgets', 'techsmart21!', 'smart_gadgets@gmail.com', '890 Birch Boulevard, City, Kerala, India', 9200.00),
    -> (32, 'trendy_tees', 'fashiontrendy123', 'trendy_tees@yahoo.com', '1234 Willow Boulevard, City, Punjab, India', 9800.00),
    -> (33, 'art_affair', 'artycreations2023', 'art_affair@outlook.com', '5678 Cherry Boulevard, City, Uttar Pradesh, India', 10100.00),
    -> (34, 'health_hub', 'healthylife!2023', 'health_hub@gmail.com', '9012 Walnut Boulevard, City, Maharashtra, India', 9500.00),
    -> (35, 'beauty_basics', 'basicbeauty21!', 'beauty_basics@yahoo.com', '3456 Sycamore Boulevard, City, West Bengal, India', 10700.00),
    -> (36, 'kitchen_korner', 'kornerkitchen123', 'kitchen_korner@outlook.com', '7890 Ash Boulevard, City, Tamil Nadu, India', 9300.00),
    -> (37, 'fashion_frenzy', 'fashionfrenzy21!', 'fashion_frenzy@gmail.com', '123 Oak Court, City, Karnataka, India', 10200.00),
    -> (38, 'pet_palace', 'petsarefamily!', 'pet_palace@yahoo.com', '456 Elm Court, City, Kerala, India', 9600.00),
    -> (39, 'tech_trends', 'techsavvy2023!', 'tech_trends@outlook.com', '789 Cedar Court, City, Punjab, India', 10800.00),
    -> (40, 'eco_store', 'ecofriendly21!', 'eco_store@gmail.com', '101 Maple Court, City, Uttar Pradesh, India', 9000.00),
    -> (41, 'fitness_fiesta', 'fitandfabulous23', 'fitness_fiesta@yahoo.com', '234 Pine Court, City, Maharashtra, India', 10400.00),
    -> (42, 'gourmet_corner', 'gourmetdelight!', 'gourmet_corner@outlook.com', '567 Cedar Court, City, West Bengal, India', 9400.00),
    -> (43, 'gadget_guru', 'gadgetguru2023!', 'gadget_guru@gmail.com', '890 Birch Court, City, Tamil Nadu, India', 10700.00),
    -> (44, 'home_haven', 'homecomforts!', 'home_haven@yahoo.com', '1234 Willow Court, City, Karnataka, India', 9200.00),
    -> (45, 'fashion_forward', 'fashionforward21!', 'fashion_forward@outlook.com', '5678 Cherry Court, City, Punjab, India', 10100.00),
    -> (46, 'garden_glory', 'gardenofbliss!', 'garden_glory@gmail.com', '9012 Walnut Court, City, Uttar Pradesh, India', 9300.00),
    -> (47, 'art_arena', 'artlovers2023!', 'art_arena@yahoo.com', '3456 Sycamore Court, City, Maharashtra, India', 10500.00),
    -> (48, 'health_harmony', 'harmoniouslife21!', 'health_harmony@outlook.com', '7890 Ash Court, City, West Bengal, India', 9600.00),
    -> (49, 'fashion_fabrics', 'fabriclovers!2023', 'fashion_fabrics@gmail.com', '123 Oak Drive, City, Tamil Nadu, India', 10900.00),
    -> (50, 'kitchen_kraft', 'kraftykitchen123!', 'kitchen_kraft@yahoo.com', '456 Elm Drive, City, Karnataka, India', 9100.00);
Query OK, 50 rows affected (0.006 sec)
Records: 50  Duplicates: 0  Warnings: 0
```

3. Items Table

```
MariaDB [dbms]> INSERT INTO Items (Item_ID, Seller_ID, Item_Name, Description, Starting_Price, Auction_End_Time, Category) VALUES
    -> (1, 1, 'Handcrafted Pot', 'Handmade terracotta pot with floral motifs', 600.00, '2024-05-10 18:00:00', 'Home & Decor'),
    -> (2, 2, 'Organic Spices', 'Assorted organic spices from Nilgiri farms', 350.00, '2024-05-12 12:00:00', 'Grocery'),
    -> (3, 3, 'Designer Saree', 'Pure silk saree with zari work', 1200.00, '2024-05-11 20:00:00', 'Fashion'),
    -> (4, 4, 'Smart LED TV', 'Samsung 55-inch QLED TV with 4K resolution', 25000.00, '2024-05-15 15:00:00', 'Electronics'),
    -> (5, 5, 'Handloom Rug', 'Traditional Persian-style handwoven rug', 900.00, '2024-05-14 10:00:00', 'Home & Decor'),
    -> (6, 6, 'Kitchen Mixer', 'Bajaj GX7 500W mixer grinder', 1500.00, '2024-05-09 08:00:00', 'Appliances'),
    -> (7, 7, 'Gaming Console', 'Sony PlayStation 5 gaming console', 45000.00, '2024-05-13 16:00:00', 'Electronics'),
    -> (8, 8, 'Leather Jacket', 'Genuine leather jacket by Levi\'s', 3000.00, '2024-05-11 14:00:00', 'Fashion'),
    -> (9, 9, 'DSLR Camera', 'Canon EOS 90D DSLR camera with kit lens', 60000.00, '2024-05-16 11:00:00', 'Electronics'),
    -> (10, 10, 'Antique Clock', 'Victorian-era mantel clock in mahogany', 1800.00, '2024-05-18 14:00:00', 'Collectibles'),
    -> (11, 11, 'Tea Set', 'Bone china tea set with floral patterns', 800.00, '2024-05-17 09:00:00', 'Home & Decor'),
    -> (12, 12, 'Fitness Tracker', 'Fitbit Charge 4 fitness tracker', 4000.00, '2024-05-10 10:00:00', 'Electronics'),
    -> (13, 13, 'Smartphone', 'OnePlus 9 Pro smartphone', 55000.00, '2024-05-14 12:00:00', 'Electronics'),
    -> (14, 14, 'Office Chair', 'Ergonomic mesh office chair by Herman Miller', 8000.00, '2024-05-16 15:00:00', 'Furniture'),
    -> (15, 15, 'Cocktail Dress', 'Black satin cocktail dress by Vera Wang', 7000.00, '2024-05-11 18:00:00', 'Fashion'),
    -> (16, 16, 'Air Purifier', 'Philips 2000 series air purifier', 10000.00, '2024-05-19 13:00:00', 'Appliances'),
    -> (17, 17, 'Bicycle', 'Firefox Bikes Cyclone 27.5T mountain bike', 12000.00, '2024-05-20 16:00:00', 'Sports & Outdoors'),
    -> (18, 18, 'Bookshelf', 'Wooden bookshelf with 5 shelves', 2500.00, '2024-05-12 11:00:00', 'Furniture'),
    -> (19, 19, 'Sneakers', 'Nike Air Max 270 sneakers', 5000.00, '2024-05-15 17:00:00', 'Fashion'),
    -> (20, 20, 'Smart Watch', 'Apple Watch Series 6', 35000.00, '2024-05-22 10:00:00', 'Electronics'),
    -> (21, 21, 'Backpack', 'Wildcraft 45L hiking backpack', 2000.00, '2024-05-23 12:00:00', 'Travel & Outdoor'),
    -> (22, 22, 'Sofa Set', 'L-shaped fabric sofa set', 18000.00, '2024-05-24 14:00:00', 'Furniture'),
    -> (23, 23, 'Smart Speaker', 'Amazon Echo Dot (4th Gen)', 3000.00, '2024-05-25 16:00:00', 'Electronics'),
    -> (24, 24, 'Wall Art', 'Canvas wall art of a sunset landscape', 1500.00, '2024-05-28 18:00:00', 'Home & Decor'),
    -> (25, 25, 'Sunglasses', 'Ray-Ban Aviator sunglasses', 4500.00, '2024-05-26 10:00:00', 'Fashion'),
    -> (26, 26, 'Laptop', 'Dell XPS 13 9300 laptop', 85000.00, '2024-05-30 12:00:00', 'Electronics'),
    -> (27, 27, 'Barbecue Grill', 'Weber Spirit II E-310 gas grill', 20000.00, '2024-05-29 15:00:00', 'Appliances'),
    -> (28, 28, 'Yoga Mat', 'Manduka PRO Yoga mat', 4000.00, '2024-05-31 09:00:00', 'Fitness'),
    -> (29, 29, 'Wall Clock', 'Seiko Wall Clock', 2500.00, '2024-06-01 11:00:00', 'Home & Decor'),
    -> (30, 30, 'Wireless Headphones', 'Sony WH-1000XM4 wireless headphones', 20000.00, '2024-06-02 13:00:00', 'Electronics'),
    -> (31, 31, 'Dinner Set', 'Corelle 16-piece dinner set', 6000.00, '2024-06-03 15:00:00', 'Home & Decor'),
    -> (32, 32, 'Running Shoes', 'Adidas Ultraboost running shoes', 6500.00, '2024-06-05 17:00:00', 'Fashion'),
    -> (33, 33, 'Coffee Maker', 'Philips 12-cup coffee maker', 3000.00, '2024-06-06 10:00:00', 'Appliances'),
    -> (34, 34, 'Backpack', 'American Tourister 30L laptop backpack', 2500.00, '2024-06-07 12:00:00', 'Travel & Outdoor'),
    -> (35, 35, 'Pendant Light', 'Industrial-style pendant light', 3500.00, '2024-06-08 14:00:00', 'Home & Decor'),
    -> (36, 36, 'DSLR Camera', 'Nikon D7500 DSLR camera with 18-140mm lens', 70000.00, '2024-06-09 16:00:00', 'Electronics'),
    -> (37, 37, 'Microwave Oven', 'Samsung 28L Convection Microwave', 10000.00, '2024-06-10 18:00:00', 'Appliances'),
    -> (38, 38, 'Gaming Mouse', 'Razer DeathAdder Elite gaming mouse', 4000.00, '2024-06-11 09:00:00', 'Electronics'),
    -> (39, 39, 'Travel Backpack', 'Osprey Farpoint 40 travel backpack', 8000.00, '2024-06-12 11:00:00', 'Travel & Outdoor'),
    -> (40, 40, 'Bluetooth Speaker', 'JBL Flip 5 portable Bluetooth speaker', 8000.00, '2024-06-13 13:00:00', 'Electronics'),
    -> (41, 41, 'Study Desk', 'IKEA MICKE desk with integrated storage', 5000.00, '2024-06-14 15:00:00', 'Furniture'),
    -> (42, 42, 'Running Shoes', 'Nike Air Zoom Pegasus 37 running shoes', 7500.00, '2024-06-15 17:00:00', 'Fashion'),
    -> (43, 43, 'Electric Kettle', 'Philips 1.5L electric kettle', 1500.00, '2024-06-16 10:00:00', 'Appliances'),
    -> (44, 44, 'Study Lamp', 'LED desk lamp with adjustable brightness', 1200.00, '2024-06-17 12:00:00', 'Home & Decor'),
    -> (45, 45, 'Bluetooth Earphones', 'Sony WF-1000XM3 Bluetooth earphones', 12000.00, '2024-06-18 14:00:00', 'Electronics'),
    -> (46, 46, 'Wall Mirror', 'Decorative round wall mirror', 2000.00, '2024-06-19 16:00:00', 'Home & Decor'),
    -> (47, 47, 'Yoga Mat', 'Liforme Yoga mat', 6000.00, '2024-06-20 18:00:00', 'Fitness'),
    -> (48, 48, 'Blender', 'NutriBullet Pro 900W blender', 5000.00, '2024-06-21 09:00:00', 'Appliances'),
    -> (49, 49, 'Laptop Backpack', 'Samsonite Kombi 17" laptop backpack', 4000.00, '2024-06-22 11:00:00', 'Travel & Outdoor'),
    -> (50, 50, 'Wall Art', 'Canvas print of city skyline', 3000.00, '2024-06-23 13:00:00', 'Home & Decor');
Query OK, 50 rows affected (0.006 sec)
Records: 50  Duplicates: 0  Warnings: 0
```

4. Bids Table

```
MariaDB [dbms]> INSERT INTO Bids (Bid_ID, Bidder_ID, Item_ID, Bid_Amount, Bid_Time, Bid_Status, Bid_Increment) VALUES
    -> (1, 1, 1, 700.00, '2024-05-09 14:30:00', 'Winning', 100.00),
    -> (2, 2, 1, 650.00, '2024-05-09 15:45:00', 'Outbid', 100.00),
    -> (3, 3, 2, 400.00, '2024-05-09 16:20:00', 'Winning', 50.00),
    -> (4, 4, 2, 450.00, '2024-05-09 17:00:00', 'Outbid', 50.00),
    -> (5, 5, 3, 1500.00, '2024-05-09 18:30:00', 'Won', 100.00),
    -> (6, 6, 3, 1450.00, '2024-05-09 19:00:00', 'Lost', 100.00),
    -> (7, 7, 4, 26000.00, '2024-05-09 20:00:00', 'Winning', 1000.00),
    -> (8, 8, 4, 25500.00, '2024-05-09 21:00:00', 'Outbid', 1000.00),
    -> (9, 9, 5, 1000.00, '2024-05-09 22:00:00', 'Winning', 100.00),
    -> (10, 10, 5, 950.00, '2024-05-09 23:00:00', 'Outbid', 100.00),
    -> (11, 11, 6, 1600.00, '2024-05-10 09:30:00', 'Winning', 200.00),
    -> (12, 12, 6, 1500.00, '2024-05-10 10:00:00', 'Outbid', 200.00),
    -> (13, 13, 7, 47000.00, '2024-05-10 11:30:00', 'Winning', 2000.00),
    -> (14, 14, 7, 46000.00, '2024-05-10 12:00:00', 'Outbid', 2000.00),
    -> (15, 15, 8, 3300.00, '2024-05-10 13:30:00', 'Winning', 300.00),
    -> (16, 16, 8, 3100.00, '2024-05-10 14:00:00', 'Outbid', 300.00),
    -> (17, 17, 9, 62000.00, '2024-05-10 15:30:00', 'Winning', 2000.00),
    -> (18, 18, 9, 61000.00, '2024-05-10 16:00:00', 'Outbid', 2000.00),
    -> (19, 19, 10, 1900.00, '2024-05-10 17:30:00', 'Winning', 100.00),
    -> (20, 20, 10, 1800.00, '2024-05-10 18:00:00', 'Outbid', 100.00),
    -> (21, 21, 11, 950.00, '2024-05-10 19:30:00', 'Winning', 50.00),
    -> (22, 22, 11, 900.00, '2024-05-10 20:00:00', 'Outbid', 50.00),
    -> (23, 23, 12, 4300.00, '2024-05-10 21:30:00', 'Winning', 300.00),
    -> (24, 24, 12, 4000.00, '2024-05-10 22:00:00', 'Outbid', 300.00),
    -> (25, 25, 13, 57000.00, '2024-05-10 23:30:00', 'Winning', 2000.00),
    -> (26, 26, 13, 55000.00, '2024-05-11 00:00:00', 'Outbid', 2000.00),
    -> (27, 27, 14, 8400.00, '2024-05-11 09:30:00', 'Winning', 400.00),
    -> (28, 28, 14, 8000.00, '2024-05-11 10:00:00', 'Outbid', 400.00),
    -> (29, 29, 15, 7700.00, '2024-05-11 11:30:00', 'Winning', 500.00),
    -> (30, 30, 15, 7200.00, '2024-05-11 12:00:00', 'Outbid', 500.00),
    -> (31, 31, 16, 11500.00, '2024-05-11 13:30:00', 'Winning', 500.00),
    -> (32, 32, 16, 11000.00, '2024-05-11 14:00:00', 'Outbid', 500.00),
    -> (33, 33, 17, 15000.00, '2024-05-11 15:30:00', 'Winning', 1000.00),
    -> (34, 34, 17, 14000.00, '2024-05-11 16:00:00', 'Outbid', 1000.00),
    -> (35, 35, 18, 3000.00, '2024-05-11 17:30:00', 'Winning', 200.00),
    -> (36, 36, 18, 2800.00, '2024-05-11 18:00:00', 'Outbid', 200.00),
    -> (37, 37, 19, 5500.00, '2024-05-11 19:30:00', 'Winning', 300.00),
    -> (38, 38, 19, 5200.00, '2024-05-11 20:00:00', 'Outbid', 300.00),
    -> (39, 39, 20, 37000.00, '2024-05-11 21:30:00', 'Winning', 2000.00),
    -> (40, 40, 20, 35000.00, '2024-05-11 22:00:00', 'Outbid', 2000.00),
    -> (41, 41, 21, 2100.00, '2024-05-11 23:30:00', 'Winning', 100.00),
    -> (42, 42, 21, 2000.00, '2024-05-12 00:00:00', 'Outbid', 100.00),
    -> (43, 43, 22, 19000.00, '2024-05-12 09:30:00', 'Winning', 1000.00),
    -> (44, 44, 22, 18000.00, '2024-05-12 10:00:00', 'Outbid', 1000.00),
    -> (45, 45, 23, 3500.00, '2024-05-12 11:30:00', 'Winning', 200.00),
    -> (46, 46, 23, 3300.00, '2024-05-12 12:00:00', 'Outbid', 200.00),
    -> (47, 47, 24, 1800.00, '2024-05-12 13:30:00', 'Winning', 100.00),
    -> (48, 48, 24, 1700.00, '2024-05-12 14:00:00', 'Outbid', 100.00),
    -> (49, 49, 25, 41000.00, '2024-05-12 15:30:00', 'Winning', 2000.00),
    -> (50, 50, 25, 39000.00, '2024-05-12 16:00:00', 'Outbid', 2000.00);
Query OK, 50 rows affected (0.006 sec)
Records: 50  Duplicates: 0  Warnings: 0
```

5. Auctions Table

```
MariaDB [dbms]> INSERT INTO Auctions (Auction_ID, Item_ID, Auction_Start_Time, Auction_End_Time, Auction_Status, Reserve_Price) VALUES
    -> (1, 1, '2024-04-10 08:00:00', '2024-04-15 08:00:00', 'Active', 800.00),
    -> (2, 2, '2024-04-11 10:00:00', '2024-04-16 10:00:00', 'Closed', 400.00),
    -> (3, 3, '2024-04-12 12:00:00', '2024-04-17 12:00:00', 'Pending', 1500.00),
    -> (4, 4, '2024-04-13 14:00:00', '2024-04-18 14:00:00', 'Cancelled', 24000.00),
    -> (5, 5, '2024-04-14 16:00:00', '2024-04-19 16:00:00', 'Active', 700.00),
    -> (6, 6, '2024-04-15 18:00:00', '2024-04-20 18:00:00', 'Active', 1000.00),
    -> (7, 7, '2024-04-16 20:00:00', '2024-04-21 20:00:00', 'Closed', 300.00),
    -> (8, 8, '2024-04-17 22:00:00', '2024-04-22 22:00:00', 'Pending', 5000.00),
    -> (9, 9, '2024-04-18 08:00:00', '2024-04-23 08:00:00', 'Active', 1800.00),
    -> (10, 10, '2024-04-19 10:00:00', '2024-04-24 10:00:00', 'Active', 1200.00),
    -> (11, 11, '2024-04-20 12:00:00', '2024-04-25 12:00:00', 'Closed', 700.00),
    -> (12, 12, '2024-04-21 14:00:00', '2024-04-26 14:00:00', 'Closed', 3000.00),
    -> (13, 13, '2024-04-22 16:00:00', '2024-04-27 16:00:00', 'Active', 2500.00),
    -> (14, 14, '2024-04-23 18:00:00', '2024-04-28 18:00:00', 'Active', 900.00),
    -> (15, 15, '2024-04-24 20:00:00', '2024-04-29 20:00:00', 'Closed', 400.00),
    -> (16, 16, '2024-04-25 22:00:00', '2024-04-30 22:00:00', 'Active', 1500.00),
    -> (17, 17, '2024-04-26 08:00:00', '2024-05-01 08:00:00', 'Pending', 800.00),
    -> (18, 18, '2024-04-27 10:00:00', '2024-05-02 10:00:00', 'Active', 2200.00),
    -> (19, 19, '2024-04-28 12:00:00', '2024-05-03 12:00:00', 'Active', 1200.00),
    -> (20, 20, '2024-04-29 14:00:00', '2024-05-04 14:00:00', 'Closed', 1700.00),
    -> (21, 21, '2024-04-30 16:00:00', '2024-05-05 16:00:00', 'Active', 800.00),
    -> (22, 22, '2024-05-01 18:00:00', '2024-05-06 18:00:00', 'Active', 3000.00),
    -> (23, 23, '2024-05-02 20:00:00', '2024-05-07 20:00:00', 'Closed', 2500.00),
    -> (24, 24, '2024-05-03 22:00:00', '2024-05-08 22:00:00', 'Active', 1800.00),
    -> (25, 25, '2024-05-04 08:00:00', '2024-05-09 08:00:00', 'Pending', 4000.00),
    -> (26, 26, '2024-05-05 10:00:00', '2024-05-10 10:00:00', 'Active', 3500.00),
    -> (27, 27, '2024-05-06 12:00:00', '2024-05-11 12:00:00', 'Active', 4200.00),
    -> (28, 28, '2024-05-07 14:00:00', '2024-05-12 14:00:00', 'Closed', 1900.00),
    -> (29, 29, '2024-05-08 16:00:00', '2024-05-13 16:00:00', 'Active', 2300.00),
    -> (30, 30, '2024-05-09 18:00:00', '2024-05-14 18:00:00', 'Active', 1400.00),
    -> (31, 31, '2024-05-10 20:00:00', '2024-05-15 20:00:00', 'Closed', 1600.00),
    -> (32, 32, '2024-05-11 22:00:00', '2024-05-16 22:00:00', 'Active', 800.00),
    -> (33, 33, '2024-05-12 08:00:00', '2024-05-17 08:00:00', 'Active', 2100.00),
    -> (34, 34, '2024-05-13 10:00:00', '2024-05-18 10:00:00', 'Closed', 2400.00),
    -> (35, 35, '2024-05-14 12:00:00', '2024-05-19 12:00:00', 'Pending', 3000.00),
    -> (36, 36, '2024-05-15 14:00:00', '2024-05-20 14:00:00', 'Active', 1800.00),
    -> (37, 37, '2024-05-16 16:00:00', '2024-05-21 16:00:00', 'Active', 1500.00),
    -> (38, 38, '2024-05-17 18:00:00', '2024-05-22 18:00:00', 'Closed', 800.00),
    -> (39, 39, '2024-05-18 20:00:00', '2024-05-23 20:00:00', 'Active', 2200.00),
    -> (40, 40, '2024-05-19 22:00:00', '2024-05-24 22:00:00', 'Active', 2500.00),
    -> (41, 41, '2024-05-20 08:00:00', '2024-05-25 08:00:00', 'Closed', 1200.00),
    -> (42, 42, '2024-05-21 10:00:00', '2024-05-26 10:00:00', 'Active', 1300.00),
    -> (43, 43, '2024-05-22 12:00:00', '2024-05-27 12:00:00', 'Active', 800.00),
    -> (44, 44, '2024-05-23 14:00:00', '2024-05-28 14:00:00', 'Closed', 1800.00),
    -> (45, 45, '2024-05-24 16:00:00', '2024-05-29 16:00:00', 'Active', 2000.00),
    -> (46, 46, '2024-05-25 18:00:00', '2024-05-30 18:00:00', 'Active', 900.00),
    -> (47, 47, '2024-05-26 20:00:00', '2024-05-31 20:00:00', 'Closed', 1500.00),
    -> (48, 48, '2024-05-27 22:00:00', '2024-06-01 22:00:00', 'Active', 300.00),
    -> (49, 49, '2024-05-28 08:00:00', '2024-06-02 08:00:00', 'Active', 1200.00),
    -> (50, 50, '2024-05-29 10:00:00', '2024-06-03 10:00:00', 'Pending', 800.00);
Query OK, 50 rows affected (0.006 sec)
Records: 50  Duplicates: 0  Warnings: 0
```

6. Transactions Table

```
MariaDB [dbms]> INSERT INTO Transactions (Transaction_ID, Buyer_ID, Seller_ID, Item_ID, Transaction_Amount, Transaction_Time, Payment_Method, Transaction_Status) VALUES
    -> (1, 1, 11, 1, 900.00, '2024-05-15 08:30:00', 'Cash', 'Completed'),
    -> (2, 2, 12, 2, 450.00, '2024-05-16 10:30:00', 'Credit Card', 'Completed'),
    -> (3, 3, 13, 3, 1600.00, '2024-05-17 12:30:00', 'UPI', 'Completed'),
    -> (4, 4, 14, 4, 25000.00, '2024-05-18 14:30:00', 'Cash', 'Completed'),
    -> (5, 5, 15, 5, 800.00, '2024-05-19 16:30:00', 'Credit Card', 'Completed'),
    -> (6, 6, 16, 6, 1200.00, '2024-05-20 18:30:00', 'Debit Card', 'Completed'),
    -> (7, 7, 17, 7, 7000.00, '2024-05-21 20:30:00', 'Net Banking', 'Completed'),
    -> (8, 8, 18, 8, 3500.00, '2024-05-22 22:30:00', 'Cash', 'Completed'),
    -> (9, 9, 19, 9, 1800.00, '2024-05-23 08:30:00', 'UPI', 'Completed'),
    -> (10, 10, 20, 10, 1900.00, '2024-05-24 10:30:00', 'Credit Card', 'Completed'),
    -> (11, 11, 21, 11, 800.00, '2024-05-25 12:30:00', 'Cash', 'Completed'),
    -> (12, 12, 22, 12, 3000.00, '2024-05-26 14:30:00', 'Debit Card', 'Completed'),
    -> (13, 13, 23, 13, 2500.00, '2024-05-27 16:30:00', 'Net Banking', 'Completed'),
    -> (14, 14, 24, 14, 900.00, '2024-05-28 18:30:00', 'Cash', 'Completed'),
    -> (15, 15, 25, 15, 1200.00, '2024-05-29 20:30:00', 'UPI', 'Completed'),
    -> (16, 16, 26, 16, 1500.00, '2024-05-30 22:30:00', 'Credit Card', 'Completed'),
    -> (17, 17, 27, 17, 8000.00, '2024-05-31 08:30:00', 'Debit Card', 'Completed'),
    -> (18, 18, 28, 18, 2200.00, '2024-06-01 10:30:00', 'Net Banking', 'Completed'),
    -> (19, 19, 29, 19, 1200.00, '2024-06-02 12:30:00', 'Cash', 'Completed'),
    -> (20, 20, 30, 20, 1700.00, '2024-06-03 14:30:00', 'Credit Card', 'Completed'),
    -> (21, 21, 31, 21, 800.00, '2024-06-04 16:30:00', 'UPI', 'Completed'),
    -> (22, 22, 32, 22, 3000.00, '2024-06-05 18:30:00', 'Debit Card', 'Completed'),
    -> (23, 23, 33, 23, 2500.00, '2024-06-06 20:30:00', 'Net Banking', 'Completed'),
    -> (24, 24, 34, 24, 1800.00, '2024-06-07 22:30:00', 'Cash', 'Completed'),
    -> (25, 25, 35, 25, 4000.00, '2024-06-08 08:30:00', 'Credit Card', 'Completed'),
    -> (26, 26, 36, 26, 3500.00, '2024-06-09 10:30:00', 'UPI', 'Completed'),
    -> (27, 27, 37, 27, 4200.00, '2024-06-10 12:30:00', 'Debit Card', 'Completed'),
    -> (28, 28, 38, 28, 1900.00, '2024-06-11 14:30:00', 'Net Banking', 'Completed'),
    -> (29, 29, 39, 29, 2300.00, '2024-06-12 16:30:00', 'Cash', 'Completed'),
    -> (30, 30, 40, 30, 1400.00, '2024-06-13 18:30:00', 'Credit Card', 'Completed'),
    -> (31, 31, 41, 31, 1600.00, '2024-06-14 20:30:00', 'UPI', 'Completed'),
    -> (32, 32, 42, 32, 800.00, '2024-06-15 22:30:00', 'Debit Card', 'Completed'),
    -> (33, 33, 43, 33, 2100.00, '2024-06-16 08:30:00', 'Net Banking', 'Completed'),
    -> (34, 34, 44, 34, 2400.00, '2024-06-17 10:30:00', 'Cash', 'Completed'),
    -> (35, 35, 45, 35, 3000.00, '2024-06-18 12:30:00', 'Credit Card', 'Completed'),
    -> (36, 36, 46, 36, 1800.00, '2024-06-19 14:30:00', 'UPI', 'Completed'),
    -> (37, 37, 47, 37, 1500.00, '2024-06-20 16:30:00', 'Debit Card', 'Completed'),
    -> (38, 38, 48, 38, 800.00, '2024-06-21 18:30:00', 'Net Banking', 'Completed'),
    -> (39, 39, 49, 39, 2200.00, '2024-06-22 20:30:00', 'Cash', 'Completed'),
    -> (40, 40, 50, 40, 2500.00, '2024-06-23 22:30:00', 'Credit Card', 'Completed'),
    -> (41, 41, 1, 41, 1200.00, '2024-06-24 08:30:00', 'UPI', 'Completed'),
    -> (42, 42, 2, 42, 1300.00, '2024-06-25 10:30:00', 'Debit Card', 'Completed'),
    -> (43, 43, 3, 43, 800.00, '2024-06-26 12:30:00', 'Net Banking', 'Completed'),
    -> (44, 44, 4, 44, 1800.00, '2024-06-27 14:30:00', 'Cash', 'Completed'),
    -> (45, 45, 5, 45, 2000.00, '2024-06-28 16:30:00', 'Credit Card', 'Completed'),
    -> (46, 46, 6, 46, 900.00, '2024-06-29 18:30:00', 'UPI', 'Completed'),
    -> (47, 47, 7, 47, 1500.00, '2024-06-30 20:30:00', 'Debit Card', 'Completed'),
    -> (48, 48, 8, 48, 300.00, '2024-07-01 22:30:00', 'Net Banking', 'Completed'),
    -> (49, 49, 9, 49, 1200.00, '2024-07-02 08:30:00', 'Cash', 'Completed'),
    -> (50, 50, 10, 50, 800.00, '2024-07-03 10:30:00', 'Credit Card', 'Completed');
Query OK, 50 rows affected (0.007 sec)
Records: 50  Duplicates: 0  Warnings: 0
```

Complex SQL queries utilizing aggregate functions like AVG, GROUP BY, and sorting to manipulate data for the ER Data Model:

SQL-DML should explore Complex queries, Aggregate functions like Avg, group, sort, etc to manipulate data for your ER Data Model

1. Calculate the average bid amount for each item:

SELECT Item_ID, AVG(Bid_Amount) AS Avg_Bid_Amount

FROM Bids

GROUP BY Item_ID;

```
MariaDB [dbms]> SELECT Item_ID, AVG(Bid_Amount) AS Avg_Bid_Amount
    -> FROM Bids
    -> GROUP BY Item_ID;
+---------+-----------------+
| Item_ID | Avg_Bid_Amount  |
+---------+-----------------+
|       1 |     675.000000 |
|       2 |     425.000000 |
|       3 |    1475.000000 |
|       4 |   25750.000000 |
|       5 |     975.000000 |
|       6 |    1550.000000 |
|       7 |   46500.000000 |
|       8 |    3200.000000 |
|       9 |   61500.000000 |
|      10 |    1850.000000 |
|      11 |     925.000000 |
|      12 |    4150.000000 |
|      13 |   56000.000000 |
|      14 |    8200.000000 |
|      15 |    7450.000000 |
|      16 |   11250.000000 |
|      17 |   14500.000000 |
|      18 |    2900.000000 |
|      19 |    5350.000000 |
|      20 |   36000.000000 |
|      21 |    2050.000000 |
|      22 |   18500.000000 |
|      23 |    3400.000000 |
|      24 |    1750.000000 |
|      25 |   40000.000000 |
+---------+-----------------+
25 rows in set (0.001 sec)
```

2. List the total account balance for buyers and sellers:

SELECT 'Buyer' AS User_Type, SUM(Account_Balance) AS Total_Buyer_Balance

FROM Buyers

UNION

SELECT 'Seller' AS User_Type, SUM(Account_Balance) AS Total_Seller_Balance

FROM Sellers;

```
MariaDB [dbms]> SELECT 'Buyer' AS User_Type, SUM(Account_Balance) AS Total_Buyer_Balance
    -> FROM Buyers
    -> UNION
    -> SELECT 'Seller' AS User_Type, SUM(Account_Balance) AS Total_Seller_Balance
    -> FROM Sellers;
+-----------+----------------------+
| User_Type | Total_Buyer_Balance |
+-----------+----------------------+
| Buyer     |           259600.00 |
| Seller    |           494200.00 |
+-----------+----------------------+
2 rows in set (0.004 sec)
```

3. Find the highest bid amount for each item:

SELECT Item_ID, MAX(Bid_Amount) AS Highest_Bid_Amount

FROM Bids

GROUP BY Item_ID;

```
MariaDB [dbms]> SELECT Item_ID, MAX(Bid_Amount) AS Highest_Bid_Amount
    -> FROM Bids
    -> GROUP BY Item_ID;
+---------+----------------------+
| Item_ID | Highest_Bid_Amount |
+---------+----------------------+
|       1 |              700.00 |
|       2 |              450.00 |
|       3 |             1500.00 |
|       4 |            26000.00 |
|       5 |             1000.00 |
|       6 |             1600.00 |
|       7 |            47000.00 |
|       8 |             3300.00 |
|       9 |            62000.00 |
|      10 |             1900.00 |
|      11 |              950.00 |
|      12 |             4300.00 |
|      13 |            57000.00 |
|      14 |             8400.00 |
|      15 |             7700.00 |
|      16 |            11500.00 |
|      17 |            15000.00 |
|      18 |             3000.00 |
|      19 |             5500.00 |
|      20 |            37000.00 |
|      21 |             2100.00 |
|      22 |            19000.00 |
|      23 |             3500.00 |
|      24 |             1800.00 |
|      25 |            41000.00 |
+---------+----------------------+
25 rows in set (0.002 sec)
```

4. List items with their respective number of bids, sorted by the number of bids in descending order:

```
SELECT i.Item_ID, i.Item_Name, COUNT(b.Bid_ID) AS Num_Bids
FROM Items i
LEFT JOIN Bids b ON i.Item_ID = b.Item_ID
GROUP BY i.Item_ID, i.Item_Name
ORDER BY Num_Bids DESC;
```

```
MariaDB [dbms]> SELECT i.Item_ID, i.Item_Name, COUNT(b.Bid_ID) AS Num_Bids
    -> FROM Items i
    -> LEFT JOIN Bids b ON i.Item_ID = b.Item_ID
    -> GROUP BY i.Item_ID, i.Item_Name
    -> ORDER BY Num_Bids DESC;
+---------+----------------------+----------+
| Item_ID | Item_Name            | Num_Bids |
+---------+----------------------+----------+
|       4 | Smart LED TV         |        2 |
|      20 | Smart Watch          |        2 |
|       5 | Handloom Rug         |        2 |
|      21 | Backpack             |        2 |
|       6 | Kitchen Mixer        |        2 |
|      22 | Sofa Set             |        2 |
|       7 | Gaming Console       |        2 |
|      23 | Smart Speaker        |        2 |
|       8 | Leather Jacket       |        2 |
|      24 | Wall Art             |        2 |
|       9 | DSLR Camera          |        2 |
|      25 | Sunglasses           |        2 |
|      10 | Antique Clock        |        2 |
|      11 | Tea Set              |        2 |
|      12 | Fitness Tracker      |        2 |
|      13 | Smartphone           |        2 |
|      14 | Office Chair         |        2 |
|      15 | Cocktail Dress       |        2 |
|      16 | Air Purifier         |        2 |
|       1 | Handcrafted Pot      |        2 |
|      17 | Bicycle              |        2 |
|       2 | Organic Spices       |        2 |
|      18 | Bookshelf            |        2 |
|       3 | Designer Saree       |        2 |
|      19 | Sneakers             |        2 |
|      36 | DSLR Camera          |        0 |
|      37 | Microwave Oven       |        0 |
|      38 | Gaming Mouse         |        0 |
|      39 | Travel Backpack      |        0 |
|      40 | Bluetooth Speaker    |        0 |
|      41 | Study Desk           |        0 |
|      26 | Laptop               |        0 |
|      42 | Running Shoes        |        0 |
|      27 | Barbecue Grill       |        0 |
|      43 | Electric Kettle      |        0 |
|      28 | Yoga Mat             |        0 |
|      44 | Study Lamp           |        0 |
|      29 | Wall Clock           |        0 |
|      45 | Bluetooth Earphones  |        0 |
|      30 | Wireless Headphones  |        0 |
|      46 | Wall Mirror          |        0 |
|      31 | Dinner Set           |        0 |
|      47 | Yoga Mat             |        0 |
|      32 | Running Shoes        |        0 |
|      48 | Blender              |        0 |
|      33 | Coffee Maker         |        0 |
|      49 | Laptop Backpack      |        0 |
|      34 | Backpack             |        0 |
|      50 | Wall Art             |        0 |
|      35 | Pendant Light        |        0 |
+---------+----------------------+----------+
50 rows in set (0.004 sec)
```

5. Calculate the total transaction amount for each seller:

```
SELECT t.Seller_ID, s.Username AS Seller_Username,
SUM(t.Transaction_Amount) AS Total_Transaction_Amount
FROM Transactions t
JOIN Sellers s ON t.Seller_ID = s.Seller_ID
GROUP BY t.Seller_ID, s.Username;
```

```
MariaDB [dbms]> SELECT t.Seller_ID, s.Username AS Seller_Username, SUM(t.Transaction_Amount) AS Total_Transaction_Amount
    -> FROM Transactions t
    -> JOIN Sellers s ON t.Seller_ID = s.Seller_ID
    -> GROUP BY t.Seller_ID, s.Username;
+-----------+-----------------+---------------------------+
| Seller_ID | Seller_Username | Total_Transaction_Amount  |
+-----------+-----------------+---------------------------+
|         1 | indian_crafts   |                   1200.00 |
|         2 | organic_farm    |                   1300.00 |
|         3 | fashion_hub     |                    800.00 |
|         4 | spices_galore   |                   1800.00 |
|         5 | tech_store      |                   2000.00 |
|         6 | aroma_world     |                    900.00 |
|         7 | greenery_decor  |                   1500.00 |
|         8 | desi_dresses    |                    300.00 |
|         9 | bookworm_books  |                   1200.00 |
|        10 | home_appliances |                    800.00 |
|        11 | crafty_creations|                    900.00 |
|        12 | wellness_corner |                    450.00 |
|        13 | organic_beauty  |                   1600.00 |
|        14 | garden_oasis    |                  25000.00 |
|        15 | ethnic_attire   |                    800.00 |
|        16 | gadget_galaxy   |                   1200.00 |
|        17 | fitness_freaks  |                   7000.00 |
|        18 | kitchen_king    |                   3500.00 |
|        19 | home_decor      |                   1800.00 |
|        20 | beauty_boutique |                   1900.00 |
|        21 | grocery_galore  |                    800.00 |
|        22 | gadget_hub      |                   3000.00 |
|        23 | fashionista     |                   2500.00 |
|        24 | handmade_hub    |                    900.00 |
|        25 | pet_paradise    |                   1200.00 |
|        26 | sports_corner   |                   1500.00 |
|        27 | stationary_stop |                   8000.00 |
|        28 | fashion_fusion  |                   2200.00 |
|        29 | home_furnishings|                   1200.00 |
|        30 | organic_eats    |                   1700.00 |
|        31 | smart_gadgets   |                    800.00 |
|        32 | trendy_tees     |                   3000.00 |
|        33 | art_affair      |                   2500.00 |
|        34 | health_hub      |                   1800.00 |
|        35 | beauty_basics   |                   4000.00 |
|        36 | kitchen_korner  |                   3500.00 |
|        37 | fashion_frenzy  |                   4200.00 |
|        38 | pet_palace      |                   1900.00 |
|        39 | tech_trends     |                   2300.00 |
|        40 | eco_store       |                   1400.00 |
|        41 | fitness_fiesta  |                   1600.00 |
|        42 | gourmet_corner  |                    800.00 |
|        43 | gadget_guru     |                   2100.00 |
|        44 | home_haven      |                   2400.00 |
|        45 | fashion_forward |                   3000.00 |
|        46 | garden_glory    |                   1800.00 |
|        47 | art_arena       |                   1500.00 |
|        48 | health_harmony  |                    800.00 |
|        49 | fashion_fabrics |                   2200.00 |
|        50 | kitchen_kraft   |                   2500.00 |
+-----------+-----------------+---------------------------+
50 rows in set (0.001 sec)
```

6. Find the average starting price for items in each category:

```
SELECT Category, AVG(Starting_Price) AS Avg_Starting_Price
FROM Items
GROUP BY Category;
```

```
MariaDB [dbms]> SELECT Category, AVG(Starting_Price) AS Avg_Starting_Price
    -> FROM Items
    -> GROUP BY Category;
+--------------------+---------------------+
| Category           | Avg_Starting_Price  |
+--------------------+---------------------+
| Appliances         |         7285.714286 |
| Collectibles       |         1800.000000 |
| Electronics        |        32769.230769 |
| Fashion            |         4957.142857 |
| Fitness            |         5000.000000 |
| Furniture          |         8375.000000 |
| Grocery            |          350.000000 |
| Home & Decor       |         2200.000000 |
| Sports & Outdoors  |        12000.000000 |
| Travel & Outdoor   |         4125.000000 |
+--------------------+---------------------+
10 rows in set (0.000 sec)
```

7. List the top 5 buyers with the highest account balances:

SELECT Username, Account_Balance

FROM Buyers

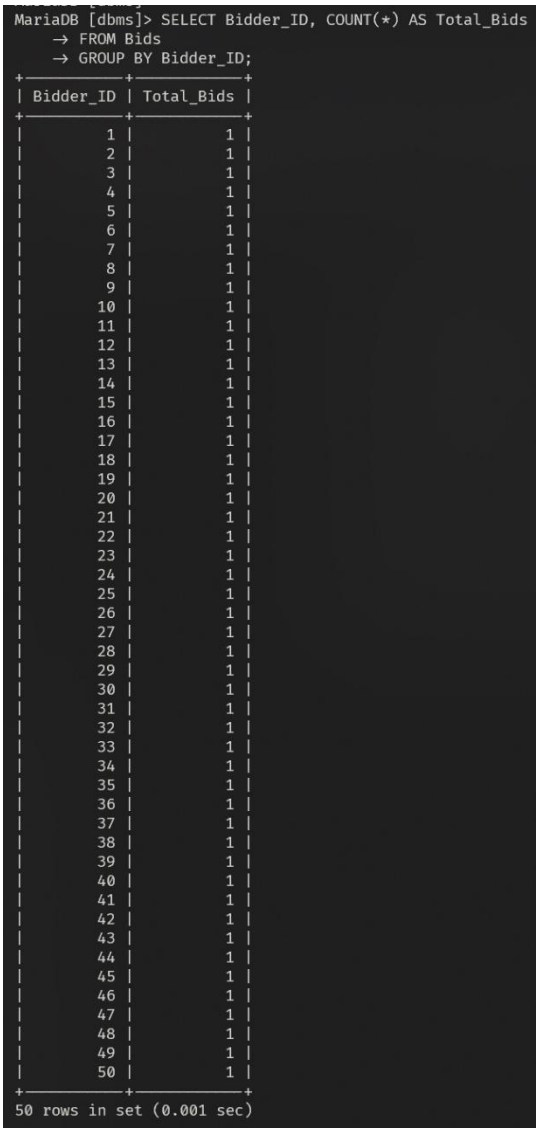
ORDER BY Account_Balance DESC

LIMIT 5;

```
MariaDB [dbms]> SELECT Username, Account_Balance
    -> FROM Buyers
    -> ORDER BY Account_Balance DESC
    -> LIMIT 5;
+--------------+-----------------+
| Username     | Account_Balance |
+--------------+-----------------+
| raj_sharma   |         7000.00 |
| arnav.kapoor |         6800.00 |
| aditya31     |         6500.00 |
| shreya.m     |         6200.00 |
| harshit123   |         6200.00 |
+--------------+-----------------+
5 rows in set (0.000 sec)
```

8. Calculate the total number of bids made by each buyer:

```
SELECT Bidder_ID, COUNT(*) AS Total_Bids
FROM Bids
GROUP BY Bidder_ID;
```



```
MariaDB [dbms]> SELECT Bidder_ID, COUNT(*) AS Total_Bids
    -> FROM Bids
    -> GROUP BY Bidder_ID;
+-----------+-------------+
| Bidder_ID | Total_Bids |
+-----------+-------------+
|         1 |           1 |
|         2 |           1 |
|         3 |           1 |
|         4 |           1 |
|         5 |           1 |
|         6 |           1 |
|         7 |           1 |
|         8 |           1 |
|         9 |           1 |
|        10 |           1 |
|        11 |           1 |
|        12 |           1 |
|        13 |           1 |
|        14 |           1 |
|        15 |           1 |
|        16 |           1 |
|        17 |           1 |
|        18 |           1 |
|        19 |           1 |
|        20 |           1 |
|        21 |           1 |
|        22 |           1 |
|        23 |           1 |
|        24 |           1 |
|        25 |           1 |
|        26 |           1 |
|        27 |           1 |
|        28 |           1 |
|        29 |           1 |
|        30 |           1 |
|        31 |           1 |
|        32 |           1 |
|        33 |           1 |
|        34 |           1 |
|        35 |           1 |
|        36 |           1 |
|        37 |           1 |
|        38 |           1 |
|        39 |           1 |
|        40 |           1 |
|        41 |           1 |
|        42 |           1 |
|        43 |           1 |
|        44 |           1 |
|        45 |           1 |
|        46 |           1 |
|        47 |           1 |
|        48 |           1 |
|        49 |           1 |
|        50 |           1 |
+-----------+-------------+
50 rows in set (0.001 sec)
```

9. Find the average bid increment for each item:

```
SELECT Item_ID, AVG(Bid_Increment) AS Avg_Bid_Increment
FROM Bids
GROUP BY Item_ID;
```

```
MariaDB [dbms]> SELECT Item_ID, AVG(Bid_Increment) AS Avg_Bid_Increment
    -> FROM Bids
    -> GROUP BY Item_ID;
+---------+--------------------+
| Item_ID | Avg_Bid_Increment |
+---------+--------------------+
|       1 |         100.000000 |
|       2 |          50.000000 |
|       3 |         100.000000 |
|       4 |        1000.000000 |
|       5 |         100.000000 |
|       6 |         200.000000 |
|       7 |        2000.000000 |
|       8 |         300.000000 |
|       9 |        2000.000000 |
|      10 |         100.000000 |
|      11 |          50.000000 |
|      12 |         300.000000 |
|      13 |        2000.000000 |
|      14 |         400.000000 |
|      15 |         500.000000 |
|      16 |         500.000000 |
|      17 |        1000.000000 |
|      18 |         200.000000 |
|      19 |         300.000000 |
|      20 |        2000.000000 |
|      21 |         100.000000 |
|      22 |        1000.000000 |
|      23 |         200.000000 |
|      24 |         100.000000 |
|      25 |        2000.000000 |
+---------+--------------------+
25 rows in set (0.000 sec)
```

10. List sellers who have not made any transactions yet:

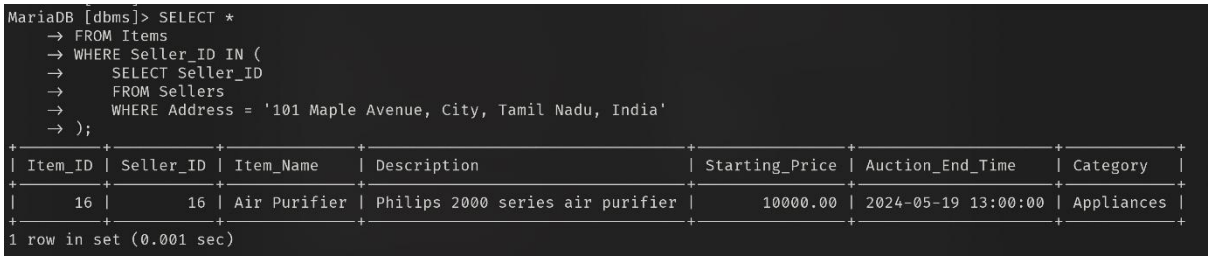
```
SELECT s.Seller_ID, s.Username
FROM Sellers s
LEFT JOIN Transactions t ON s.Seller_ID = t.Seller_ID
WHERE t.Seller_ID IS NULL;
```

```
MariaDB [dbms]> SELECT s.Seller_ID, s.Username
    -> FROM Sellers s
    -> LEFT JOIN Transactions t ON s.Seller_ID = t.Seller_ID
    -> WHERE t.Seller_ID IS NULL;
Empty set (0.000 sec)
```

SQL-DML should explore Nested queries, Join Queries, along with the complex queries, Aggregate functions etc to manipulate data for the ER Data Model

1. **Nested Query:** Get all items listed by sellers with a specific address

```
SELECT *
FROM Items
WHERE Seller_ID IN (
   SELECT Seller_ID
   FROM Sellers
   WHERE Address = '101 Maple Avenue, City, Tamil Nadu, India'
);
```



```
MariaDB [dbms]> SELECT *
    -> FROM Items
    -> WHERE Seller_ID IN (
    ->     SELECT Seller_ID
    ->     FROM Sellers
    ->     WHERE Address = '101 Maple Avenue, City, Tamil Nadu, India'
    -> );
+---------+-----------+-------------+-------------------------------+----------------+---------------------+------------+
| Item_ID | Seller_ID | Item_Name   | Description                   | Starting_Price | Auction_End_Time    | Category   |
+---------+-----------+-------------+-------------------------------+----------------+---------------------+------------+
|      16 |        16 | Air Purifier | Philips 2000 series air purifier |       10000.00 | 2024-05-19 13:00:00 | Appliances |
+---------+-----------+-------------+-------------------------------+----------------+---------------------+------------+
1 row in set (0.001 sec)
```

2. **Join Query:** Get the details of all bids along with bidder information

```
SELECT B.*, Bu.Username AS Bidder_Username, Bu.Email AS Bidder_Email
FROM Bids B
INNER JOIN Buyers Bu ON B.Bidder_ID = Bu.Buyer_ID;
```

```
MariaDB [dbms]> SELECT B.*, Bu.Username AS Bidder_Username, Bu.Email AS Bidder_Email
    -> FROM Bids B
    -> INNER JOIN Buyers Bu ON B.Bidder_ID = Bu.Buyer_ID;
+--------+-----------+---------+------------+---------------------+------------+---------------+-----------------+-----------------------+
| Bid_ID | Bidder_ID | Item_ID | Bid_Amount | Bid_Time            | Bid_Status | Bid_Increment | Bidder_Username | Bidder_Email          |
+--------+-----------+---------+------------+---------------------+------------+---------------+-----------------+-----------------------+
|      1 |         1 |       1 |     700.00 | 2024-05-09 14:30:00 | Winning    |        100.00 | ananya_gupta    | ananya.gupta@gmail.com |
|      2 |         2 |       1 |     650.00 | 2024-05-09 15:45:00 | Outbid     |        100.00 | raj_sharma      | raj.sharma@yahoo.com  |
|      3 |         3 |       2 |     400.00 | 2024-05-09 16:20:00 | Winning    |         50.00 | rahul.kumar     | rahul.kumar@outlook.com |
|      4 |         4 |       2 |     450.00 | 2024-05-09 17:00:00 | Outbid     |         50.00 | sangeeta25      | sangeeta25@gmail.com  |
|      5 |         5 |       3 |    1500.00 | 2024-05-09 18:30:00 | Won        |        100.00 | vikram_singh    | vikram.singh@yahoo.com |
|      6 |         6 |       3 |    1450.00 | 2024-05-09 19:00:00 | Lost       |        100.00 | sonali_89       | sonali_89@outlook.com |
|      7 |         7 |       4 |   26000.00 | 2024-05-09 20:00:00 | Winning    |       1000.00 | amit_patil      | amit.patil@gmail.com  |
|      8 |         8 |       4 |   25500.00 | 2024-05-09 21:00:00 | Outbid     |       1000.00 | shreya.m        | shreya.m@yahoo.com    |
|      9 |         9 |       5 |    1000.00 | 2024-05-09 22:00:00 | Winning    |        100.00 | nikhil_2000     | nikhil_2000@outlook.com |
|     10 |        10 |       5 |     950.00 | 2024-05-09 23:00:00 | Outbid     |        100.00 | priya12         | priya12@gmail.com     |
|     11 |        11 |       6 |    1600.00 | 2024-05-10 09:30:00 | Winning    |        200.00 | arnav.kapoor    | arnav.kapoor@yahoo.com |
|     12 |        12 |       6 |    1500.00 | 2024-05-10 10:00:00 | Outbid     |        200.00 | megha.sen       | megha.sen@outlook.com |
|     13 |        13 |       7 |   47000.00 | 2024-05-10 11:30:00 | Winning    |       2000.00 | vishal32        | vishal32@gmail.com    |
|     14 |        14 |       7 |   46000.00 | 2024-05-10 12:00:00 | Outbid     |       2000.00 | anjali_11       | anjali_11@yahoo.com   |
|     15 |        15 |       8 |    3300.00 | 2024-05-10 13:30:00 | Winning    |        300.00 | ritika.m        | ritika.m@outlook.com  |
|     16 |        16 |       8 |    3100.00 | 2024-05-10 14:00:00 | Outbid     |        300.00 | harshit123      | harshit123@gmail.com  |
|     17 |        17 |       9 |   62000.00 | 2024-05-10 15:30:00 | Winning    |       2000.00 | sneha_04        | sneha_04@yahoo.com    |
|     18 |        18 |       9 |   61000.00 | 2024-05-10 16:00:00 | Outbid     |       2000.00 | amar_kumar      | amar_kumar@outlook.com |
|     19 |        19 |      10 |    1900.00 | 2024-05-10 17:30:00 | Winning    |        100.00 | divya_09        | divya_09@gmail.com    |
|     20 |        20 |      10 |    1800.00 | 2024-05-10 18:00:00 | Outbid     |        100.00 | rohit.b         | rohit.b@yahoo.com     |
|     21 |        21 |      11 |     950.00 | 2024-05-10 19:30:00 | Winning    |         50.00 | anu_1990        | anu_1990@outlook.com  |
|     22 |        22 |      11 |     900.00 | 2024-05-10 20:00:00 | Outbid     |         50.00 | aditya31        | aditya31@gmail.com    |
|     23 |        23 |      12 |    4300.00 | 2024-05-10 21:30:00 | Winning    |        300.00 | priyanka.m      | priyanka.m@yahoo.com  |
|     24 |        24 |      12 |    4000.00 | 2024-05-10 22:00:00 | Outbid     |        300.00 | rakesh.singh    | rakesh.singh@outlook.com |
|     25 |        25 |      13 |   57000.00 | 2024-05-10 23:30:00 | Winning    |       2000.00 | shivani_g       | shivani_g@gmail.com   |
|     26 |        26 |      13 |   55000.00 | 2024-05-11 00:00:00 | Outbid     |       2000.00 | ankit_08        | ankit_08@yahoo.com    |
|     27 |        27 |      14 |    8400.00 | 2024-05-11 09:30:00 | Winning    |        400.00 | saurabh.k       | saurabh.k@outlook.com |
|     28 |        28 |      14 |    8000.00 | 2024-05-11 10:00:00 | Outbid     |        400.00 | nisha_45        | nisha_45@gmail.com    |
|     29 |        29 |      15 |    7700.00 | 2024-05-11 11:30:00 | Winning    |        500.00 | deepak_1987     | deepak_1987@yahoo.com |
|     30 |        30 |      15 |    7200.00 | 2024-05-11 12:00:00 | Outbid     |        500.00 | mohit.kumar     | mohit.kumar@outlook.com |
|     31 |        31 |      16 |   11500.00 | 2024-05-11 13:30:00 | Winning    |        500.00 | pooja_18        | pooja_18@gmail.com    |
|     32 |        32 |      16 |   11000.00 | 2024-05-11 14:00:00 | Outbid     |        500.00 | vivek.p         | vivek.p@yahoo.com     |
|     33 |        33 |      17 |   15000.00 | 2024-05-11 15:30:00 | Winning    |       1000.00 | tanvi_09        | tanvi_09@outlook.com  |
|     34 |        34 |      17 |   14000.00 | 2024-05-11 16:00:00 | Outbid     |       1000.00 | rohan_gupta     | rohan_gupta@gmail.com |
|     35 |        35 |      18 |    3000.00 | 2024-05-11 17:30:00 | Winning    |        200.00 | shikha_34       | shikha_34@yahoo.com   |
|     36 |        36 |      18 |    2800.00 | 2024-05-11 18:00:00 | Outbid     |        200.00 | devansh.s       | devansh.s@outlook.com |
|     37 |        37 |      19 |    5500.00 | 2024-05-11 19:30:00 | Winning    |        300.00 | priyanka.k      | priyanka.k@gmail.com  |
|     38 |        38 |      19 |    5200.00 | 2024-05-11 20:00:00 | Outbid     |        300.00 | akash_joshi     | akash_joshi@yahoo.com |
|     39 |        39 |      20 |   37000.00 | 2024-05-11 21:30:00 | Winning    |       2000.00 | neha12          | neha12@outlook.com    |
|     40 |        40 |      20 |   35000.00 | 2024-05-11 22:00:00 | Outbid     |       2000.00 | amit_23         | amit_23@gmail.com     |
|     41 |        41 |      21 |    2100.00 | 2024-05-11 23:30:00 | Winning    |        100.00 | riya.45         | riya.45@yahoo.com     |
|     42 |        42 |      21 |    2000.00 | 2024-05-12 00:00:00 | Outbid     |        100.00 | harshita.s      | harshita.s@outlook.com |
|     43 |        43 |      22 |   19000.00 | 2024-05-12 09:30:00 | Winning    |       1000.00 | shubham_07      | shubham_07@gmail.com  |
|     44 |        44 |      22 |   18000.00 | 2024-05-12 10:00:00 | Outbid     |       1000.00 | kritika.k       | kritika.k@yahoo.com   |
|     45 |        45 |      23 |    3500.00 | 2024-05-12 11:30:00 | Winning    |        200.00 | rahul_1988      | rahul_1988@outlook.com |
|     46 |        46 |      23 |    3300.00 | 2024-05-12 12:00:00 | Outbid     |        200.00 | meena.sharma    | meena.sharma@gmail.com |
|     47 |        47 |      24 |    1800.00 | 2024-05-12 13:30:00 | Winning    |        100.00 | shivamg         | shivamg@yahoo.com     |
|     48 |        48 |      24 |    1700.00 | 2024-05-12 14:00:00 | Outbid     |        100.00 | kavya_123       | kavya_123@outlook.com |
|     49 |        49 |      25 |   41000.00 | 2024-05-12 15:30:00 | Winning    |       2000.00 | anurag_86       | anurag_86@gmail.com   |
|     50 |        50 |      25 |   39000.00 | 2024-05-12 16:00:00 | Outbid     |       2000.00 | jyoti.11        | jyoti.11@yahoo.com    |
+--------+-----------+---------+------------+---------------------+------------+---------------+-----------------+-----------------------+
50 rows in set (0.000 sec)
```

3. **Complex Query:** Get the total transaction amount for each seller

```
SELECT
    S.Seller_ID,
    S.Username AS Seller_Username,
    SUM(T.Transaction_Amount) AS Total_Transaction_Amount
FROM Sellers S
LEFT JOIN Items I ON S.Seller_ID = I.Seller_ID
LEFT JOIN Transactions T ON I.Item_ID = T.Item_ID
GROUP BY S.Seller_ID, S.Username;
```

```
MariaDB [dbms]> SELECT
    ->     S.Seller_ID,
    ->     S.Username AS Seller_Username,
    ->     SUM(T.Transaction_Amount) AS Total_Transaction_Amount
    -> FROM Sellers S
    -> LEFT JOIN Items I ON S.Seller_ID = I.Seller_ID
    -> LEFT JOIN Transactions T ON I.Item_ID = T.Item_ID
    -> GROUP BY S.Seller_ID, S.Username;
+-----------+-----------------+----------------------------+
| Seller_ID | Seller_Username | Total_Transaction_Amount |
+-----------+-----------------+----------------------------+
|         1 | indian_crafts   |                     900.00 |
|         2 | organic_farm    |                     450.00 |
|         3 | fashion_hub     |                    1600.00 |
|         4 | spices_galore   |                   25000.00 |
|         5 | tech_store      |                     800.00 |
|         6 | aroma_world     |                    1200.00 |
|         7 | greenery_decor  |                    7000.00 |
|         8 | desi_dresses    |                    3500.00 |
|         9 | bookworm_books  |                    1800.00 |
|        10 | home_appliances |                    1900.00 |
|        11 | crafty_creations|                     800.00 |
|        12 | wellness_corner |                    3000.00 |
|        13 | organic_beauty  |                    2500.00 |
|        14 | garden_oasis    |                     900.00 |
|        15 | ethnic_attire   |                    1200.00 |
|        16 | gadget_galaxy   |                    1500.00 |
|        17 | fitness_freaks  |                    8000.00 |
|        18 | kitchen_king    |                    2200.00 |
|        19 | home_decor      |                    1200.00 |
|        20 | beauty_boutique |                    1700.00 |
|        21 | grocery_galore  |                     800.00 |
|        22 | gadget_hub      |                    3000.00 |
|        23 | fashionista     |                    2500.00 |
|        24 | handmade_hub    |                    1800.00 |
|        25 | pet_paradise    |                    4000.00 |
|        26 | sports_corner   |                    3500.00 |
|        27 | stationary_stop |                    4200.00 |
|        28 | fashion_fusion  |                    1900.00 |
|        29 | home_furnishings|                    2300.00 |
|        30 | organic_eats    |                    1400.00 |
|        31 | smart_gadgets   |                    1600.00 |
|        32 | trendy_tees     |                     800.00 |
|        33 | art_affair      |                    2100.00 |
|        34 | health_hub      |                    2400.00 |
|        35 | beauty_basics   |                    3000.00 |
|        36 | kitchen_korner  |                    1800.00 |
|        37 | fashion_frenzy  |                    1500.00 |
|        38 | pet_palace      |                     800.00 |
|        39 | tech_trends     |                    2200.00 |
|        40 | eco_store       |                    2500.00 |
|        41 | fitness_fiesta  |                    1200.00 |
|        42 | gourmet_corner  |                    1300.00 |
|        43 | gadget_guru     |                     800.00 |
|        44 | home_haven      |                    1800.00 |
|        45 | fashion_forward |                    2000.00 |
|        46 | garden_glory    |                     900.00 |
|        47 | art_arena       |                    1500.00 |
|        48 | health_harmony  |                     300.00 |
|        49 | fashion_fabrics |                    1200.00 |
|        50 | kitchen_kraft   |                     800.00 |
+-----------+-----------------+----------------------------+
50 rows in set (0.004 sec)
```

4. **Aggregate Function:** Get the average account balance of all buyers

```
SELECT AVG(Account_Balance) AS Average_Account_Balance
FROM Buyers;
```

```
MariaDB [dbms]> SELECT AVG(Account_Balance) AS Average_Account_Balance
    -> FROM Buyers;
+-----------------------------+
| Average_Account_Balance |
+-----------------------------+
|             5192.000000 |
+-----------------------------+
1 row in set (0.001 sec)
```

5. **Complex Query:** Get the total number of auctions that ended with a successful bid

```
SELECT COUNT(DISTINCT A.Auction_ID) AS Successful_Auctions
FROM Auctions A
INNER JOIN Bids B ON A.Item_ID = B.Item_ID
WHERE B.Bid_Status = 'Successful';
```

```
MariaDB [dbms]> SELECT COUNT(DISTINCT A.Auction_ID) AS Successful_Auctions
    -> FROM Auctions A
    -> INNER JOIN Bids B ON A.Item_ID = B.Item_ID
    -> WHERE B.Bid_Status = 'Successful';
+----------------------+
| Successful_Auctions |
+----------------------+
|                    0 |
+----------------------+
1 row in set (0.003 sec)
```

6. **Nested Query:** Get all items listed in a specific category

```
SELECT *
FROM Items
WHERE Category = (
    SELECT Category
    FROM Items
    GROUP BY Category
    HAVING COUNT(*) > 1
    LIMIT 1
);
```

```
MariaDB [dbms]> SELECT *
    -> FROM Items
    -> WHERE Category = (
    ->     SELECT Category
    ->     FROM Items
    ->     GROUP BY Category
    ->     HAVING COUNT(*) > 1
    ->     LIMIT 1
    -> );
+---------+-----------+--------------+-----------------------------------+----------------+---------------------+------------+
| Item_ID | Seller_ID | Item_Name    | Description                       | Starting_Price | Auction_End_Time    | Category   |
+---------+-----------+--------------+-----------------------------------+----------------+---------------------+------------+
|       6 |         6 | Kitchen Mixer   | Bajaj GX7 500W mixer grinder      |        1500.00 | 2024-05-09 08:00:00 | Appliances |
|      16 |        16 | Air Purifier    | Philips 2000 series air purifier  |       10000.00 | 2024-05-19 13:00:00 | Appliances |
|      27 |        27 | Barbecue Grill  | Weber Spirit II E-310 gas grill   |       20000.00 | 2024-05-29 15:00:00 | Appliances |
|      33 |        33 | Coffee Maker    | Philips 12-cup coffee maker       |        3000.00 | 2024-06-06 10:00:00 | Appliances |
|      37 |        37 | Microwave Oven  | Samsung 28L Convection Microwave  |       10000.00 | 2024-06-10 18:00:00 | Appliances |
|      43 |        43 | Electric Kettle | Philips 1.5L electric kettle      |        1500.00 | 2024-06-16 10:00:00 | Appliances |
|      48 |        48 | Blender         | NutriBullet Pro 900W blender      |        5000.00 | 2024-06-21 09:00:00 | Appliances |
+---------+-----------+--------------+-----------------------------------+----------------+---------------------+------------+
7 rows in set (0.003 sec)
```

TRIGGERS AND PROCEDURE

a) Write TRIGGERS on tables for entered data validations, updating derived attributes, updating foreign key data updates, etc. See that how TRIIGERS will help to make correct, non-repeated data entry.

Triggers:

1. Unique Buyer and Seller username

TRIGGER CODE:

```
DELIMITER //

CREATE TRIGGER trg_unique_username
BEFORE INSERT ON Buyers
FOR EACH ROW
BEGIN
    DECLARE username_count INT;
    SELECT COUNT(*) INTO username_count FROM Buyers WHERE Username = NEW.Username;
    IF username_count > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Username already exists in
Buyers table';
    END IF;
END //

CREATE TRIGGER trg_unique_username_seller
BEFORE INSERT ON Sellers
FOR EACH ROW
BEGIN
    DECLARE username_count INT;
    SELECT COUNT(*) INTO username_count FROM Sellers WHERE Username =
NEW.Username;
    IF username_count > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Username already exists in
Sellers table';
    END IF;
END //
DELIMITER ;
```

TRIGGER EXPLANATION:

These triggers ensure that a username is unique within the respective tables **Buyers** and **Sellers**.

1. **Trigger for Buyers Table (trg_unique_username):**
    - This trigger is fired before inserting a new row into the **Buyers** table.
    - It checks if the **Username** being inserted already exists in the **Buyers** table.
    - If a record with the same **Username** already exists, it raises an error using **SIGNAL SQLSTATE** with a custom error message.
    - This prevents the insertion of a duplicate username into the **Buyers** table.
2. **Trigger for Sellers Table (trg_unique_username_seller):**
    - This trigger is similar to the previous one but applies to the **Sellers** table.
    - It is fired before inserting a new row into the **Sellers** table.
    - It checks if the **Username** being inserted already exists in the **Sellers** table.
    - If a record with the same **Username** already exists, it raises an error using **SIGNAL SQLSTATE** with a custom error message.
    - This ensures that usernames are unique within the **Sellers** table.

```
INSERT INTO Buyers (Buyer_ID, Username, Password, Email, Address,
Account_Balance)  VALUES (99, 'raj_sharma', 'password123', 'raj@example.com', '123
Street', 1000.00);

INSERT INTO Sellers (Seller_ID, Username, Password, Email, Address,
Account_Balance)  VALUES (99, 'fashion_hub', 'password456', 'fashion@example.com',
'456 Avenue', 2000.00);
```

If we try to insert a duplicate username for either buyers or sellers, the triggers will prevent it and raise an error.

2. Set Last_Bid Default Value

TRIGGER CODE:

```
DELIMITER //

CREATE TRIGGER set_last_bid_default BEFORE INSERT ON items
FOR EACH ROW
BEGIN
    IF NEW.Last_bid IS NULL THEN
        SET NEW.Last_bid = NEW.Starting_Price;
    END IF;
END//

DELIMITER ;
```

TRIGGER EXPLANATION:

This trigger ensures that when a new row is inserted into the **Items** table, the **Last_bid** field is set to the **Starting_Price** if it's initially NULL.

Trigger Description:

- **Trigger Name: set_last_bid_default**
- **Event: BEFORE INSERT** on the **Items** table.
- **Trigger Type: FOR EACH ROW**, meaning it will execute for each row being inserted.
- **Action:** Sets the **Last_bid** field to the **Starting_Price** if **Last_bid** is initially NULL for the row being inserted.

Explanation:

1. **IF Statement:**
    - Checks if the **Last_bid** field for the new row being inserted is NULL (**IF NEW.Last_bid IS NULL**).
2. **Setting Default Value:**
    - If the **Last_bid** is indeed NULL, it sets the **Last_bid** field to the **Starting_Price** for that item (**SET NEW.Last_bid = NEW.Starting_Price;**).

Purpose:

- This trigger ensures that when a new item is added to the **Items** table and no previous bids have been made (thus **Last_bid** is NULL), the **Last_bid** field is initialized with the **Starting_Price** of the item. This helps to maintain consistency in the data and ensures that there is always a starting point for bidding on an item.

3. Update Last Bid trigger

TRIGGER CODE:

```
DELIMITER //
CREATE TRIGGER trg_update_last_bid
AFTER INSERT ON Bids
FOR EACH ROW
BEGIN
    UPDATE Items
    SET Last_Bidder = NEW.Bidder_ID,
        Last_Bid = NEW.Bid_Amount
    WHERE Item_ID = NEW.Item_ID;
END //
DELIMITER ;
```

TRIGGER EXPLANATION:

This trigger, named **trg_update_last_bid**, is designed to update the **Last_Bidder** and **Last_Bid** fields in the **Items** table whenever a new bid is inserted into the **Bids** table.

Trigger Description:

- **Trigger Name: trg_update_last_bid**
- **Event: AFTER INSERT** on the **Bids** table.
- **Trigger Type: FOR EACH ROW**, indicating that the trigger will execute once for each row that is inserted into the **Bids** table.
- **Action:** Updates the **Last_Bidder** and **Last_Bid** fields in the **Items** table based on the information of the newly inserted bid.

Explanation:

1. **UPDATE Statement:**

- The trigger executes an **UPDATE** statement on the **Items** table.
- It sets the **Last_Bidder** field in the **Items** table to the **Bidder_ID** of the newly inserted bid (**SET Last_Bidder = NEW.Bidder_ID**).
- It also updates the **Last_Bid** field in the **Items** table to the **Bid_Amount** of the newly inserted bid (**SET Last_Bid = NEW.Bid_Amount**).

2. **WHERE Clause:**
    - The **UPDATE** statement applies these changes only to the row in the **Items** table where the **Item_ID** matches the **Item_ID** of the newly inserted bid (**WHERE Item_ID = NEW.Item_ID**).

Purpose:

- This trigger ensures that whenever a new bid is made on an item (**Bids** table), the corresponding **Last_Bidder** and **Last_Bid** fields in the **Items** table are updated accordingly. This helps to keep track of the latest bid and the bidder for each item without requiring manual updates.

EXAMPLE:

```
INSERT INTO Bids (Bid_ID, Bidder_ID, Item_ID, Bid_Amount, Bid_Time, Bid_Status,
Bid_Increment) VALUES (123, 1, 19, 200.00, NOW(), 'Active', 0.00);


SELECT Last_Bidder, Last_Bid FROM Items WHERE Item_ID = 19;
```

After inserting a new bid into the **Bids** table with **Item_ID** 19, the trigger will automatically update the **Last_Bidder** and **Last_Bid** fields in the **Items** table. We can then retrieve this updated information using the **SELECT** statement provided.

4. Update Buyer and Seller balance

TRIGGER CODE:

```
DELIMITER //

CREATE TRIGGER trg_update_buyer_balance
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
    DECLARE current_balance DECIMAL(10, 2);
    SELECT Account_Balance INTO current_balance FROM Buyers WHERE Buyer_ID =
NEW.Buyer_ID;

    IF current_balance >= NEW.Transaction_Amount THEN
        UPDATE Buyers
        SET Account_Balance = current_balance - NEW.Transaction_Amount
        WHERE Buyer_ID = NEW.Buyer_ID;
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient balance for the
transaction';
    END IF;
END //
DELIMITER ;


DELIMITER //
CREATE TRIGGER trg_update_seller_balance
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
    UPDATE Sellers
    SET Account_Balance = Account_Balance + NEW.Transaction_Amount
    WHERE Seller_ID = NEW.Seller_ID;
END //
DELIMITER ;
```

TRIGGER EXPLANATION:

These two triggers, **trg_update_buyer_balance** and **trg_update_seller_balance**, are designed to update the account balances of buyers and sellers respectively after a new transaction is inserted into the **Transactions** table.

Trigger trg_update_buyer_balance:

- **Trigger Name:** trg_update_buyer_balance
- **Event:** AFTER INSERT on the Transactions table.

- **Trigger Type:** FOR EACH ROW, meaning it executes once for each row that is inserted into the Transactions table.
- **Purpose:** Update the account balance of the buyer who made the transaction.

Explanation:

1. **DECLARE Statement:**
    - Declares a local variable **current_balance** of type DECIMAL(10, 2) to store the current account balance of the buyer.
2. **SELECT Statement:**
    - Retrieves the current account balance of the buyer (**Account_Balance**) from the Buyers table based on the **Buyer_ID** of the newly inserted transaction (**NEW.Buyer_ID**).
3. **IF Statement:**
    - Checks if the current balance (**current_balance**) is sufficient for the transaction amount (**NEW.Transaction_Amount**).
4. **UPDATE Statement (IF condition is met):**
    - If the current balance is sufficient, it updates the buyer's account balance by subtracting the transaction amount.
5. **SIGNAL Statement (IF condition is not met):**
    - If the current balance is insufficient, it raises an error using SIGNAL SQLSTATE with a custom error message indicating "Insufficient balance for the transaction".

Trigger trg_update_seller_balance:

- **Trigger Name:** trg_update_seller_balance
- **Event:** AFTER INSERT on the Transactions table.
- **Trigger Type:** FOR EACH ROW.
- **Purpose:** Update the account balance of the seller who made the transaction.

Explanation:

1. **UPDATE Statement:**
    - Updates the account balance of the seller by adding the transaction amount (**NEW.Transaction_Amount**) to their existing account balance.

Tests:

- After inserting a new transaction into the Transactions table, you can verify the updated account balances of the buyer and seller by querying the Buyers and Sellers tables respectively.

EXAMPLE:

```
INSERT INTO Transactions (Transaction_ID, Buyer_ID, Seller_ID, Item_ID,
Transaction_Amount, Transaction_Time, Payment_Method, Transaction_Status) VALUES
(456, 1, 2, 19, 500.00, NOW(), 'Credit Card', 'Success');

SELECT Account_Balance FROM Buyers WHERE Buyer_ID = 1;

SELECT Account_Balance FROM Sellers WHERE Seller_ID = 2;
```

6. Set Winning Bid After Auction Update

TRIGGER CODE:

```
DELIMITER //
CREATE TRIGGER after_auction_update_set_winning_bid_trigger
AFTER UPDATE ON Auctions
FOR EACH ROW
BEGIN
    DECLARE max_bid_id INT;
    DECLARE bidder_id INT;
    DECLARE winning_bid DECIMAL(10, 2);

    IF NEW.Auction_Status = 'Closed' AND OLD.Auction_Status != 'Closed' THEN

        -- Selecting the highest bid amount for the item
        SELECT MAX(Bid_Amount) INTO winning_bid FROM Bids WHERE Item_ID =
NEW.Item_ID;

        IF winning_bid IS NOT NULL THEN
            -- Selecting the maximum bid ID and incrementing it by 1 for the new
bid
            SELECT COALESCE(MAX(Bid_ID), 0) + 1 INTO max_bid_id FROM Bids;

            -- Selecting the bidder ID associated with the highest bid
            SELECT Bidder_ID INTO bidder_id FROM Bids WHERE Item_ID = NEW.Item_ID
AND Bid_Amount = winning_bid ORDER BY Bid_Time DESC LIMIT 1;

            -- Inserting the winning bid into the Bids table
            IF bidder_id IS NOT NULL THEN
                INSERT INTO Bids (Bid_ID, Bidder_ID, Item_ID, Bid_Amount,
Bid_Time, Bid_Status, Bid_Increment)
                VALUES (max_bid_id, bidder_id, NEW.Item_ID, winning_bid, NOW(),
'Winning', 0);
            END IF;
        END IF;
    END IF;
END //
DELIMITER ;
```

TRIGGER EXPLANATION:

This trigger, named **after_auction_update_set_winning_bid_trigger**, is designed to automatically set the winning bid for an item in the **Bids** table after an auction is closed.

Trigger Description:

- **Trigger Name:** after_auction_update_set_winning_bid_trigger
- **Event:** AFTER UPDATE on the **Auctions** table.
- **Trigger Type:** FOR EACH ROW, meaning it executes once for each row that is updated in the **Auctions** table.
- **Purpose:** Automatically sets the winning bid for an item in the **Bids** table when the corresponding auction is closed.

Explanation:

1. **IF Statement:**
    - Checks if the **Auction_Status** has changed to 'Closed' (**IF NEW.Auction_Status = 'Closed' AND OLD.Auction_Status != 'Closed'**), indicating that the auction has been closed.
2. **Selecting the Winning Bid:**
    - If the auction is closed, it selects the highest bid amount (**MAX(Bid_Amount)**) for the item associated with the closed auction and stores it in the **winning_bid** variable.
3. **Handling Winning Bid Existence:**
    - If a winning bid exists (**winning_bid IS NOT NULL**), it proceeds to select the maximum bid ID and increments it by 1 to generate a new bid ID for the winning bid.
    - It then selects the bidder ID associated with the highest bid amount for the item.
4. **Inserting Winning Bid:**
    - If a bidder ID is found (**bidder_id IS NOT NULL**), it inserts a new row into the **Bids** table with the generated bid ID, bidder ID, item ID, winning bid amount, current timestamp, 'Winning' status, and zero bid increment.

Test:

- The provided test updates the **Auction_Status** of a specific auction (**Auction_ID = 24**) to 'Closed' and then selects the winning bid from the **Bids** table for the item associated with that auction.

Example:

```
UPDATE Auctions SET Auction_Status = 'Closed' WHERE Auction_ID = 24;
SELECT * FROM Bids WHERE Item_ID = (SELECT Item_ID FROM Auctions WHERE Auction_ID
= 24) AND Bid_Status = 'Winning';
```

7. Insert Transaction on Auction End
   TRIGGER CODE:

```
DELIMITER //
CREATE TRIGGER end_auction_insert_transactions_trigger
BEFORE UPDATE ON Auctions
FOR EACH ROW
BEGIN
    DECLARE new_transaction_id INT;

    -- Check if the auction end time is in the past or now
    IF OLD.Auction_Status != 'Closed' AND NEW.Auction_Status = 'Closed' AND
NEW.Auction_End_Time <= NOW() THEN
        -- Generate a new transaction ID by incrementing the maximum transaction
ID
        SET new_transaction_id = (SELECT COALESCE(MAX(Transaction_ID), 0) + 1 FROM
Transactions);

        -- Fetch the Last_Bidder and Last_Bid from the Items table based on the
Item_ID associated with the auction
        SET @bidder_username := (SELECT Last_Bidder FROM Items WHERE Item_ID =
NEW.Item_ID);
        SET @last_bid := (SELECT Last_Bid FROM Items WHERE Item_ID = NEW.Item_ID);

        -- Fetch the Buyer_ID using the bidder's username
        SET @buyer_id := (SELECT Buyer_ID FROM Buyers WHERE Username =
@bidder_username);

        -- Insert a new transaction into the Transactions table
        INSERT INTO Transactions (Transaction_ID, Buyer_ID, Seller_ID, Item_ID,
Transaction_Amount, Transaction_Time, Payment_Method, Transaction_Status)
        VALUES (
            new_transaction_id, -- New transaction ID
            @buyer_id, -- Buyer ID based on Last_Bidder
            (SELECT Seller_ID FROM Items WHERE Item_ID = NEW.Item_ID), -- Seller
ID from the auction
```

```
            NEW.Item_ID, -- Item ID from the auction
            @last_bid, -- Last bid amount from the auction
            NOW(), -- Current timestamp
            'Online', -- Payment method (assuming online)
            'Completed' -- Transaction status (assuming completed)
        );

        -- Update the buyer's account balance by subtracting the last bid amount
        UPDATE Buyers
        SET Account_Balance = Account_Balance - @last_bid
        WHERE Buyer_ID = @buyer_id;
    END IF;
END //
DELIMITER ;
```

TRIGGER EXPLANATION:

This trigger, named **end_auction_insert_transactions_trigger**, is designed to automatically insert a transaction into the **Transactions** table and update the buyer's account balance when an auction is closed.

Trigger Description:

- **Trigger Name:** end_auction_insert_transactions_trigger
- **Event:** BEFORE UPDATE on the **Auctions** table.
- **Trigger Type:** FOR EACH ROW, indicating that the trigger will execute once for each row that is updated in the **Auctions** table.
- **Purpose:** Automatically inserts a transaction into the **Transactions** table and updates the buyer's account balance when the auction is closed and its end time has passed.

Explanation:

1. **IF Statement:**
    - Checks if the **Auction_Status** has changed to 'Closed' (**IF OLD.Auction_Status != 'Closed' AND NEW.Auction_Status = 'Closed'**) and if the **Auction_End_Time** is less than or equal to the current time (**AND NEW.Auction_End_Time <= NOW()**), indicating that the auction has ended.
2. **Generating Transaction ID:**
    - Generates a new transaction ID by incrementing the maximum transaction ID found in the **Transactions** table.
3. **Fetching Bidder Information:**
    - Retrieves the **Last_Bidder** and **Last_Bid** from the **Items** table based on the **Item_ID** associated with the auction.
4. **Fetching Buyer ID:**

- Retrieves the **Buyer_ID** using the bidder's username obtained in the previous step.

5. **Inserting Transaction:**
   - Inserts a new row into the **Transactions** table with the generated transaction ID, buyer ID, seller ID (obtained from the **Items** table), item ID, last bid amount, current timestamp, payment method ('Online'), and transaction status ('Completed').
6. **Updating Buyer's Account Balance:**
   - Updates the buyer's account balance by subtracting the last bid amount (**@last_bid**).

Test:

- The provided test doesn't explicitly show how the trigger works but assumes that an auction with a specific **Auction_ID** has ended, triggering the update on the **Auctions** table. The trigger then automatically inserts a transaction into the **Transactions** table and updates the buyer's account balance.

Example:

```
UPDATE Auctions
SET Auction_Status = 'Closed'
WHERE Auction_ID = 16

-- Check the latest transaction
SELECT * FROM Transactions ORDER BY Transaction_ID DESC LIMIT 1;

-- Check the buyer's account balance
SELECT * FROM Buyers WHERE Buyer_ID = buyer_id;
```

After executing this update statement, the trigger will automatically insert a transaction into the **Transactions** table and update the buyer's account balance if the conditions specified in the trigger are met.

8. Update Auction Status
   TRIGGER CODE:

```
DELIMITER $$
CREATE TRIGGER update_auction_status
BEFORE INSERT ON auctions
FOR EACH ROW
BEGIN
    IF NEW.Auction_End_Time <= NOW() THEN
        SET NEW.Auction_Status = 'Closed';
    END IF;
END$$
DELIMITER ;


test
-- Insert a new auction with Auction_End_Time set to a past time
INSERT INTO auctions (Auction_ID, Item_ID, Auction_Start_Time, Auction_End_Time,
Auction_Status, Reserve_Price)
VALUES (99, 1, '2024-05-01 12:00:00', '2024-05-15 12:00:00', 'pen', 100.00);

-- Check the inserted auction
SELECT * FROM auctions WHERE Auction_ID = 99;
```

TRIGGER EXPLANATION:

This trigger, named **update_auction_status**, is designed to automatically update the **Auction_Status** field when a new auction is inserted into the **auctions** table.

Trigger Description:

- **Trigger Name:** update_auction_status
- **Event:** BEFORE INSERT on the **auctions** table.
- **Trigger Type:** FOR EACH ROW, indicating that the trigger will execute once for each row that is being inserted into the **auctions** table.
- **Purpose:** Automatically updates the **Auction_Status** field to 'Closed' if the **Auction_End_Time** for the new auction is in the past or equals the current time.

Explanation:

1. **IF Statement:**
   - Checks if the **Auction_End_Time** for the new auction (**NEW.Auction_End_Time**) is less than or equal to the current time (**NOW()**).
2. **Updating Auction Status:**

- If the **Auction_End_Time** is indeed in the past or equals the current time, it sets the **Auction_Status** for the new auction (**NEW.Auction_Status**) to 'Closed'.

Test:

- The provided test inserts a new auction into the **auctions** table with an **Auction_End_Time** that is set to a past time ('2024-05-15 12:00:00'). This should trigger the execution of the trigger, automatically updating the **Auction_Status** to 'Closed'.

Example:

```
-- Insert a new auction with Auction_End_Time set to a past time

INSERT INTO auctions (Auction_ID, Item_ID, Auction_Start_Time, Auction_End_Time,
Auction_Status, Reserve_Price)
VALUES (99, 1, '2024-05-01 12:00:00', '2024-05-15 12:00:00', 'pen', 100.00);

-- Check the inserted auction
SELECT * FROM auctions WHERE Auction_ID = 99;
```

After executing this insert statement, the trigger will automatically update the **Auction_Status** to 'Closed' for the newly inserted auction because its **Auction_End_Time** is in the past. When you check the inserted auction, you should see the updated **Auction_Status** reflecting this change.

9. Create Bid Entry on Update in items
   TRIGGER CODE:

```
DELIMITER $$
DROP TRIGGER IF EXISTS create_bid_entry_after_update $$
CREATE TRIGGER create_bid_entry_after_update
AFTER UPDATE ON items
FOR EACH ROW
BEGIN
    DECLARE bidder_id INT;
    DECLARE last_bid DECIMAL(10, 2);
    DECLARE bid_increment DECIMAL(10, 2);

    SET bidder_id = (SELECT Buyer_ID FROM buyers WHERE Username =
NEW.Last_Bidder);

    SELECT Last_Bid INTO last_bid FROM items WHERE Item_ID = NEW.Item_ID;

    SET bid_increment = GREATEST(NEW.Last_Bid - COALESCE(last_bid, 0), 0);

    INSERT INTO bids (Bid_ID, Bidder_ID, Item_ID, Bid_Amount, Bid_Time,
Bid_Status, Bid_Increment)
    SELECT COALESCE(MAX(Bid_ID), 0) + 1, bidder_id, NEW.Item_ID, NEW.Last_Bid,
NOW(), 'Outbid', bid_increment FROM bids;
END$$
DELIMITER ;
```

TRIGGER EXPLANATION:

This trigger, named **create_bid_entry_after_update**, is designed to automatically create a new bid entry in the **bids** table after an update on the **items** table.

Trigger Description:

- **Trigger Name:** create_bid_entry_after_update
- **Event:** AFTER UPDATE on the **items** table.
- **Trigger Type:** FOR EACH ROW, indicating that the trigger will execute once for each row that is updated in the **items** table.
- **Purpose:** Automatically creates a new bid entry in the **bids** table when the **Last_Bid** field of an item is updated.

Explanation:

1. **Declaration of Variables:**
   - Declares three variables: **bidder_id** to store the ID of the bidder, **last_bid** to store the last bid amount for the item, and **bid_increment** to store the difference between the new bid amount and the previous bid amount.

2. **Fetching Bidder ID:**
    - Sets the **bidder_id** variable by selecting the **Buyer_ID** from the **buyers** table based on the username (**NEW.Last_Bidder**) provided in the updated row.
3. **Fetching Last Bid:**
    - Selects the **Last_Bid** from the **items** table based on the **Item_ID** of the updated row and stores it in the **last_bid** variable.
4. **Calculating Bid Increment:**
    - Calculates the bid increment by finding the difference between the new bid amount (**NEW.Last_Bid**) and the previous bid amount (**last_bid**). It ensures that the bid increment is always non-negative.
5. **Inserting Bid Entry:**
    - Inserts a new row into the **bids** table with the following values:
        - **Bid_ID**: The maximum bid ID incremented by 1.
        - **Bidder_ID**: The ID of the bidder fetched earlier.
        - **Item_ID**: The **Item_ID** of the updated row.
        - **Bid_Amount**: The new bid amount (**NEW.Last_Bid**).
        - **Bid_Time**: The current timestamp.
        - **Bid_Status**: 'Outbid' indicating that the previous bid has been outbid.
        - **Bid_Increment**: The calculated bid increment.

Test:

- This trigger is automatically executed after an update is performed on the **items** table. You can simulate this by updating the **Last_Bid** field of an item in the **items** table.

Example:

```
-- Update the Last_Bid field of an item in the items table
UPDATE items SET Last_Bid = 150.00 WHERE Item_ID = 1;

-- This update will trigger the execution of the trigger
create_bid_entry_after_update
```

After executing this update statement, the trigger will automatically create a new bid entry in the **bids** table based on the updated **Last_Bid** value for the item with **Item_ID** 1.

b) Write PROCEDURES to minimize the complexity of AGGREGATE functions used in SQL Query, to create NEW FUNCTIONS for which AGGREGATE function is not available, etc.

PROCEDURES:

1. **CalculateAverageTransaction**: This procedure calculates the average transaction amount for a given seller ID. It takes the seller ID as input and calculates the average transaction amount by querying the Transactions table for transactions made by that seller. The result is then returned.

   PROCEDURE CODE:

```
DELIMITER //

CREATE PROCEDURE CalculateAverageTransaction(IN sellerID INT)
BEGIN
    DECLARE avgTransaction DECIMAL(10, 2);

    SELECT AVG(Transaction_Amount) INTO avgTransaction
    FROM Transactions
    WHERE Seller_ID = sellerID;

    SELECT avgTransaction;
END //

DELIMITER ;
```

   PROCEDURE EXAMPLE:

```
CALL CalculateAverageTransaction(1);
```

2. **CountBidsForItem**: This procedure counts the number of bids for a given item ID. It takes the item ID as input and calculates the count by querying the Bids table for bids made on that item. The result is then returned.

PROCEDURE CODE:

```
DELIMITER //
```

```
CREATE PROCEDURE CountBidsForItem(IN itemID INT)
BEGIN
    DECLARE bidCount INT;

    SELECT COUNT(*) INTO bidCount
    FROM Bids
    WHERE Item_ID = itemID;

    SELECT bidCount;
END //

DELIMITER ;
```

PROCEDURE EXAMPLE:

```
CALL CountBidsForItem(1);
```

3. **FindMaxBidAmountForItem**: This function finds the maximum bid amount for a given item ID. It takes the item ID as input and calculates the maximum bid amount by querying the Bids table for bids made on that item. The maximum bid amount is then returned.

PROCEDURE CODE:

```
DELIMITER //

CREATE FUNCTION FindMaxBidAmountForItem(itemID INT) RETURNS DECIMAL(10, 2)
BEGIN
    DECLARE maxBid DECIMAL(10, 2);

    SELECT MAX(Bid_Amount) INTO maxBid
    FROM Bids
    WHERE Item_ID = itemID;

    RETURN maxBid;
END //

DELIMITER ;
```

PROCEDURE EXAMPLE:

```
SELECT FindMaxBidAmountForItem(1);
```

4. **CalculateTotalSalesAmount**: This procedure calculates the total sales amount for a given seller ID. It takes the seller ID as input and calculates the total sales amount by querying the Transactions table for transactions made by that seller. The result is then returned.

PROCEDURE CODE:

```
DELIMITER //

CREATE PROCEDURE CalculateTotalSalesAmount(IN sellerID INT)
BEGIN
    DECLARE totalSales DECIMAL(10, 2);

    SELECT SUM(Transaction_Amount) INTO totalSales
    FROM Transactions
    WHERE Seller_ID = sellerID;

    SELECT totalSales;
END //

DELIMITER ;
```

PROCEDURE EXAMPLE:

```
CALL CalculateTotalSalesAmount(1);
```

5. **CalculateAverageAuctionDuration**: This procedure calculates the average duration of auctions in a given category. It takes the category as input and calculates the average duration by querying the Auctions and Items tables, joining them on the Item_ID, and filtering by the specified category. The duration of each auction is calculated using the difference between the auction start and end times. The average duration is then returned.

PROCEDURE CODE:

```
DELIMITER //

DROP PROCEDURE IF EXISTS CalculateAverageAuctionDuration;
DELIMITER //
```

```
CREATE PROCEDURE CalculateAverageAuctionDuration(IN category VARCHAR(50))
BEGIN
    DECLARE avgDuration DECIMAL(10, 2);

    SELECT AVG(TIMESTAMPDIFF(SECOND, Auctions.Auction_Start_Time,
Auctions.Auction_End_Time)) INTO avgDuration
    FROM Auctions
    INNER JOIN Items ON Auctions.Item_ID = Items.Item_ID
    WHERE Items.Category = category;

    SELECT avgDuration;
END //

DELIMITER ;
```

PROCEDURE EXAMPLE:

```
CALL CalculateAverageAuctionDuration('Electronics');
```

6. **CountItemsSoldByBuyer**: This function counts the number of items sold by a given buyer ID. It takes the buyer ID as input and calculates the count by querying the Transactions table for transactions where the specified buyer is the buyer. The count is then returned.

PROCEDURE CODE:

```
DELIMITER //

CREATE FUNCTION CountItemsSoldByBuyer(buyerID INT) RETURNS INT
BEGIN
    DECLARE itemCount INT;

    SELECT COUNT(*) INTO itemCount
    FROM Transactions
    WHERE Buyer_ID = buyerID;

    RETURN itemCount;
END //

DELIMITER ;
```

PROCEDURE EXAMPLE:

```
SELECT CountItemsSoldByBuyer(1);
```

Declare which part of the database has triggers and procedures defined:

In most modern database systems, the Information Schema contains views related to triggers, providing insights into their definitions and associated objects. Commonly, these views include:

- **Triggers View**: This view typically contains metadata about triggers defined in the database, including their names, associated tables, events, and actions.
- **Trigger Columns View**: Some systems offer this view to provide details about the columns affected by each trigger.

Procedures in the Information Schema:

Similar to triggers, procedures also have their metadata exposed through the Information Schema, facilitating querying and analysis. Relevant views may include:

- **Routines View**: This view usually lists all routines (including procedures and functions) defined in the database, along with their names, types, and other attributes.
- **Routine Parameters View**: If procedures accept parameters, this view provides information about the parameters, such as names, data types, and positions.

TO VIEW ALL TRIGGERS:

```
SELECT
    TRIGGER_NAME,
    EVENT_OBJECT_TABLE
FROM
    INFORMATION_SCHEMA.TRIGGERS;
```

```
+-----------------------------------------------+---------------------+
| TRIGGER_NAME                                  | EVENT_OBJECT_TABLE  |
+-----------------------------------------------+---------------------+
| update_auction_status                         | auctions            |
| end_auction_insert_transactions_trigger       | auctions            |
| after_auction_update_set_winning_bid_trigger  | auctions            |
| trg_update_last_bid                           | bids                |
| trg_unique_username                           | buyers              |
| set_last_bid_default                          | items               |
| create_bid_entry_after_update                 | items               |
| trg_unique_username_seller                    | sellers             |
| trg_update_seller_balance                     | transactions        |
| trg_update_buyer_balance                      | transactions        |
+-----------------------------------------------+---------------------+
10 rows in set (0.046 sec)
```

TO VIEW ALL PROCEDURES:

```
SELECT
  ROUTINE_SCHEMA,
  ROUTINE_NAME
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_TYPE = 'PROCEDURE';
```

```
+----------------+-----------------------------------+
| ROUTINE_SCHEMA | ROUTINE_NAME                      |
+----------------+-----------------------------------+
| dbms           | CalculateAverageAuctionDuration   |
| dbms           | CalculateAverageTransaction       |
| dbms           | CalculateTotalSalesAmount         |
| dbms           | CountBidsForItem                  |
| sys            | create_synonym_db                 |
| sys            | diagnostics                       |
| sys            | execute_prepared_stmt             |
| sys            | optimizer_switch_choice           |
| sys            | optimizer_switch_off              |
| sys            | optimizer_switch_on               |
| sys            | ps_setup_disable_background_threads |
| sys            | ps_setup_disable_consumer         |
| sys            | ps_setup_disable_instrument       |
| sys            | ps_setup_disable_thread           |
| sys            | ps_setup_enable_background_threads |
| sys            | ps_setup_enable_consumer          |
| sys            | ps_setup_enable_instrument        |
| sys            | ps_setup_enable_thread            |
| sys            | ps_setup_reload_saved             |
| sys            | ps_setup_reset_to_default         |
| sys            | ps_setup_save                     |
| sys            | ps_setup_show_disabled            |
| sys            | ps_setup_show_disabled_consumers  |
| sys            | ps_setup_show_disabled_instruments |
| sys            | ps_setup_show_enabled             |
| sys            | ps_setup_show_enabled_consumers   |
| sys            | ps_setup_show_enabled_instruments |
| sys            | ps_statement_avg_latency_histogram |
| sys            | ps_trace_statement_digest         |
| sys            | ps_trace_thread                   |
| sys            | ps_truncate_all_tables            |
| sys            | statement_performance_analyzer    |
| sys            | table_exists                      |
| testlab        | GetMaxValue                       |
| testlab        | proc_avg                          |
+----------------+-----------------------------------+
35 rows in set (0.008 sec)
```

Explain the Data Dictionary details for Triggers and Procedures.

Triggers and procedures are defined within the schema of the online auction system database. Triggers are defined to enforce data integrity, handle automatic updates, and perform certain actions upon specific events in the database tables. Procedures are defined to encapsulate reusable logic for performing calculations or data analysis within the database.

Specifically, the triggers are defined for the following tables:

- Buyers
- Sellers
- Items
- Bids
- Auctions
- Transactions

The procedures are defined to perform various calculations and data analysis tasks within the database, such as calculating averages, counting bids, finding maximum bid amounts, and calculating total sales amounts.

Triggers:

1. **trg_unique_username:**
    - **Description:** Ensures uniqueness of usernames in the Buyers table.
    - **Event:** BEFORE INSERT ON Buyers
    - **Action:** Checks if the inserted username already exists in the Buyers table and signals an error if it does.
2. **trg_unique_username_seller:**
    - **Description:** Ensures uniqueness of usernames in the Sellers table.
    - **Event:** BEFORE INSERT ON Sellers
    - **Action:** Checks if the inserted username already exists in the Sellers table and signals an error if it does.
3. **set_last_bid_default:**

- **Description:** Sets the Last_Bid field to the Starting_Price if it is null when inserting into the Items table.
- **Event:** BEFORE INSERT ON Items
- **Action:** If the Last_Bid field is null, it sets it to the Starting_Price.

4. **trg_update_last_bid:**
    - **Description:** Updates the Last_Bidder and Last_Bid fields in the Items table after a bid is inserted.
    - **Event:** AFTER INSERT ON Bids
    - **Action:** Updates the Last_Bidder and Last_Bid fields in the Items table based on the inserted bid.
5. **trg_update_buyer_balance:**
    - **Description:** Updates the Account_Balance of the buyer after a transaction is inserted, checking for sufficient balance.
    - **Event:** AFTER INSERT ON Transactions
    - **Action:** Checks if the buyer has sufficient balance for the transaction and updates the Account_Balance accordingly.
6. **trg_update_seller_balance:**
    - **Description:** Updates the Account_Balance of the seller after a transaction is inserted.
    - **Event:** AFTER INSERT ON Transactions
    - **Action:** Updates the Account_Balance of the seller based on the inserted transaction.
7. **after_auction_update_set_winning_bid_trigger:**
    - **Description:** Sets the winning bid for an item after an auction is closed.
    - **Event:** AFTER UPDATE ON Auctions
    - **Action:** Sets the winning bid for an item if the auction status is changed to 'Closed'.
8. **end_auction_insert_transactions_trigger:**
    - **Description:** Inserts a transaction after an auction ends.

- **Event:** BEFORE UPDATE ON Auctions
- **Action:** Inserts a transaction and updates buyer's account balance after an auction ends.

9. **update_auction_status:**
    - **Description:** Updates the Auction_Status to 'Closed' if the Auction_End_Time is in the past when inserting into the Auctions table.
    - **Event:** BEFORE INSERT ON Auctions
    - **Action:** Sets the Auction_Status to 'Closed' if the Auction_End_Time is in the past.
10. **create_bid_entry_after_update:**

- **Description:** Inserts a bid entry after an item's Last_Bid is updated.
- **Event:** AFTER UPDATE ON Items
- **Action:** Inserts a bid entry with the updated Last_Bid value.

Procedures:

1. **CalculateAverageTransaction:**
    - **Description:** Calculates the average transaction amount for a given seller.
    - **Parameters:** sellerID (INT)
    - **Returns:** avgTransaction (DECIMAL)
2. **CountBidsForItem:**
    - **Description:** Counts the number of bids for a given item.
    - **Parameters:** itemID (INT)
    - **Returns:** bidCount (INT)
3. **FindMaxBidAmountForItem:**
    - **Description:** Finds the maximum bid amount for a given item.
    - **Parameters:** itemID (INT)
    - **Returns:** maxBid (DECIMAL)
4. **CalculateTotalSalesAmount:**

- **Description:** Calculates the total sales amount for a given seller.
- **Parameters:** sellerID (INT)
- **Returns:** totalSales (DECIMAL)

5. **CalculateAverageAuctionDuration:**
    - **Description:** Calculates the average duration of auctions for a given category.
    - **Parameters:** category (VARCHAR)
    - **Returns:** avgDuration (DECIMAL)
6. **CountItemsSoldByBuyer:**
    - **Description:** Counts the number of items sold by a given buyer.
    - **Parameters:** buyerID (INT)
    - **Returns:** itemCount (INT)

Write the SQL Queries for fetching/processing triggers and procedures details.

Fetch Triggers Details:

```
-- Fetch all triggers in the database
SHOW TRIGGERS;

-- Fetch details of a specific trigger
SHOW CREATE TRIGGER trg_name;
```

Fetch Procedures Details:

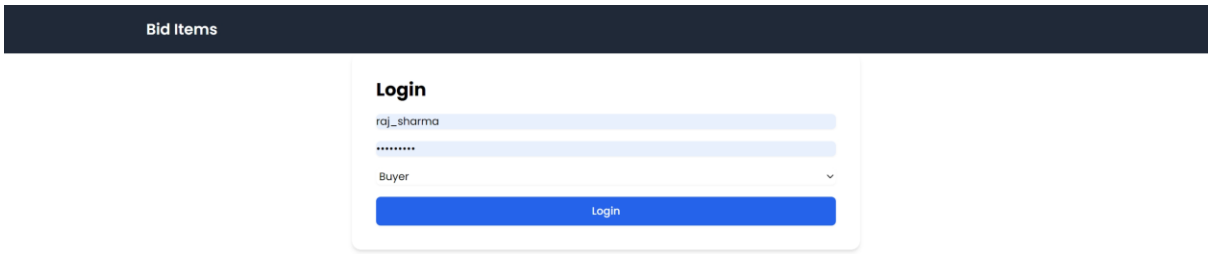
```
-- Fetch all procedures in the database
SHOW PROCEDURES;

-- Fetch details of a specific procedure
SHOW CREATE PROCEDURE proc_name;
```

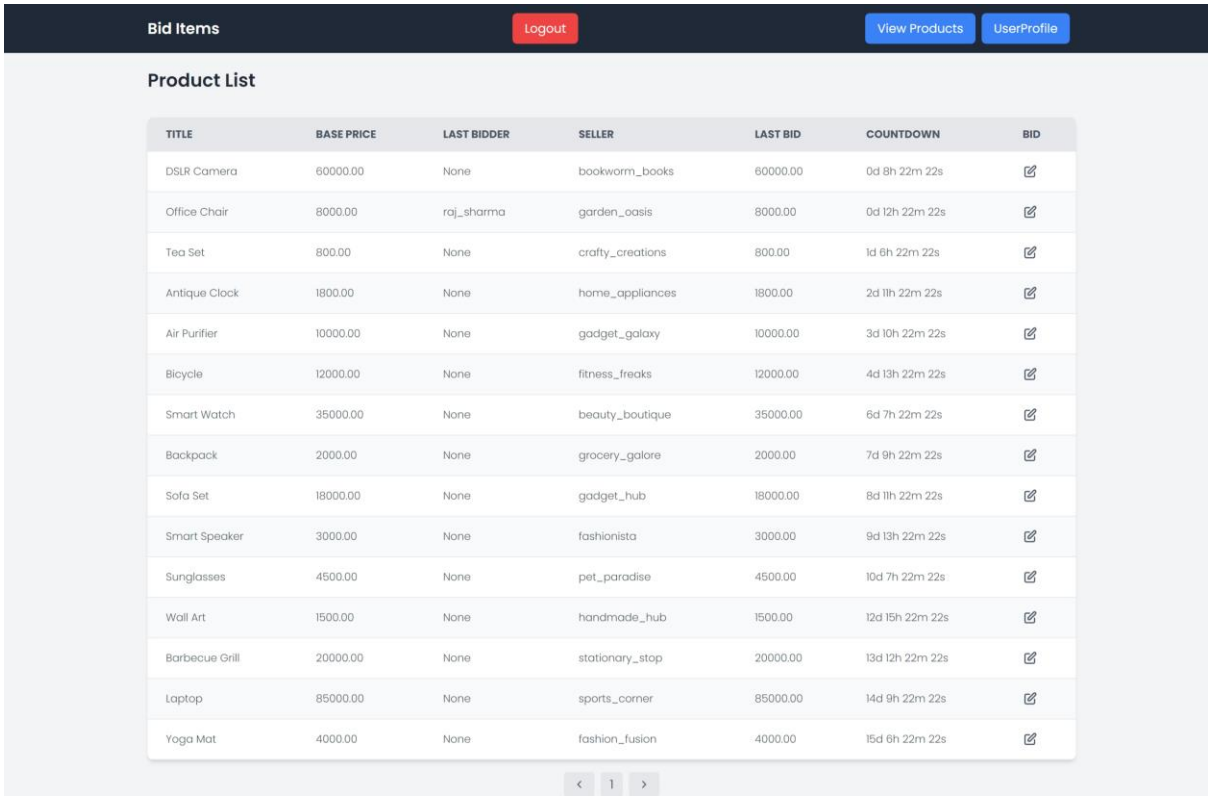
DATABASE CONNECTIVITY, GUI & REPORTS

BUYER PORTAL

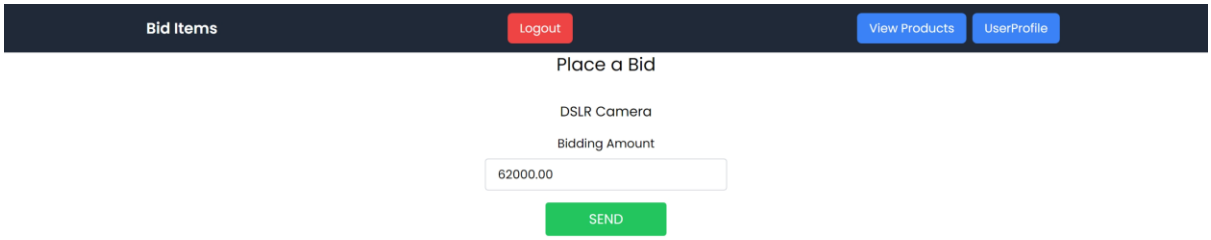
Login Page:



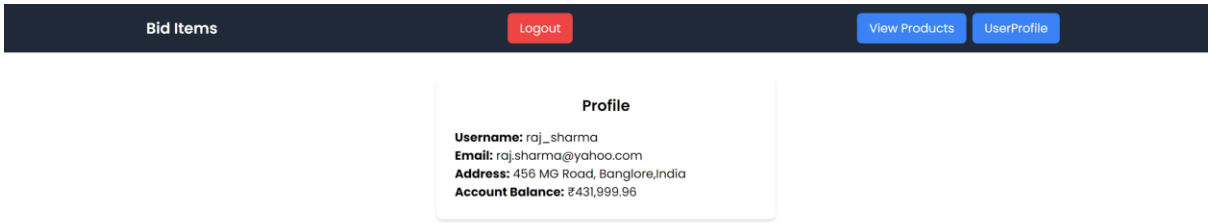
View Products Page:



Bid on Product Page:

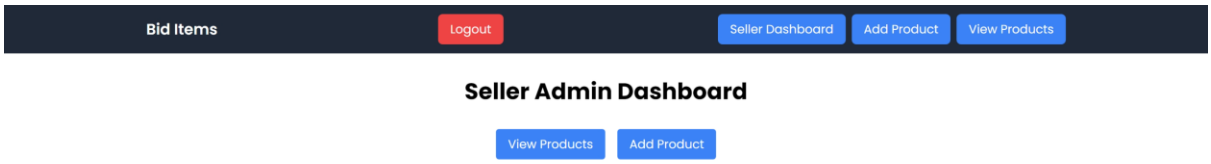


User Profile Page:



SELLER PORTAL

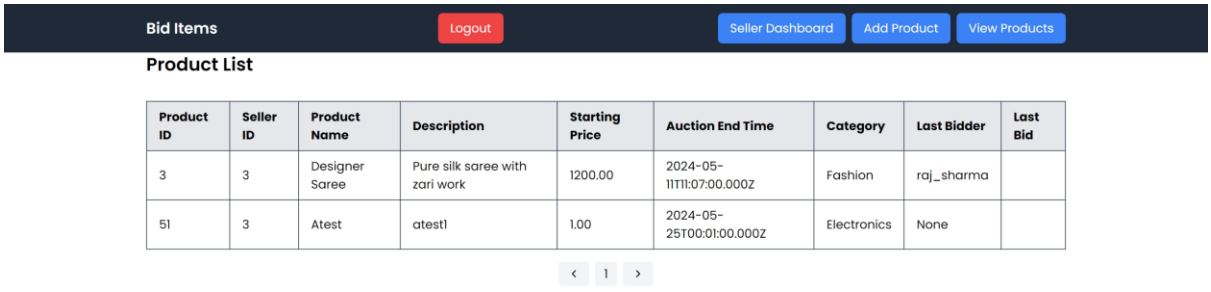
SELLER DASHBOARD PAGE:



Seller Admin Dashboard

View Products | Add Product

PRODUCTS LISTED BY SELLER PAGE:

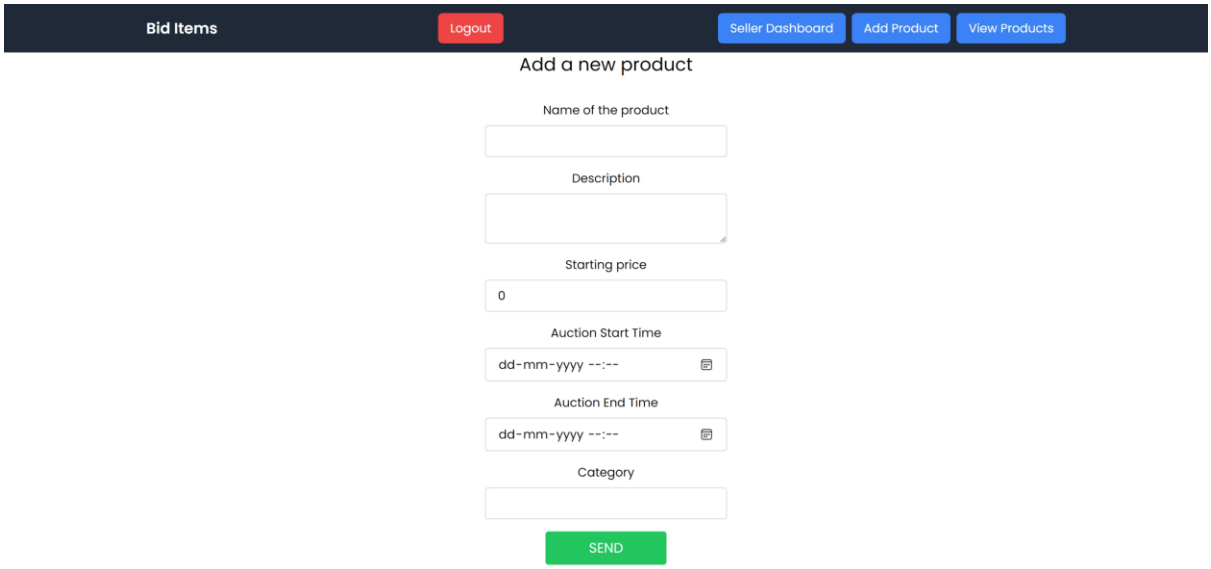


Product List

Product ID	Seller ID	Product Name	Description	Starting Price	Auction End Time	Category	Last Bidder	Last Bid
3	3	Designer Saree	Pure silk saree with zari work	1200.00	2024-05-11T11:07:00.000Z	Fashion	raj_sharma	
51	3	Atest	atest1	1.00	2024-05-25T00:01:00.000Z	Electronics	None	

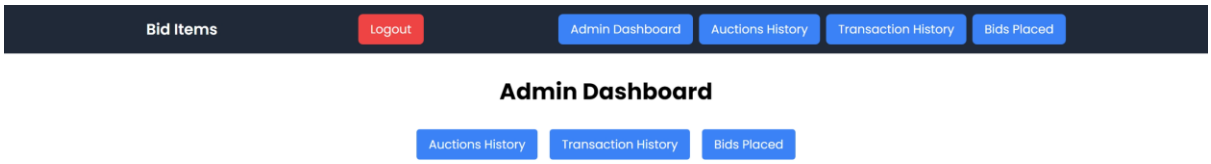
‹ 1 ›

LIST NEW PRODUCT PAGE:

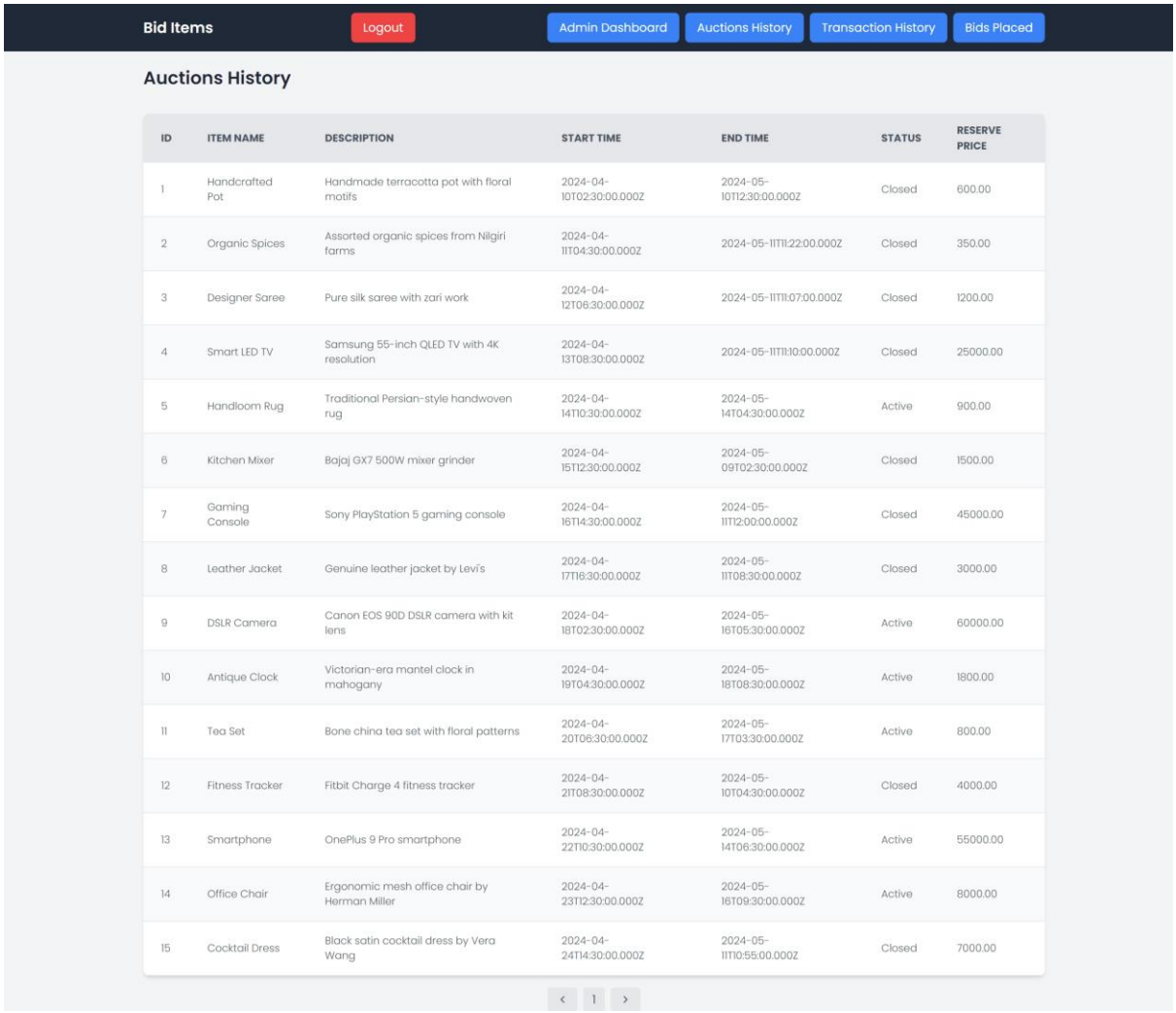


ADMIN PORTAL

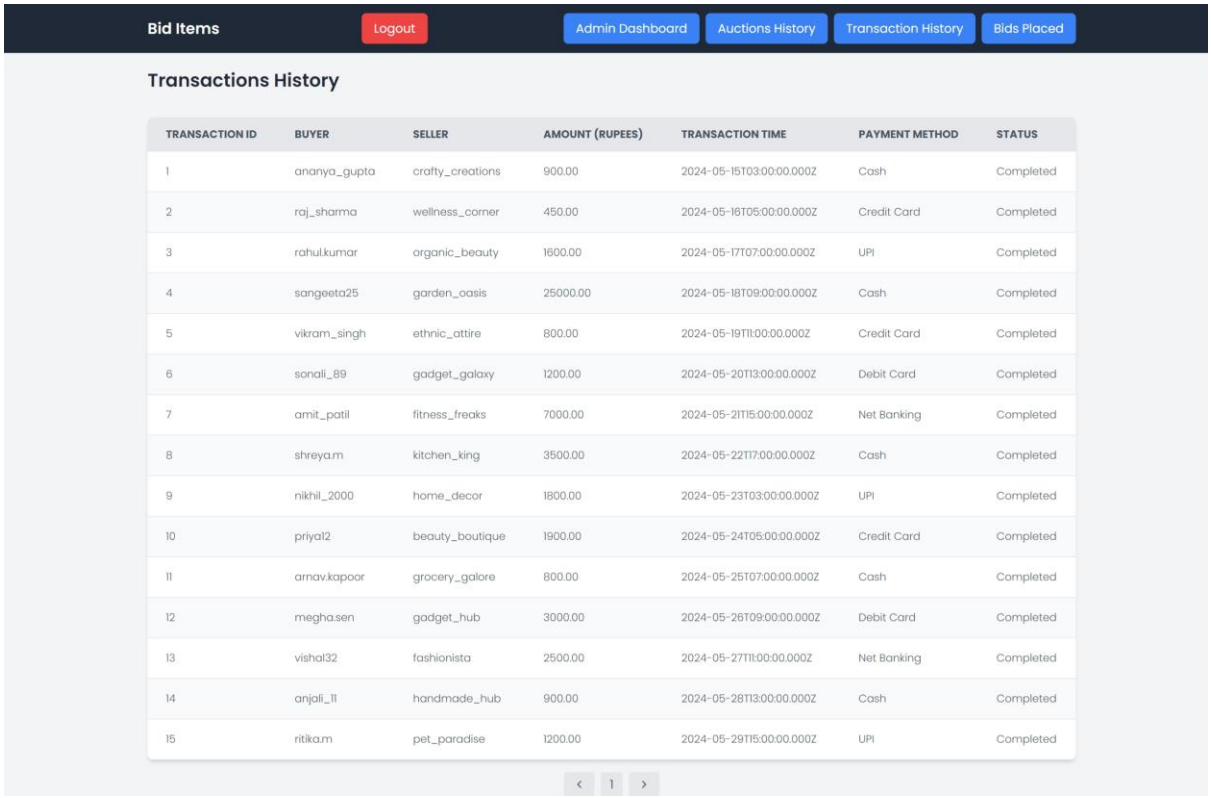
ADMIN DASHBOARD PAGE:



AUCTIONS HISTORY PAGE:



TRANSACTIONS HISTORY PAGE:



BIDS PLACED PAGE: