TITLE PAGE

Programme name: BSc(hons) Computing

Module title: Advanced Databases and Big Data

Assessment title: Assignment -1

Student name and number: Misba Mubarak Gajra (2117336)

Marking tutor: Mrs. Helen Martin

Date of submission: 10-11-2023

Word Count (Portfolio-2): 2193

Word Count (Portfolio 1- Part-4): 2722

Contents

INTRODUCTION 4
PORTFOLIO COMPONENT 1 5
PART-3 5
Queries for Bolton Auction 5
Query A 5
Query B 10
Query C 15
Query D 19
Query E 25
Query F (A) 31
Query F (B) 35
Part 4 39
Comparison and Justification to SQL and NoSQL databases 39
Database Design, Structure and Modelling 39
MySQL: 39
Creating the SQL Database: 39
MongoDB: 46
Creating the Collections and Validating the Respective Schema 47
Efficiency and Scalability 60
SQL: 60
NoSQL: 60
Data Consistency and Integrity 61
SQL: 61
NoSQL: 61
Flexibility and Evolution of Schemas 62
SQL: 62
NoSQL 62
Comparison based on Use Cases: 63
SQL: 63
NoSQL: 63
Comparison based on the Query Operations: 64
SQL: 64
NoSQL: 64
Limitations of SQL: 65

Conclusion **67**

(PORTFOLIO COMPONENT 2) 68

EXPLORING CURRENT AND EMERGING DATABASE TECHNOLOGIES FOR FUTURE INFORMATION SYSTEMS **68**

Creating Landscape for database technologies **68**

New and Emerging Database technologies **69**

Evaluation of Database Technologies for Bolton Auction House **70**

Recommendation for Bolton Auction House **71**

Analysis regarding the approach **73**

Model Flexibility for Data **75**

Multiple-Model Method **75**

Key-features of Arango DB **76**

ArangoDB for Bolton Auction: **78**

Conclusion **80**

REFERENCES 81

APPENDIX 1 – Part 1 MySQL Code 83

APPENDIX 2 - Part 2 MongoDB Code 100

INTRODUCTION

In order to achieve its goals of growth and expansion, Bolton Auction House must establish an information management system that is more flexible and efficient. Working as a junior database developer at Bolton Software Ltd. granted me the chance to see the difficulties that Bolton Auction House faces. At the moment, the auction house manages critical auction data manually using spreadsheets and paper forms. But when the company tries to meet the rising demand for increased auction operations and even online auction services, the shortcomings of this strategy is becoming more critical.

The company is currently working with Bolton Software Ltd. to update its data management system and make the switch to a digital auction platform. In order to achieve its goals of growth and expansion, Bolton Auction House will require an information management system that is more flexible and efficient.

Working as a Junior database developer, my objective is to present a comprehensive comparison and contrast of the Relational Database Management System (RDBMS) and the Non-Relational Database System (NoSQL), two different database models. Using these models for developing prototype databases, the following report attempts to evaluate each system's feasibility and viability in relation to Bolton Auction House's particular requirements. In addition, the research aims to assess these databases' performance, scalability, and adaptability in relation to the company's future growth and expansion goals.

PORTFOLIO COMPONENT 1

PART-3

Queries for Bolton Auction

The reference and guidance for MySQL is according to (Valade, 2004) and (w3schools, 2023). Whereas, for mongoDB it is from (mongodb, 2023).

Query A

Write a query to show any lots which contain items which are Toys, you must display the lot description and the date and location of the auction.

For MySql:

The query that is given finds details about lots that contain toys in their Items. From the item, lot, and auction tables, it obtains the LotNumber, ItemDescription, lotDescription, AuctionDate, and Location. When the items included in a lot are identified as toys, the query retrieves specific information from the lot, auction, and item columns and delivers results that show the appropriate lot description, auction date, and location.

SELECT Clause:

- i.LotNumber: This function gets the item table's lot number.
- i.ItemDescription: From the item table, the item description is retrieved.
- l.lotDescription: From the lot table, this function retrieves the lot description.
- AuctionDate: This function pulls the auction date from the table of auctions.
- Location: This method pulls the location out of the auction table.

FROM CLAUSE:

- item i: Uses an alias of i to specify the item table.
- Using a matching LotNumber, the INNER JOIN lot l joins the item and lot tables.
- Using matching auctionID values, an inner join (auction a) joins the auction table and the outcome of the preceding join.

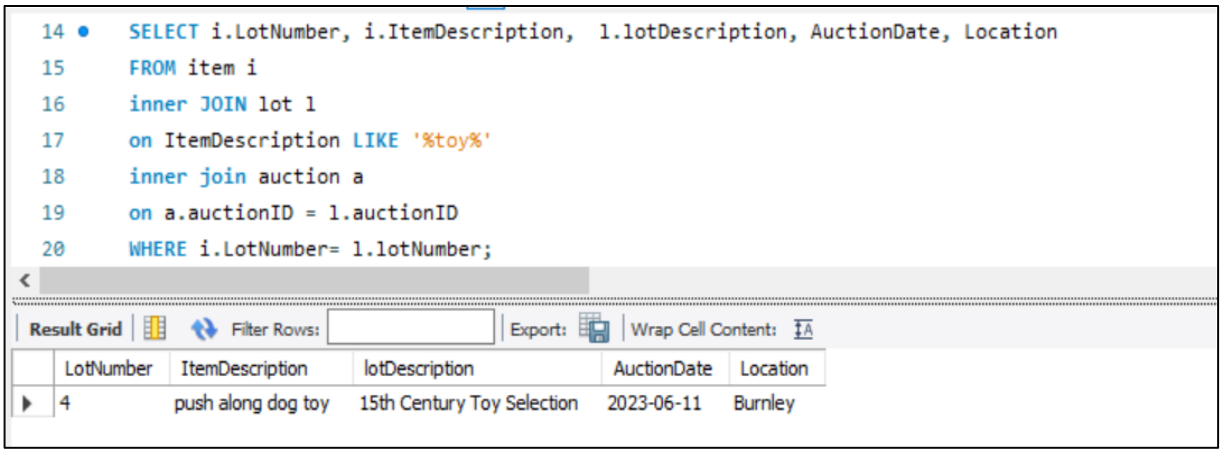
ON CLAUSE:

- ItemsDescription LIKE '%toy%' indicates that the word 'toy' must appear in the item table's ItemsDescription.
- The formula "a.auctionID = l.auctionID" indicates that the lot table's auctionID and the auction table's auctionID must match.

WHERE Section:

- Only rows where the LotNumber in the item table and the LotNumber in the lot table are included in the results are filtered using the expression i.LotNumber= l.lotNumber.

(SQL query A)



```
SELECT i.LotNumber, i.ItemDescription,  l.lotDescription, AuctionDate, Location
FROM item i
inner JOIN lot l
on ItemDescription LIKE '%toy%'
inner join auction a
on a.auctionID = l.auctionID
WHERE i.LotNumber= l.lotNumber;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

LotNumber	ItemDescription	lotDescription	AuctionDate	Location
4	push along dog toy	15th Century Toy Selection	2023-06-11	Burnley

For MongoDB:

The goal of this MongoDB query is to locate lots containing "Toys"-classified goods. To unwind the arrays and meet the predetermined requirements, it makes use of an aggregation pipeline with multiple phases.

$Unwind **Lot**

Creates a new document for each element in the array by unwinding the "Lot" array.

$Unwind Item

To build a new document for each item, unwinds the "Items" array within each lot.

$match Stage

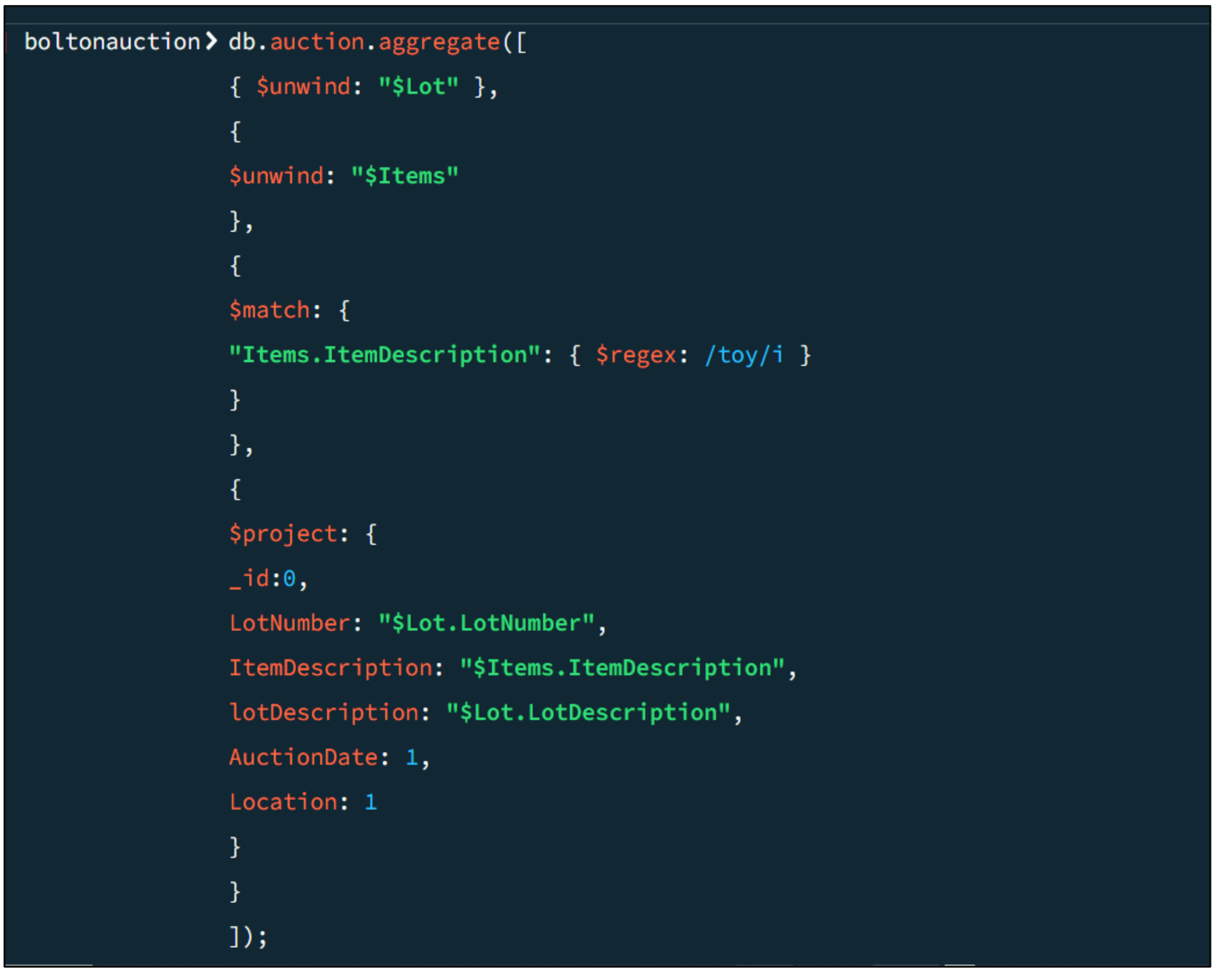
Identify those items that have the word "toy" in the "ItemDescription" field.

$project Stage

Transforms the output, adding only those fields to each matched document (such as "LotNumber," "ItemDescription," "LotDescription," "AuctionDate," and "Location") and removing the default "_id" field.

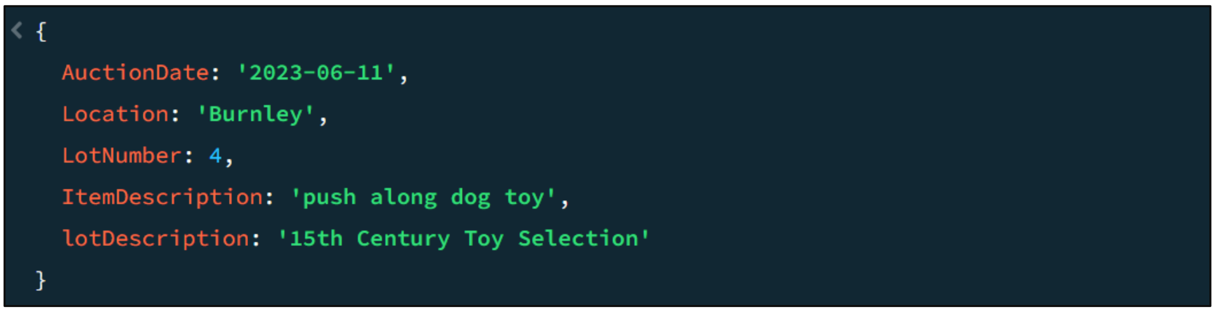
The documents are filtered by the $regex operator if the word "toy" appears in the "ItemDescription" field. The results will include any document in which the phrase "toy" appears in the "ItemDescription" (in any situation).

(Mongodb Query A)



```
boltonauction> db.auction.aggregate([
               { $unwind: "$Lot" },
               {
               $unwind: "$Items"
               },
               {
               $match: {
               "Items.ItemDescription": { $regex: /toy/i }
               }
               },
               {
               $project: {
               _id:0,
               LotNumber: "$Lot.LotNumber",
               ItemDescription: "$Items.ItemDescription",
               lotDescription: "$Lot.LotDescription",
               AuctionDate: 1,
               Location: 1
               }
               }
               ]);
```

Output:


```
{
    AuctionDate: '2023-06-11',
    Location: 'Burnley',
    LotNumber: 4,
    ItemDescription: 'push along dog toy',
    lotDescription: '15th Century Toy Selection'
  }
```

Query Comparison:

- Both queries aim to retrieve data related to lots containing items described as “Toy”.
- The SQL query retrieves data using an INNER JOIN whereas the MongoDB query manipulates data using “$unwind” to filter the data from the “Lot” array and further unwinds the “Item” array to match the requirement.

Query B

Write a query to show seller name, telephone number, lot description and Reserve Price, where they have a lot where the reserve price is more than £90 but less than £150 and only being auctioned in Manchester.

For SQL:

The query fetches the specific information about the sellers and their respective lots that meets the criteria for the Reserve Price and Location.

SELECT Clause:

- s.sellerName: This function gets the seller's table sellerName.
- s.sellerTelephone: From the seller table, the seller telephone is retrieved.
- l.lotDescription: From the lot table, this function retrieves the lot description.
- l.ReservePrice: From the lot table, this function retrieves the reserve price.
- a.Location: This method pulls the location out of the auction table.

FROM Clause:

- seller s: Uses an alias of s to specify the seller table.
- Using matching sellerID values, the inner join lot l joins the seller and lot tables.
- Using matching AuctionID values, an inner join (auction a) joins the auction table and the outcome of the preceding join.

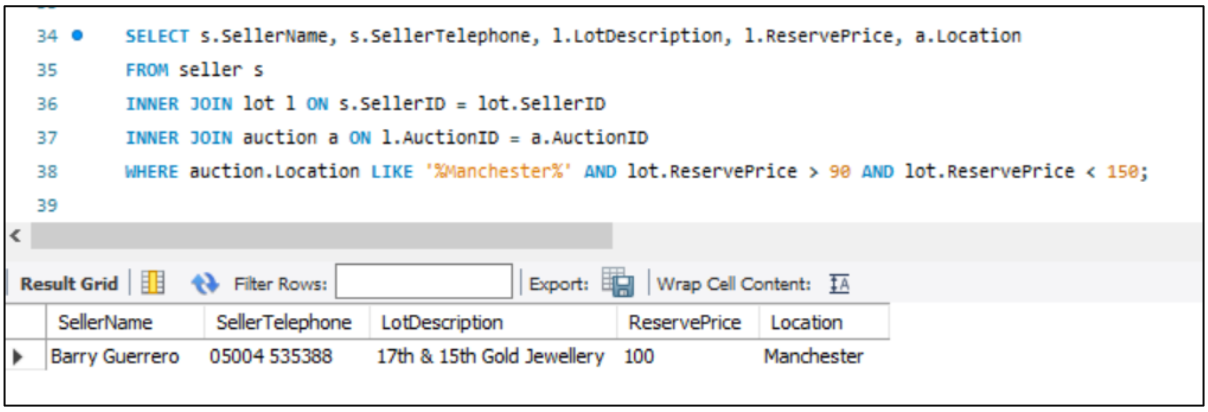
ON Clause:

- The expression “s.sellerID = l.sellerID” indicates that the sellerIDs in the lot and seller tables must match.
- The expression “l.AuctionID = l.AuctionID” indicates that the lot table's AuctionID and the auction table's AuctionID must match.

WHERE Section:

- a.location LIKE "%Manchester%": This filters the results to show only those records where the word "Manchester" appears in the auction table's location.
- l.ReservePrice > 90 AND l.ReservePrice < 150: This filter selects records in the lot table where the reserve price is more than £90 but less than £150.

(SQL query A)



```
SELECT s.SellerName, s.SellerTelephone, l.LotDescription, l.ReservePrice, a.Location
FROM seller s
INNER JOIN lot l ON s.SellerID = lot.SellerID
INNER JOIN auction a ON l.AuctionID = a.AuctionID
WHERE auction.Location LIKE '%Manchester%' AND lot.ReservePrice > 90 AND lot.ReservePrice < 150;
```

SellerName	SellerTelephone	LotDescription	ReservePrice	Location
Barry Guerrero	05004 535388	17th & 15th Gold Jewellery	100	Manchester

For MongoDB:

$unwind Stage:

For every element, a new document is created by splitting the "Lot" array and "seller" collection.

$match Stage:

Applying the predetermined criteria, the collection's documents are filtered. In this instance, it filters the documents based on two criteria: both the "Location" field should be set to "Manchester" and the "ReservePrice" field of the "Lot" sub-document must be larger than 90 and less than 150.

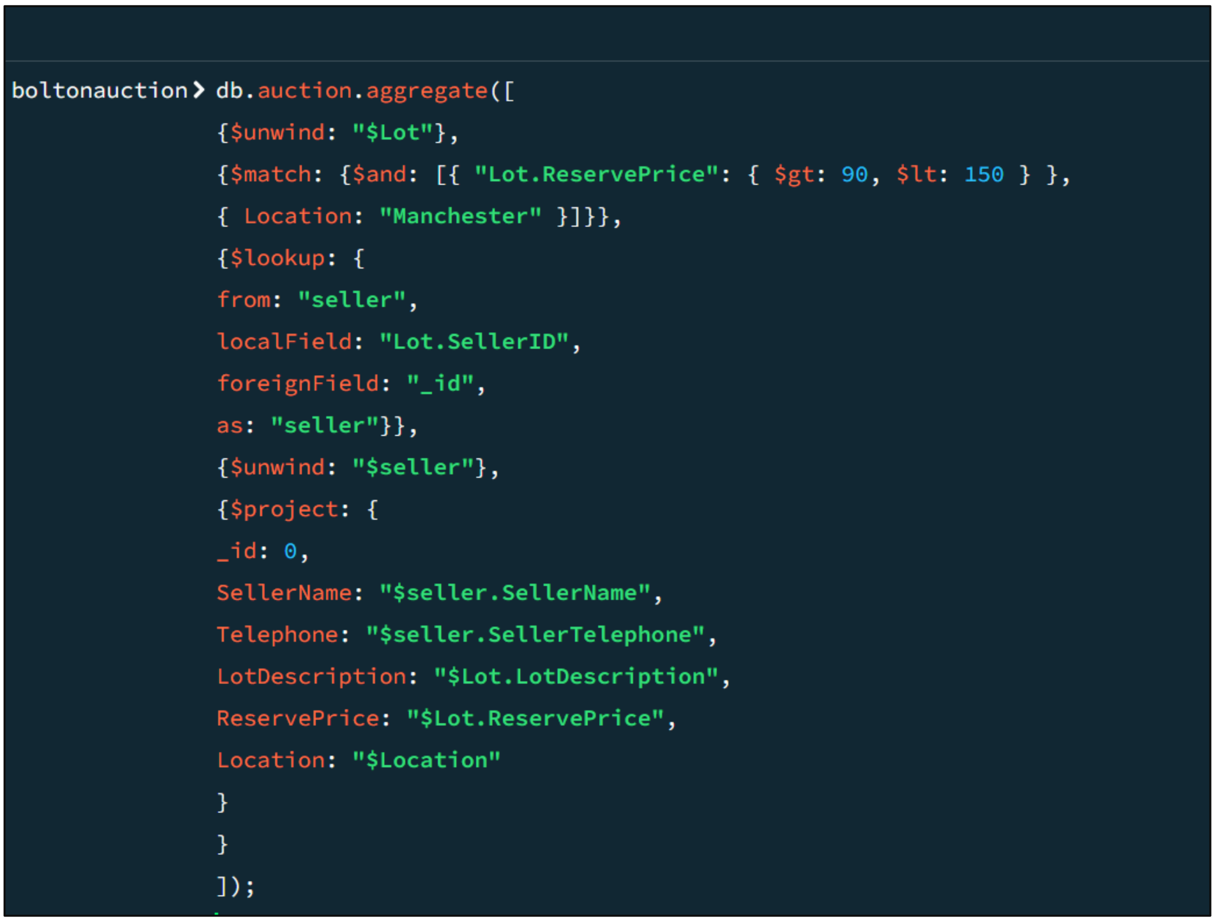
$lookup Stage:

When the value of the "_id" field in the "seller" collection and the "Lot.SellerID" field in the "auction" collection matches, documents from the "seller" collection are retrieved. An array containing the results is kept in the "seller" field.

$project Stage:

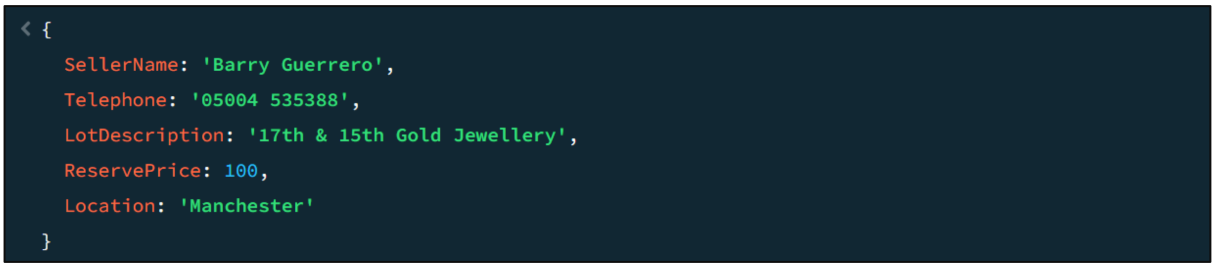
It does not contain the default "_id" column but does contain the "SellerName," "SellerTelephone," "LotDescription," "ReservePrice," and "Location" fields.

(MongoDB query B)



```
boltonauction> db.auction.aggregate([
               {$unwind: "$Lot"},
               {$match: {$and: [{ "Lot.ReservePrice": { $gt: 90, $lt: 150 } },
               { Location: "Manchester" }]}},
               {$lookup: {
               from: "seller",
               localField: "Lot.SellerID",
               foreignField: "_id",
               as: "seller"}},
               {$unwind: "$seller"},
               {$project: {
               _id: 0,
               SellerName: "$seller.SellerName",
               Telephone: "$seller.SellerTelephone",
               LotDescription: "$Lot.LotDescription",
               ReservePrice: "$Lot.ReservePrice",
               Location: "$Location"
               }
               }
               ]);
```

Output:



```
< {
    SellerName: 'Barry Guerrero',
    Telephone: '05004 535388',
    LotDescription: '17th & 15th Gold Jewellery',
    ReservePrice: 100,
    Location: 'Manchester'
  }
```

Query Comparison:

- Both queries aim to retrieve data related to seller, lots and auctions based upon the requirement.
- The SQL query uses “LIKE” and “WHERE” operator whereas NoSQL query uses “$match” and “$and” operators to fulfil the requirement.

- The SQL query retrieves data using an INNER JOIN whereas the MongoDB query manipulates data using “$lookup” to filter the data.

Query C

Write a query to show which customer has paid the highest total price for all successful bids, you should display the bidders name which should be labelled "Bidders Name" and the total price, which should be named as "Total Price".

For SQL:

This purpose of this query is to identify the bidder who has made all of their successful bids total the highest amount. The top record, which indicates the bidder with the highest total price, is retrieved by the query after it has summed the winning prices for each bidder, grouped the results by bidder, and sorted them in descending order of total price. The bidder's name and the associated total price are retrieved by the query.

The above SQL query extracts data from the bidder and lotSale tables.

SELECT Clause:

- b. Name: Fetches the bidder’s name column.
- sum(ls.WinningPrice) AS Total_Price: This renames the result as Total_price and computes the sum of the WinningPrice column from the lotSale database.

FROM Clause:

FROM boltonauction.Bidder b: With an alias of b, this indicates the Bidder table from the Bolton Auction database.

INNER JOIN Clause:

This uses the BidderID column which is included in both tables to link the Bidder and lotSale tables.

GROUP BY Clause:

The lotSale table's BidderID column is used to group the results.

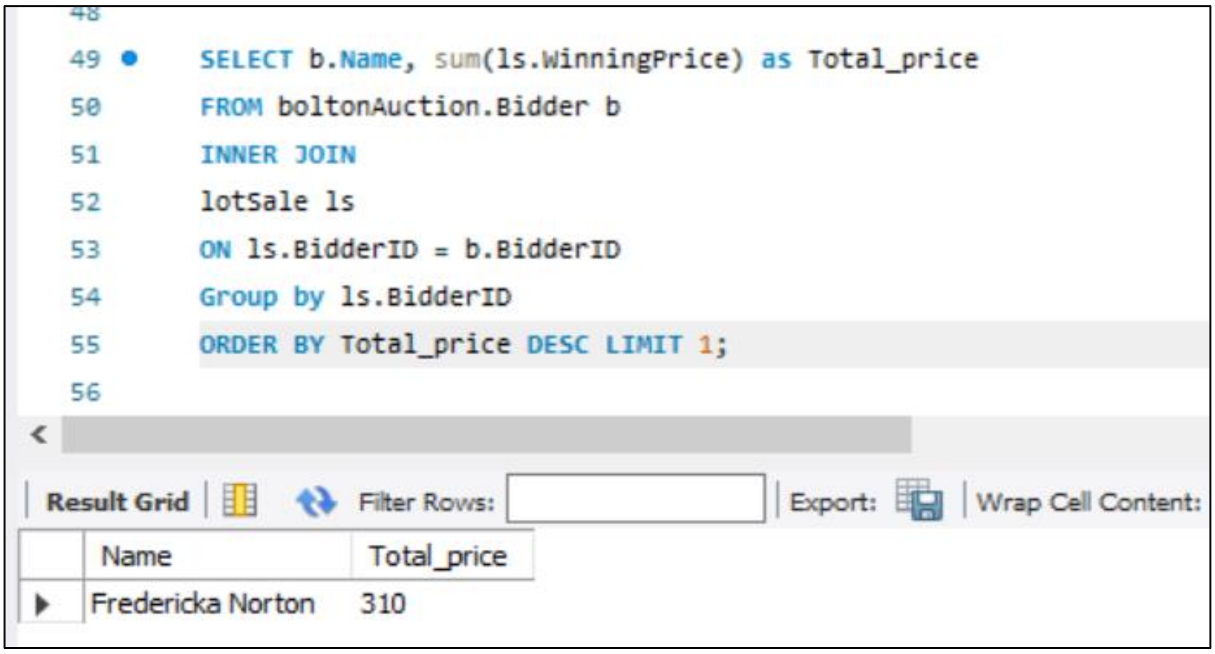
ORDER BY Clause:

Order by Total_price DESC: This places the results in descending order according to the Total_price column.

LIMIT Clause:

LIMIT 1: This ensures that just the record with the highest total price is obtained by limiting the number of rows the query can return to only one row.

(SQL query C)



```
SELECT b.Name, sum(ls.WinningPrice) as Total_price
FROM boltonAuction.Bidder b
INNER JOIN
lotSale ls
ON ls.BidderID = b.BidderID
Group by ls.BidderID
ORDER BY Total_price DESC LIMIT 1;
```

Name	Total_price
Fredericka Norton	310

For MongoDB:

This MongoDB query alters the "lot sale" collection using the aggregation pipeline framework.

$group Stage:

- Uses the **$sum** operator to total the "WinningPrice" field after grouping the input documents based on the "BidderID" field.
- To ensure precise summation, the "WinningPrice" is converted to a double number using the **$toDouble** operator.
- The field labelled "Total_Winning_Price" contains the results.

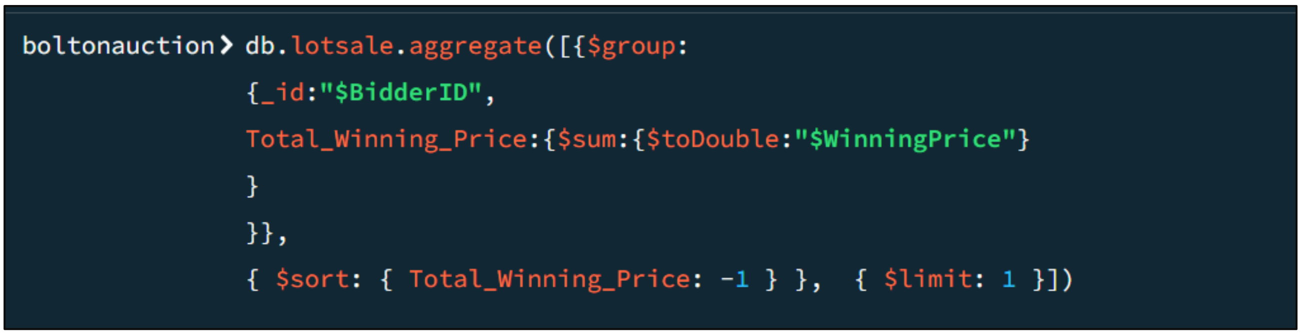
$sort Stage:

- The **"Total_Winning_Price"** parameter is used to determine the order in which the grouped data is sorted in the **$sort** Stage.
- This guarantees that the documents are arranged in descending order of total winning price.

$limit Stage:

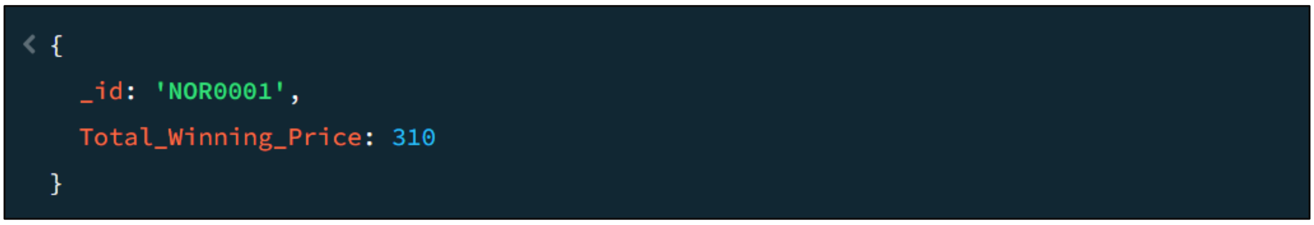
- Restricts the output's document count to only one.
- Consequently, only the document that has the largest cumulative winning price is given back.

(MongoDB query C)



```
boltonauction> db.lotsale.aggregate([{$group:
                {_id:"$BidderID",
                Total_Winning_Price:{$sum:{$toDouble:"$WinningPrice"}
                }
                }},
                { $sort: { Total_Winning_Price: -1 } },  { $limit: 1 }])
```

Output:



```
< {
    _id: 'NOR0001',
    Total_Winning_Price: 310
  }
```

Query Comparison:

- Both queries aim to retrieve data related to bidder who has paid the highest total price for successful bids.
- The SQL query retrieves data by joining the tables “Bidder” and “LotSale” whereas the MongoDB query performs data aggregation on the “LotSale” collection.
- The SQL query manipulates the data using “$sum” operator whereas the MongoDB query groups the data to calculate the highest price.
- The similarity is that both the query restricts the output to only the top result with the highest total price using “LIMIT 1” and “$limit” operator respectively.

Query D

Write a query to show the total number of successful bids per customer and their name, rename the total number of bids as "Total Bids"

Each customer's total number of successful bids is retrieved by the query and illustrated with their names next to the corresponding total number of bids. This query offers vital information about each bidder's bidding activity inside the auction system. The specific query retrieves the specified data from lotSale and bidder's database table.

SELECT Clause:

- Specific data can be retrieved from the database using the SELECT query.
- b.Name AS "Customer Name": To enhance readability in the output, this obtains the Name column from the Bidder database and renames it as "Customer Name."
- COUNT(ls.BidderID) as Total_Bids: This assigns "Total Bids" to the count after counting every instance of ls.BidderID.

FROM Clause:

- FROM boltonauction.Bidder b: With an alias of b, this represents the Bidder table from the Bolton Auction database.

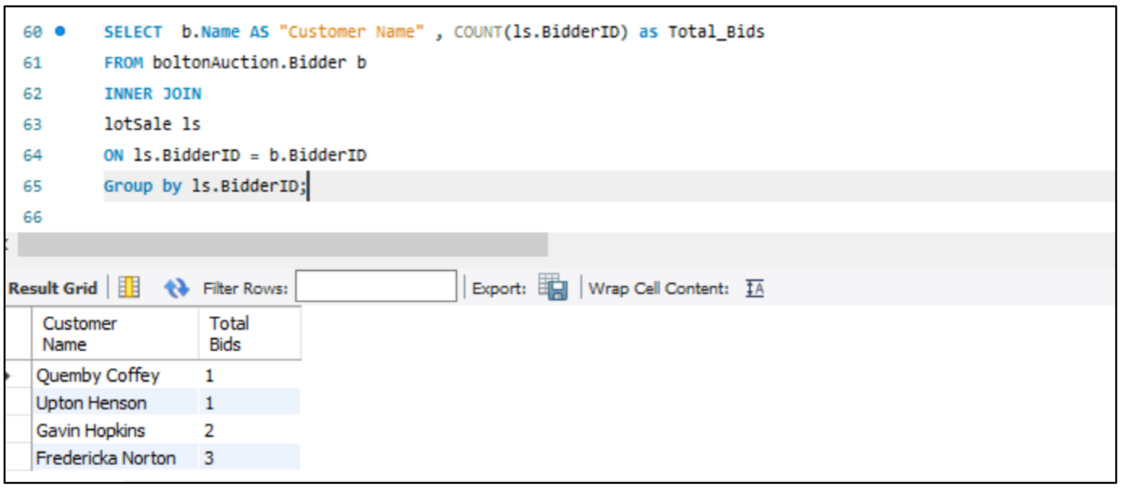
INNER JOIN Clause:

- This uses the BidderID column which is included in both tables to link the Bidder and lotSale tables.

GROUP BY Section:

- GROUP BY ls.BidderID: The lotSale table's BidderID column is used to group the results.

(SQL query D)



```
SELECT  b.Name AS "Customer Name" , COUNT(ls.BidderID) as Total_Bids
FROM boltonAuction.Bidder b
INNER JOIN
lotSale ls
ON ls.BidderID = b.BidderID
Group by ls.BidderID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Customer Name	Total Bids
Quemby Coffey	1
Upton Henson	1
Gavin Hopkins	2
Fredericka Norton	3

For MongoDB:

The aggregation pipeline is used by this MongoDB query to carry out operations on the "lotsale" collection. Overall, this query successfully unwinds the "bidder" array, groups the "lotsale" collection by "BidderID," determines the total number of bids for each bidder, looks for additional data from the "bidder" collection, and projects particular fields for the output.

$group Stage:

- Utilising the **$sum** operator, this stage groups the provided documents according to the "BidderID" field and determines the total number of bids made by each bidder.
- The "Total Bids" field contains the result.

$lookup Stage:

- When the value of the "_id" field in one collection matches the value of the "lotsale" collection's "_id" field, documents from the "bidder" collection are retrieved.
- The "bidder" field contains the results that are saved.

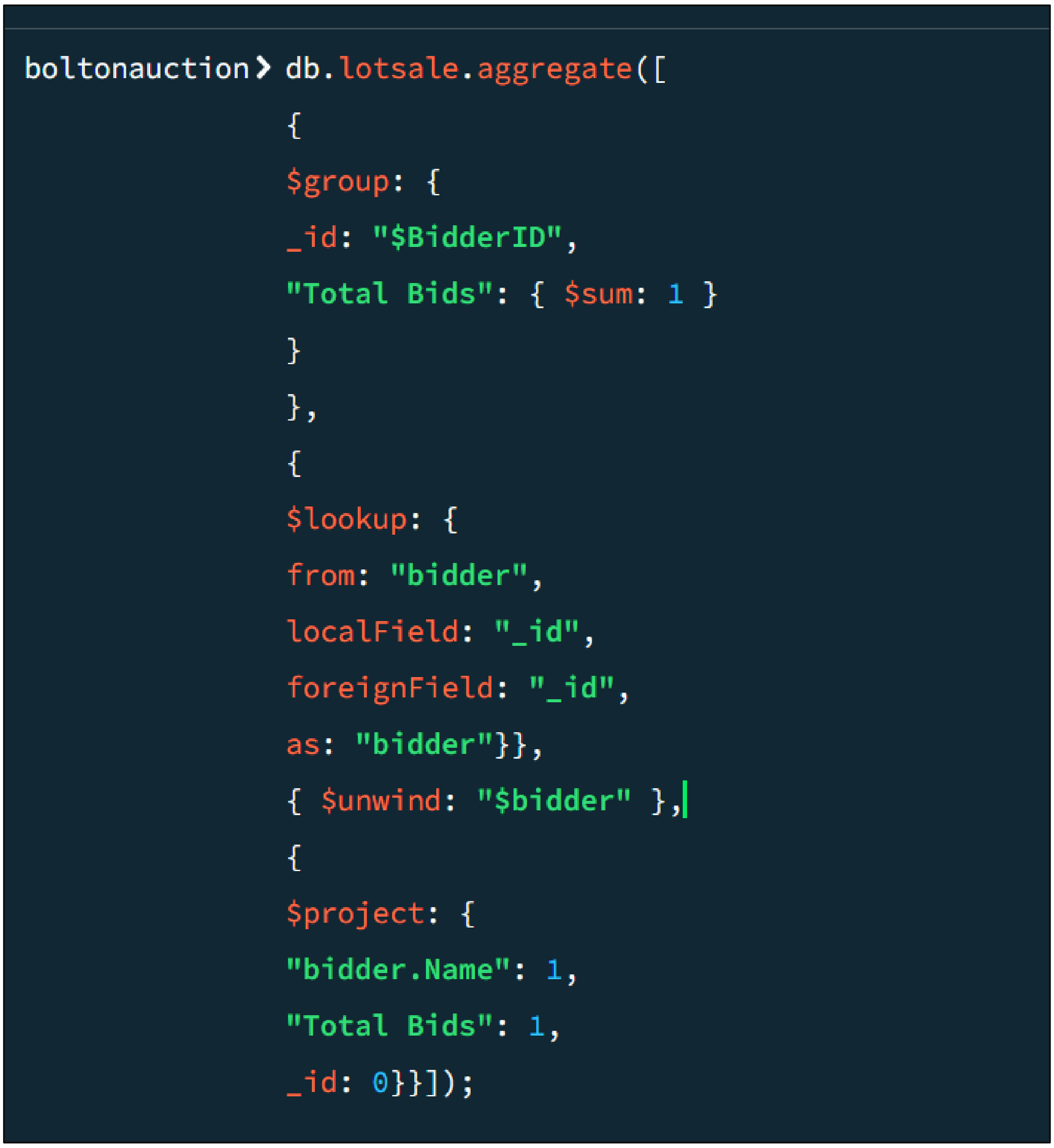
$unwind Stage:

- Every element is represented by a new document as the "bidder" array is split.

$project Stage:

- Here, the default "_id" column is excluded and just the "Name" field from the "bidder" sub-document and the "Total Bids" field are included.

(MongoDB query D)

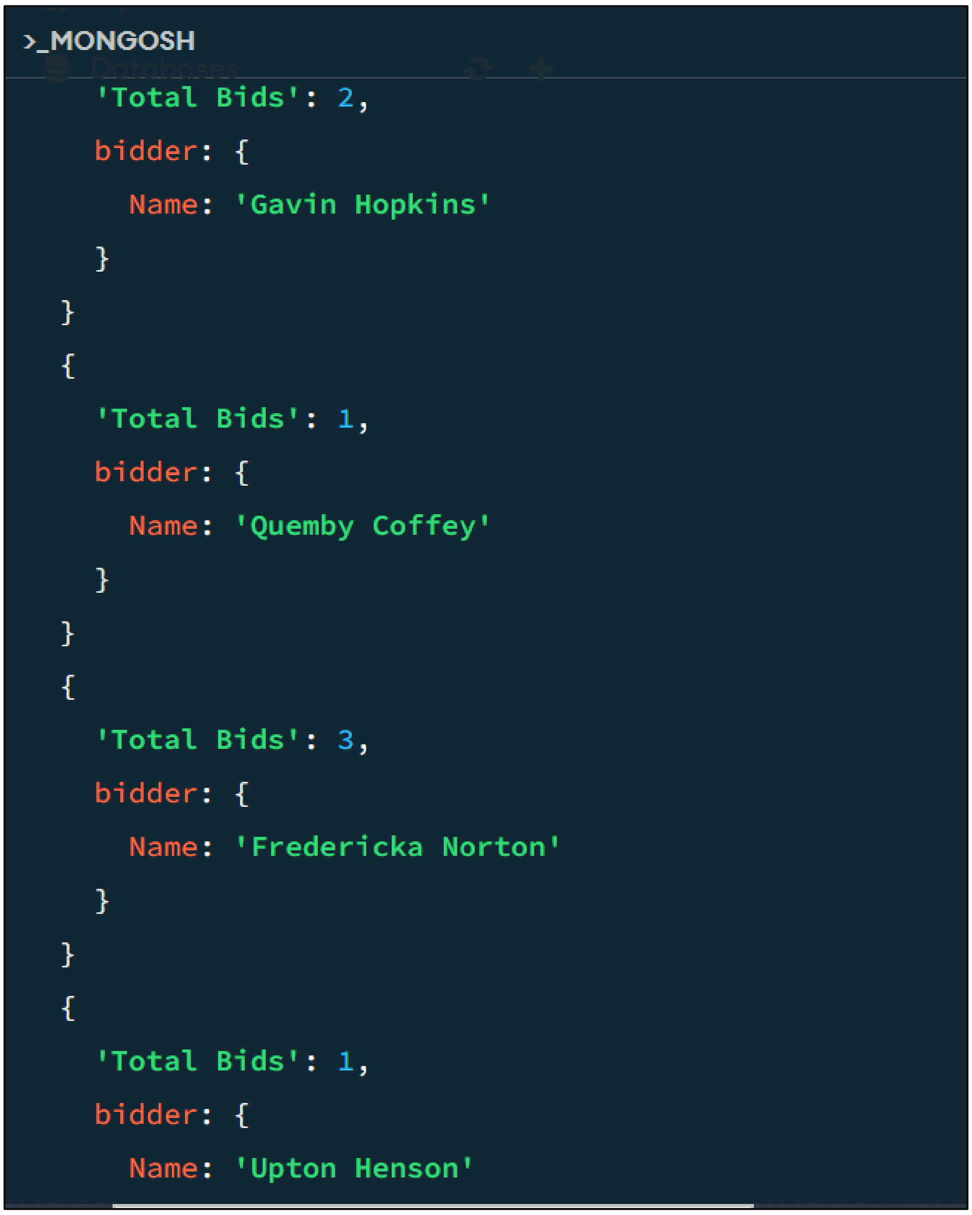


```
boltonauction> db.lotsale.aggregate([
               {
               $group: {
               _id: "$BidderID",
               "Total Bids": { $sum: 1 }
               }
               },
               {
               $lookup: {
               from: "bidder",
               localField: "_id",
               foreignField: "_id",
               as: "bidder"}},
               { $unwind: "$bidder" },
               {
               $project: {
               "bidder.Name": 1,
               "Total Bids": 1,
               _id: 0}}]);
```

Query Comparison:

- Both queries aim to retrieve data related to the total number of successful bids per customer.
- The SQL query retrieves data by joining the tables “Bidder” and “LotSale” whereas the MongoDB query performs data aggregation on “lotsale” collection grouping the “BidderID” to count the total number of bids.
- The SQL query manipulates the data using inner join clause whereas MongoDB uses “$lookup” to filter out the data.

Output:



```
    'Total Bids': 2,
    bidder: {
      Name: 'Gavin Hopkins'
    }
  }
  {
    'Total Bids': 1,
    bidder: {
      Name: 'Quemby Coffey'
    }
  }
  {
    'Total Bids': 3,
    bidder: {
      Name: 'Fredericka Norton'
    }
  }
  {
    'Total Bids': 1,
    bidder: {
      Name: 'Upton Henson'
```

Query E

Write a query to show the lowest reserve price for a seller, rename this column "Lowest Reserve Price" and show the sellers name, rename this column as "Seller Name".

For SQL:

SELECT CLAUSE:

- min(l.ReservePrice) AS Lowest_Reserve_Price: This produces "Lowest Reserve Price" in the output after determining the minimum value of the ReservePrice column from the lot table.
- s.SellerName AS "Seller Name": This causes the output to rename the SellerName column as "Seller Name" after retrieving it from the seller table.
- min(l.ReservePrice) AS Lowest_Reserve_Price: This produces "Lowest Reserve Price" in the output after determining the minimum value of the ReservePrice column from the lot table.

FROM Clause:

The table seller specified here.

JOIN Clause:

INNER JOIN lot l on l.sellerID = s.sellerID - Based on the sellerID field, which is shared by both columns, this links the seller table with the lot table.

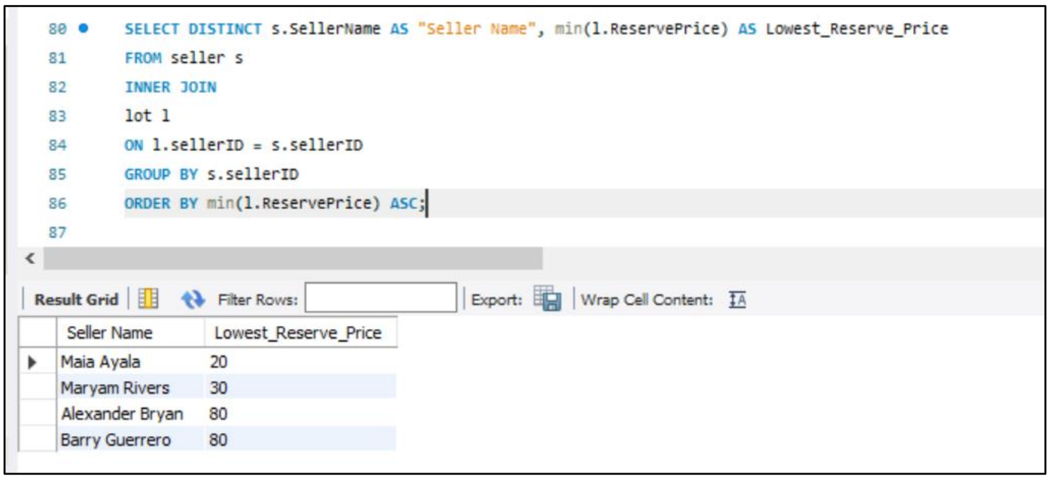
COLUMN BY Section:

GROUP BY s.sellerID : The sellerID column in the seller table is used to group the results.

ORDER BY Clause:

Min(l.ReservePrice) ASC : This displays the outcomes in ascending order based on the minimum reserve price.

(SQL query E)



```
SELECT DISTINCT s.SellerName AS "Seller Name", min(l.ReservePrice) AS Lowest_Reserve_Price
FROM seller s
INNER JOIN
lot l
ON l.sellerID = s.sellerID
GROUP BY s.sellerID
ORDER BY min(l.ReservePrice) ASC;
```

Seller Name	Lowest_Reserve_Price
Maia Ayala	20
Maryam Rivers	30
Alexander Bryan	80
Barry Guerrero	80

For MongoDB:

$unwind:

This step generates a new document for each element of the array by unwinding the 'Lot' array in each document.

$group:

Using the **$min** operator, this stage groups the documents according to the 'SellerID' from the 'Lot' field in order to determine the minimum reserve price for each seller.

$lookup:

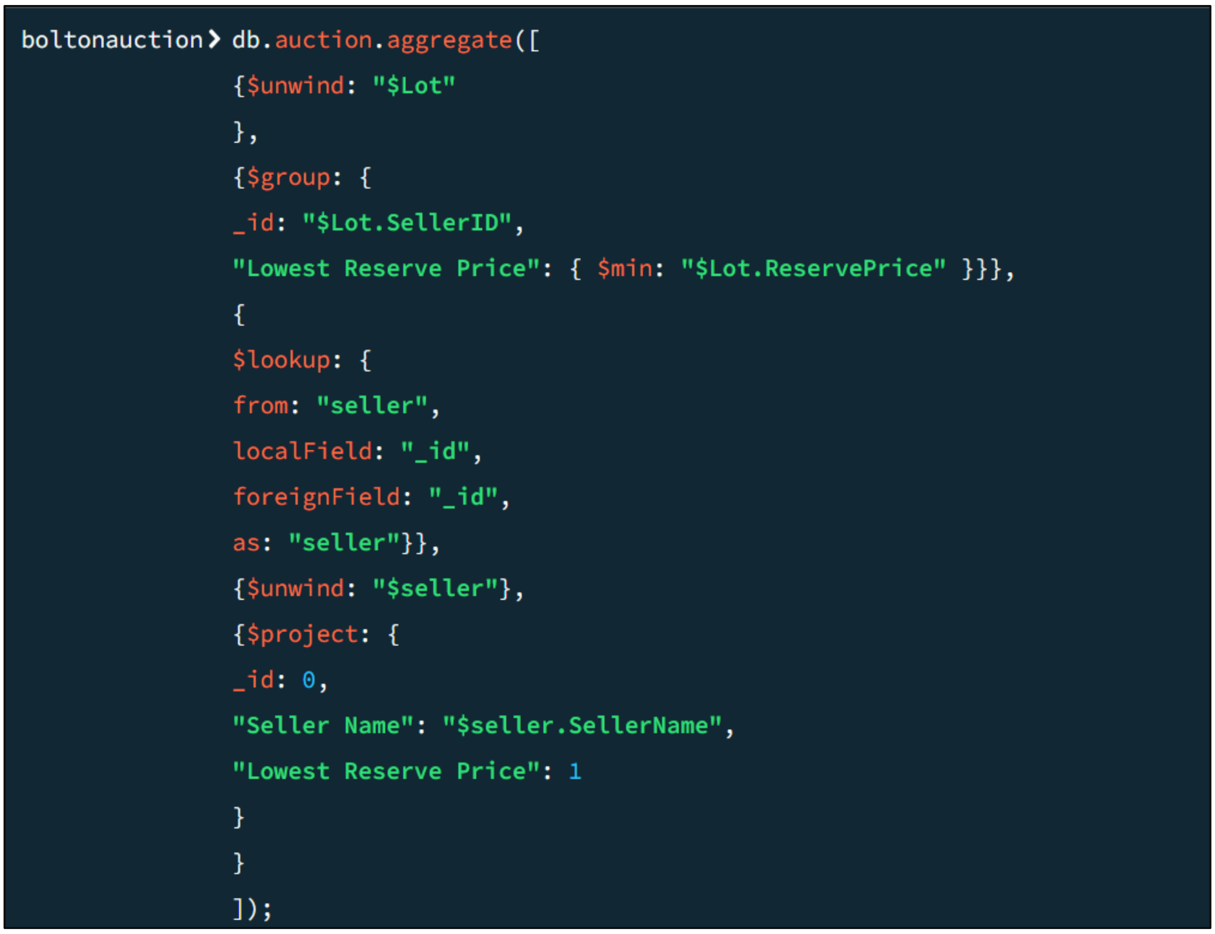
Using the '_id' column as a guide, this stage looks for the seller's details in the 'seller' collection.

$unwind:

In this step, the 'seller' array is unraveled resulting in each output document containing data from the corresponding seller.

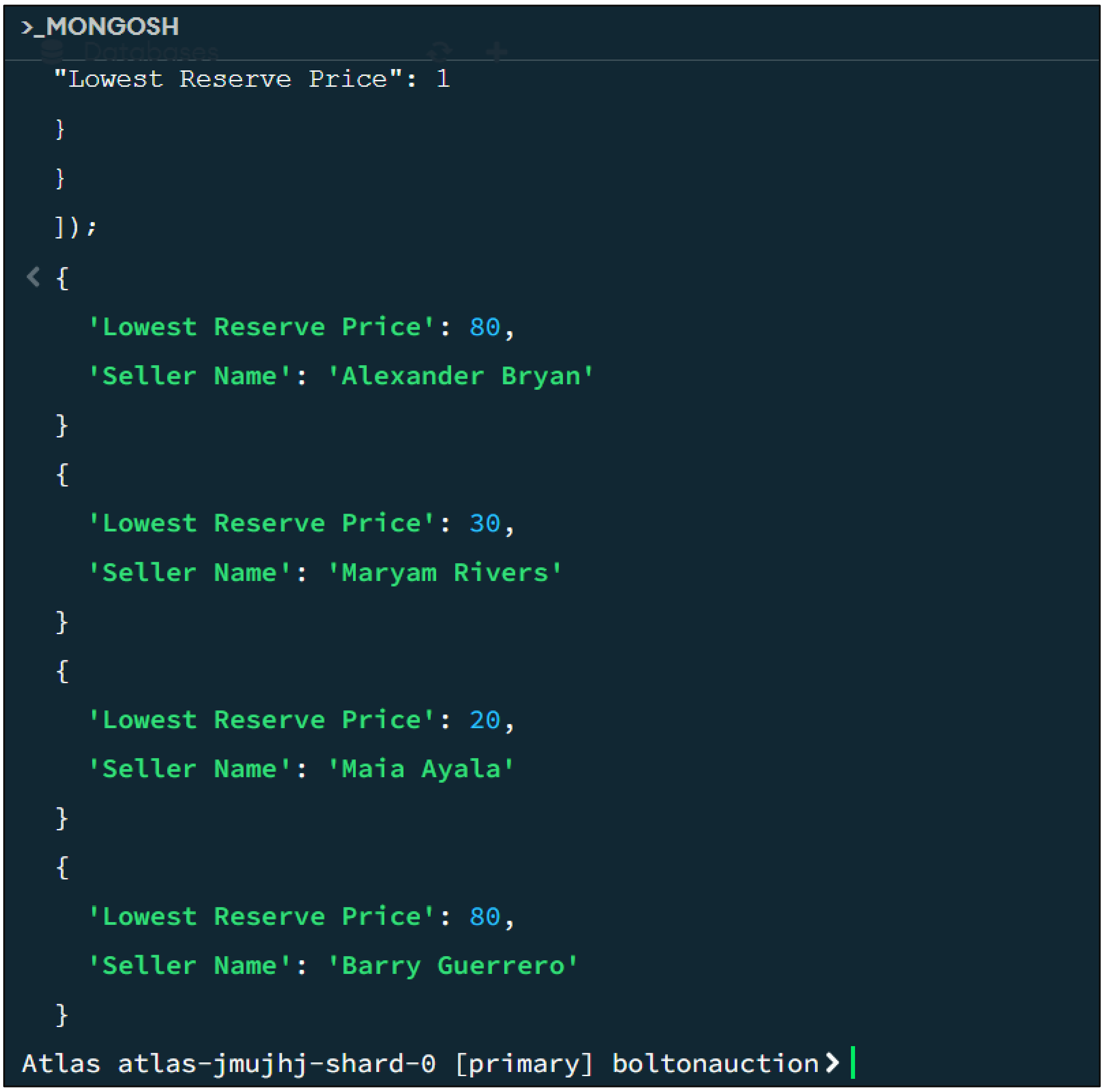
$project: The default '_id' field is removed in this instance, and the fields 'Seller Name' and 'Lowest Reserve Price' are renamed for convenience.

(MongoDB query E)



```
boltonauction> db.auction.aggregate([
               {$unwind: "$Lot"
               },
               {$group: {
               _id: "$Lot.SellerID",
               "Lowest Reserve Price": { $min: "$Lot.ReservePrice" }}},
               {
               $lookup: {
               from: "seller",
               localField: "_id",
               foreignField: "_id",
               as: "seller"}},
               {$unwind: "$seller"},
               {$project: {
               _id: 0,
               "Seller Name": "$seller.SellerName",
               "Lowest Reserve Price": 1
               }
               }
               ]);
```

Output:



```
>_MONGOSH
  "Lowest Reserve Price": 1
  }
  }
  ]);
< {
    'Lowest Reserve Price': 80,
    'Seller Name': 'Alexander Bryan'
  }
  {
    'Lowest Reserve Price': 30,
    'Seller Name': 'Maryam Rivers'
  }
  {
    'Lowest Reserve Price': 20,
    'Seller Name': 'Maia Ayala'
  }
  {
    'Lowest Reserve Price': 80,
    'Seller Name': 'Barry Guerrero'
  }
Atlas atlas-jmujhj-shard-0 [primary] boltonauction>
```

Query Comparison:

- Both queries aim to retrieve data to the lowest reserve price for each seller.
- The SQL query retrieves data by joining the tables “Seller” and “Lot” table and uses “$min” operator to find the minimum price.
- The MongoDB query utilises “$group” and “$min” operator to calculate the lowest reserve price.
- Thus, both queries can be filtered out using same logic.

Query F (A)

Query to fetch lots with “Toy” in their description and return the auction date and location.

For MySQL:

SELECT Clause:

- **l.lotDescription**: From the lot table, this function retrieves the lot description.
- **AuctionDate**: This function pulls the auction date from the table of auctions.
- **Location**: This method pulls the location out of the auction table.

FROM CLAUSE:

- Using a matching LotNumber, the INNER JOIN lot l joins the auction and lot tables.

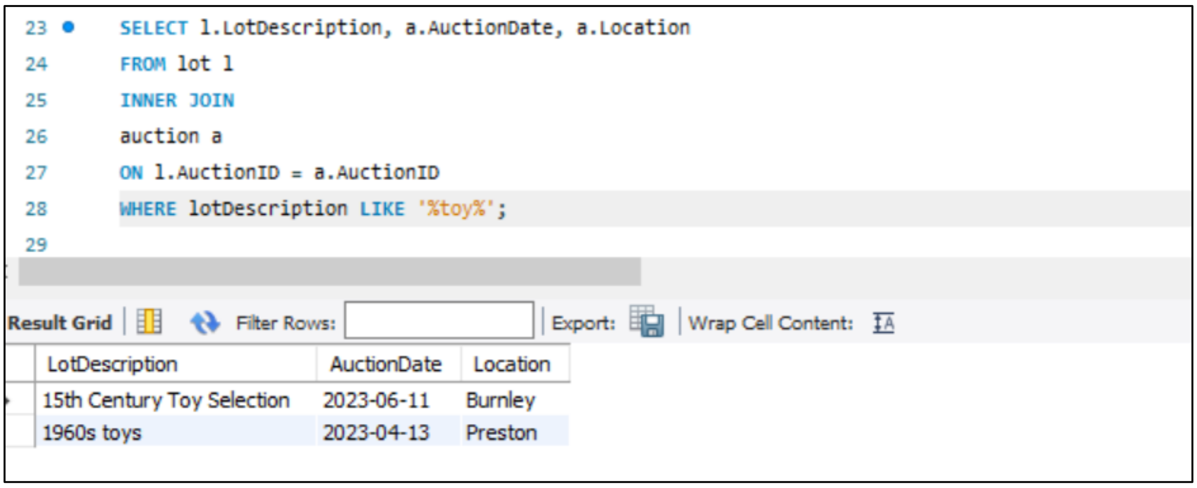
ON CLAUSE:

- The formula “a.auctionID = l.auctionID” indicates that the lot table's auctionID and the auction table's auctionID must match.

WHERE Section:

- lotDescription LIKE '%toy%' indicates that the word 'toy' must appear in the item table's ItemsDescription.

(SQL query F.a)



```
SELECT l.LotDescription, a.AuctionDate, a.Location
FROM lot l
INNER JOIN
auction a
ON l.AuctionID = a.AuctionID
WHERE lotDescription LIKE '%toy%';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

LotDescription	AuctionDate	Location
15th Century Toy Selection	2023-06-11	Burnley
1960s toys	2023-04-13	Preston

For Mongo DB:

$Unwind **Lot**

Creates a new document for each element in the array by unwinding the "Lot" array from Auction Collection

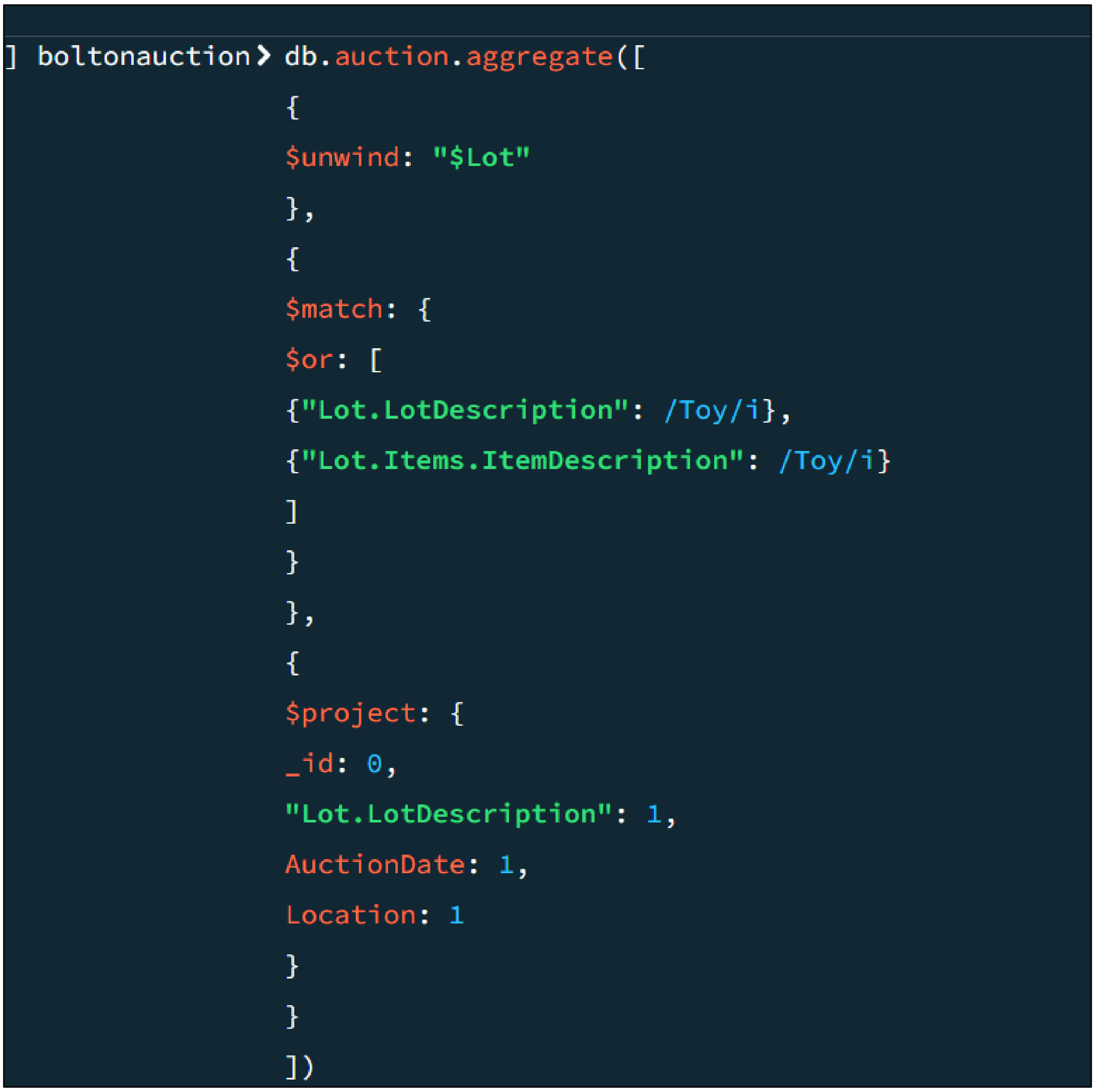
$match Stage:

Identify those lots that have the word "toy" in the lot description field.

$project Stage

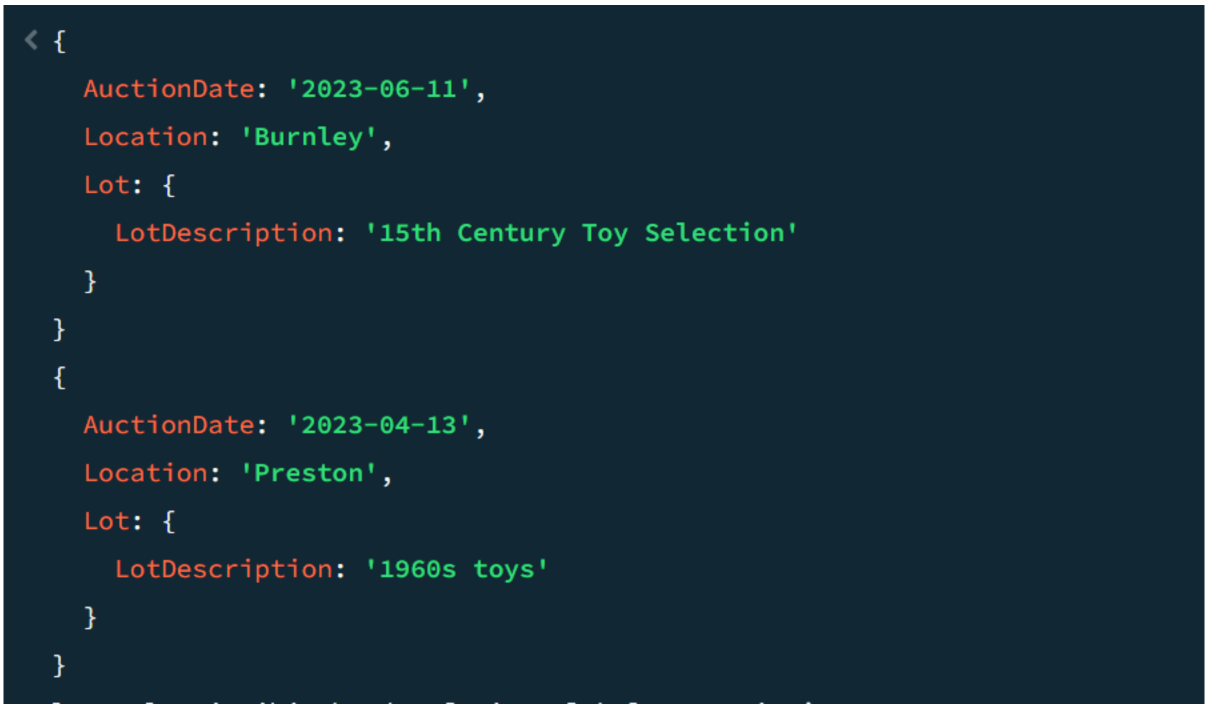
Transforms the output, adding only those fields to each matched document (such as "LotDescription," "AuctionDate," and "Location") and removing the default "_id" field.

(Mongodb query F.a)



```
] boltonauction> db.auction.aggregate([
                 {
                 $unwind: "$Lot"
                 },
                 {
                 $match: {
                 $or: [
                 {"Lot.LotDescription": /Toy/i},
                 {"Lot.Items.ItemDescription": /Toy/i}
                 ]
                 }
                 },
                 {
                 $project: {
                 _id: 0,
                 "Lot.LotDescription": 1,
                 AuctionDate: 1,
                 Location: 1
                 }
                 }
                 ])
```

Output:


```
{
  AuctionDate: '2023-06-11',
  Location: 'Burnley',
  Lot: {
    LotDescription: '15th Century Toy Selection'
  }
}
{
  AuctionDate: '2023-04-13',
  Location: 'Preston',
  Lot: {
    LotDescription: '1960s toys'
  }
}
```

Query Comparison:

- Both queries aim to display “lot description”, “auction date” and “Location”.
- The SQL query directly selects the field from the joined tables whereas MongoDB query uses the “$match” operator to access the nested “Lot” array within Auction collection.

Query F (B)

Query to fetch auction with highest total revenue.

In MySQL:

SELECT Clause:

- **AuctionID**: This function pulls the auction ID from the table of auction.
- **AuctionDate**: This function pulls the auction date from the table of auction.
- **Location**: This method pulls the location out of the auction table.
- **Total_Revenue**: A calculated column that represents total revenue generated for the auction.

FROM CLAUSE:

The inner sub-query calculates the sum of 'WinningPrice' from 'LotSale' table.

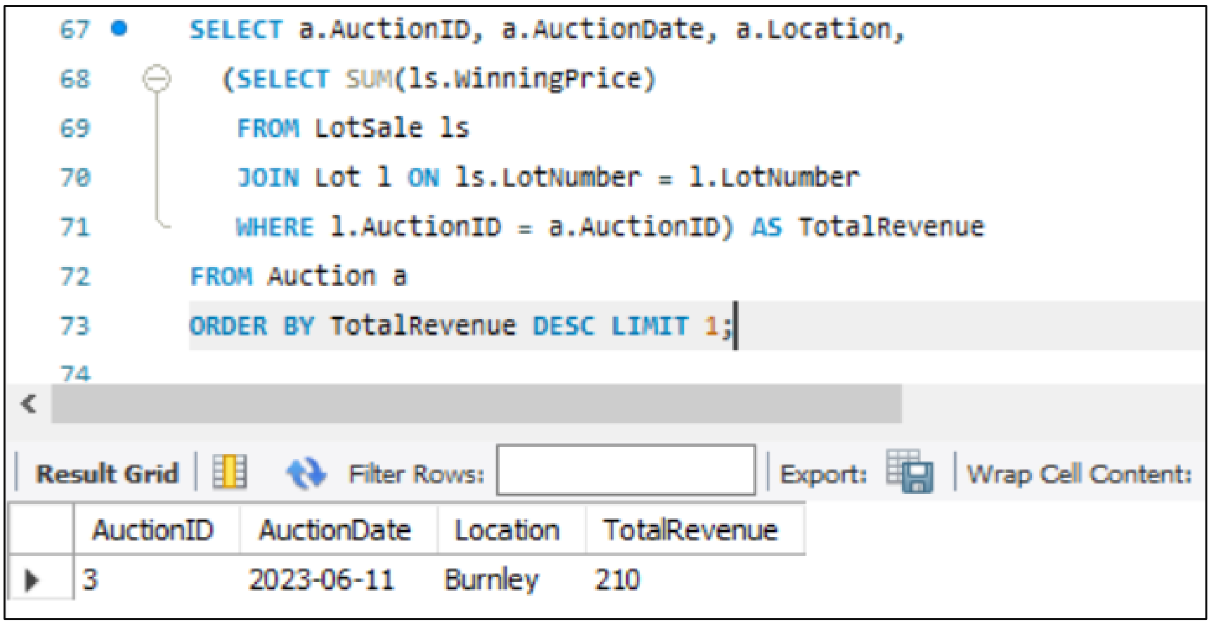
ON CLAUSE:

Then, it corelates with the outer-subquery based on the condition that 'LotNumber' and 'AuctionID' in the 'Lot' table matches with 'AuctionID' in 'Auction' table.

ORDER BY:

The function Order by is used to sort query in descending manner and limits the result just by first record in a descending manner.

(SQL query F.b)



```
SELECT a.AuctionID, a.AuctionDate, a.Location,
  (SELECT SUM(ls.WinningPrice)
   FROM LotSale ls
   JOIN Lot l ON ls.LotNumber = l.LotNumber
   WHERE l.AuctionID = a.AuctionID) AS TotalRevenue
FROM Auction a
ORDER BY TotalRevenue DESC LIMIT 1;
```

AuctionID	AuctionDate	Location	TotalRevenue
3	2023-06-11	Burnley	210

For Mongo DB:

$Unwind **Lot**

Unwinds the collection for "lotsale"

$lookup Stage:

This establishes a link between 'Lot. LotNumber' field from auction collection and 'LotNumber' field from lotsale collection.

$group Stage

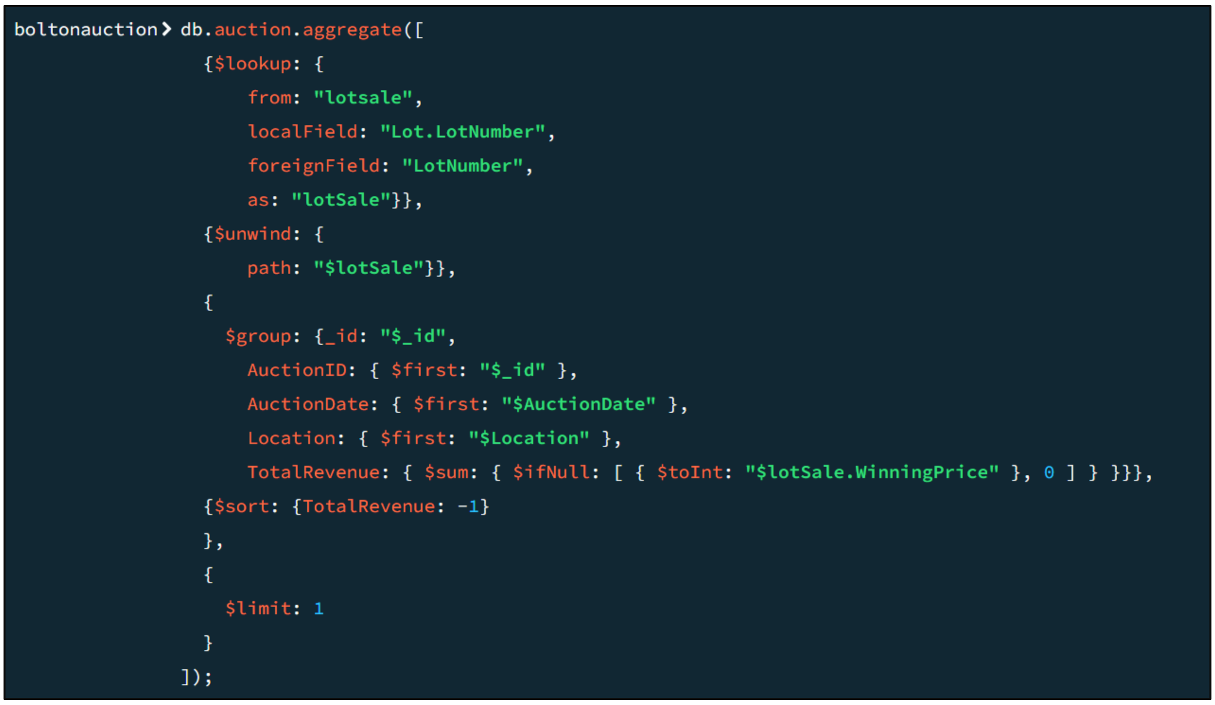
This groups the documents by the '_id' field.

For each group, it calculates the AuctionID, AuctionDate, Location, Total Revenue.

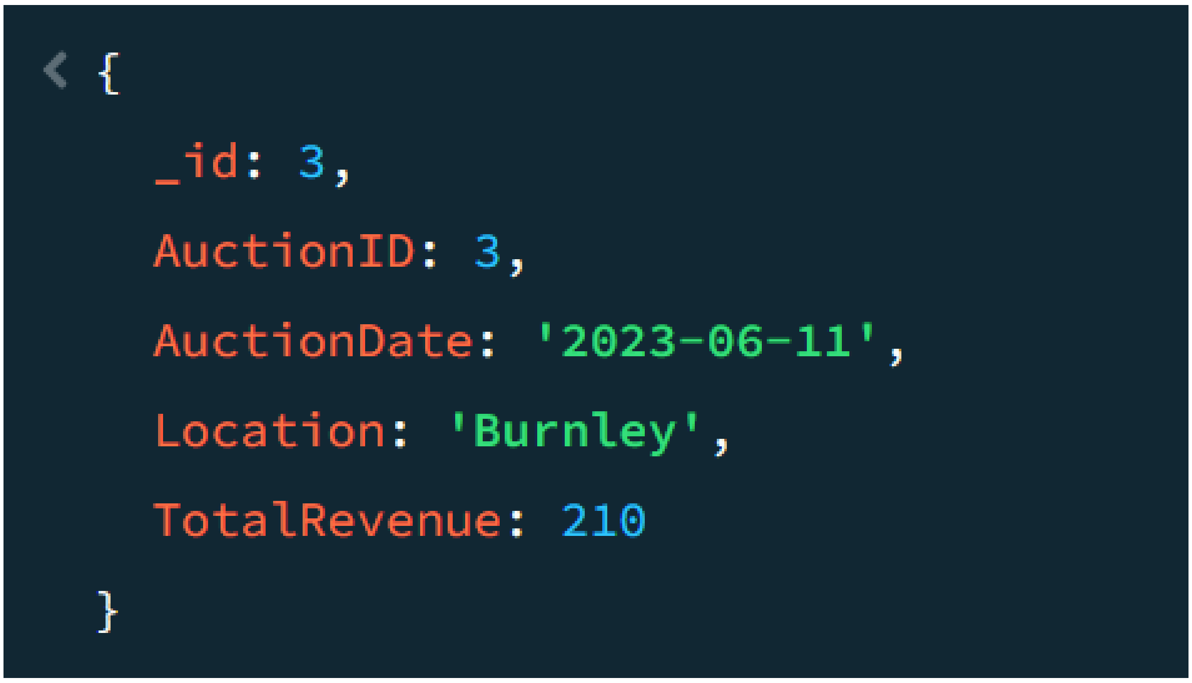
The '$ifnull' operator is used to handle cases where 'WinningPrice' field is null.

The '$toInt' operator converts the value to an integer.

(Mongodb query F.b)


```
boltonauction> db.auction.aggregate([
                 {$lookup: {
                     from: "lotsale",
                     localField: "Lot.LotNumber",
                     foreignField: "LotNumber",
                     as: "lotSale"}},
                 {$unwind: {
                     path: "$lotSale"}},
                 {
                   $group: {_id: "$_id",
                     AuctionID: { $first: "$_id" },
                     AuctionDate: { $first: "$AuctionDate" },
                     Location: { $first: "$Location" },
                     TotalRevenue: { $sum: { $ifNull: [ { $toInt: "$lotSale.WinningPrice" }, 0 ] } }}},
                 {$sort: {TotalRevenue: -1}
                 },
                 {
                   $limit: 1
                 }
               ]);
```

Output:


```
< {
    _id: 3,
    AuctionID: 3,
    AuctionDate: '2023-06-11',
    Location: 'Burnley',
    TotalRevenue: 210
  }
```

Query Comparison:

- Both queries aim to display
- The SQL query join the Auction, Lot, and LotSale tables to calculate the total revenue for each auction by summing the winning prices. Then, the GROUP BY clause to group the results by auction, and the ORDER BY clause to sort the auctions by total revenue in descending order. Finally, the LIMIT 1 to retrieve the auction with the highest total revenue.
- The MongoDB query finds the auction with the greatest total revenue by using the aggregate pipeline. It then returns details about that particular auction, including the AuctionID, AuctionDate, Location, and Total Revenue.

Part 4

Comparison and Justification to SQL and NoSQL databases

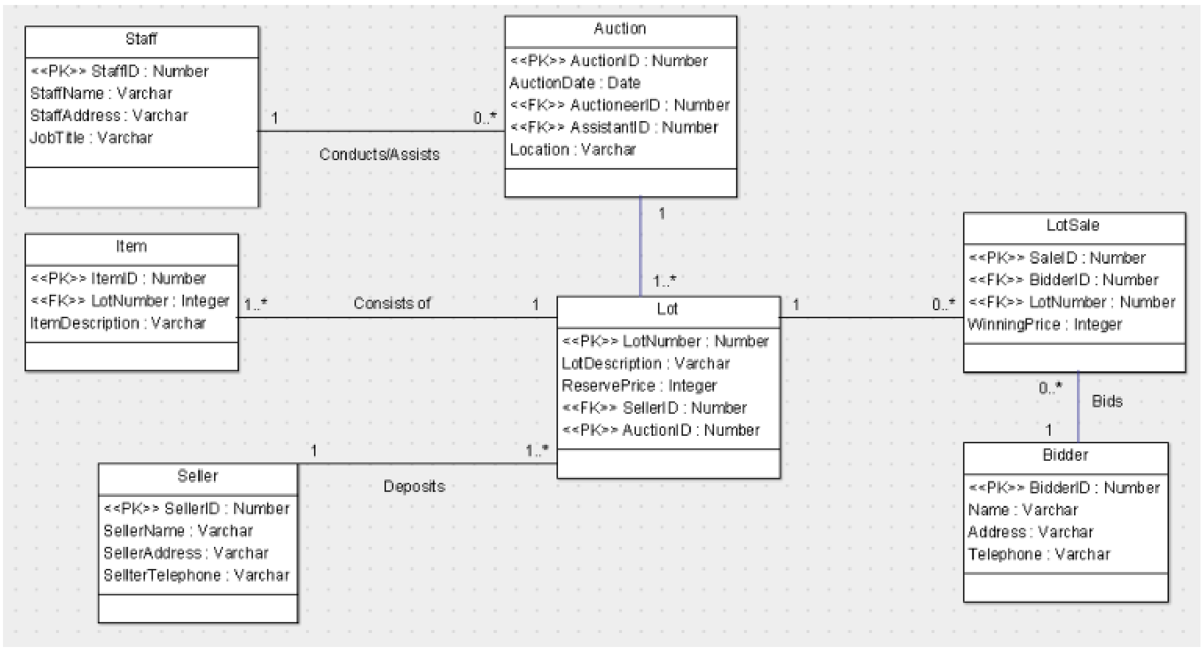
Database Design, Structure and Modelling

MySQL:

- The MySQL database contains seven tables linked with each other in a relational database structure.
- Several linked databases, including Staff, Seller, Bidder, Auction, Lot, Item, and LotSale, are used to organise the data in the Bolton Auction System.
- Each table represents an individual aspect of the auction procedure.
- Because of its structure, MySQL makes it easy to manage and retrieve data while maintaining data integrity through a combination of primary and foreign key constraints.

Creating the SQL Database:

Ensuring a thorough and effective information management system for Bolton Auction House requires the database design and development for the RDBMS based on the given ERD. The following ERD diagram is used:



The included SQL script demonstrates the ability to generate tables in a methodical manner, each with a distinct purpose of managing and organising different elements of the auction process.

Primary and foreign key restrictions connect these tables, allowing meaningful relationships to be established between different entities. Furthermore, the script includes sample data insertion, which is an actual application of the database design (Valade, 2004).

- **CREATE DATABASE IF NOT EXISTS boltonauction;**

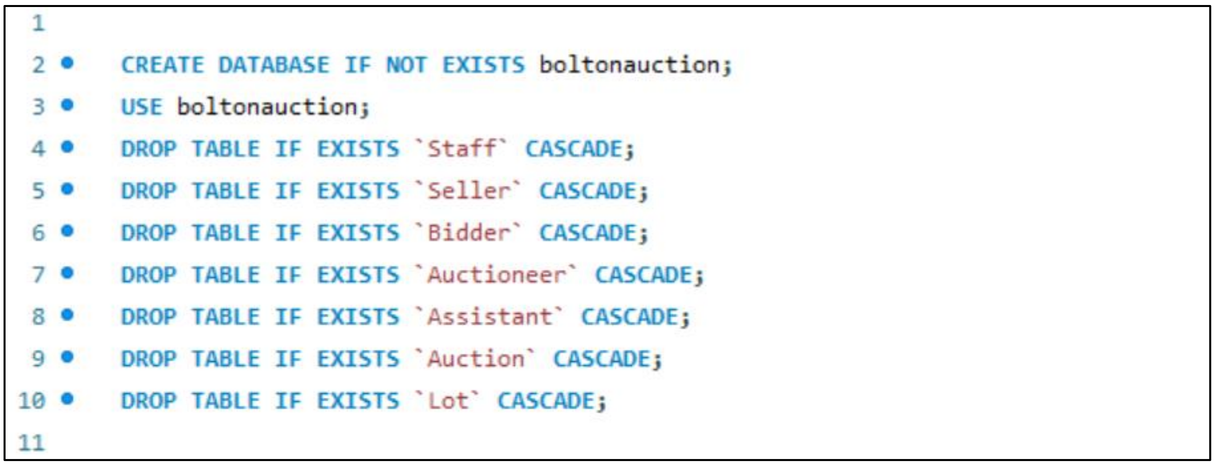
  If the Bolton auction database is yet to be created, this line builds it. The database will only be created if it does not already exist, because of to the IF NOT EXISTS condition.

- **USE boltonauction;**

  This line designates the database boltonauction as the one that will be the primary database when it is used for actions and generating the queries.

- **DROP TABLE IF EXISTS ‘table_name’ CASCADE;**

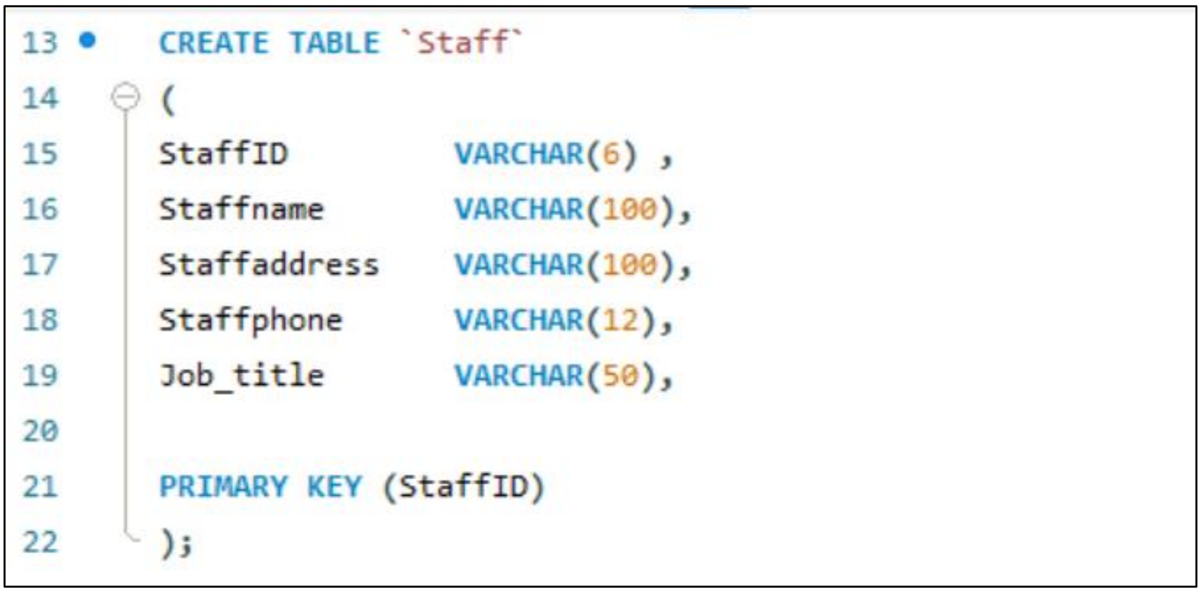
  The table will only be dropped if it is found in the database, because of to the IF EXISTS condition. To make sure that any related objects like foreign keys are eliminated along with the table, the CASCADE keyword is used.



```
CREATE DATABASE IF NOT EXISTS boltonauction;
USE boltonauction;
DROP TABLE IF EXISTS `Staff` CASCADE;
DROP TABLE IF EXISTS `Seller` CASCADE;
DROP TABLE IF EXISTS `Bidder` CASCADE;
DROP TABLE IF EXISTS `Auctioneer` CASCADE;
DROP TABLE IF EXISTS `Assistant` CASCADE;
DROP TABLE IF EXISTS `Auction` CASCADE;
DROP TABLE IF EXISTS `Lot` CASCADE;
```

Within the RDBMS, the following tables and their respective attributes have been defined:

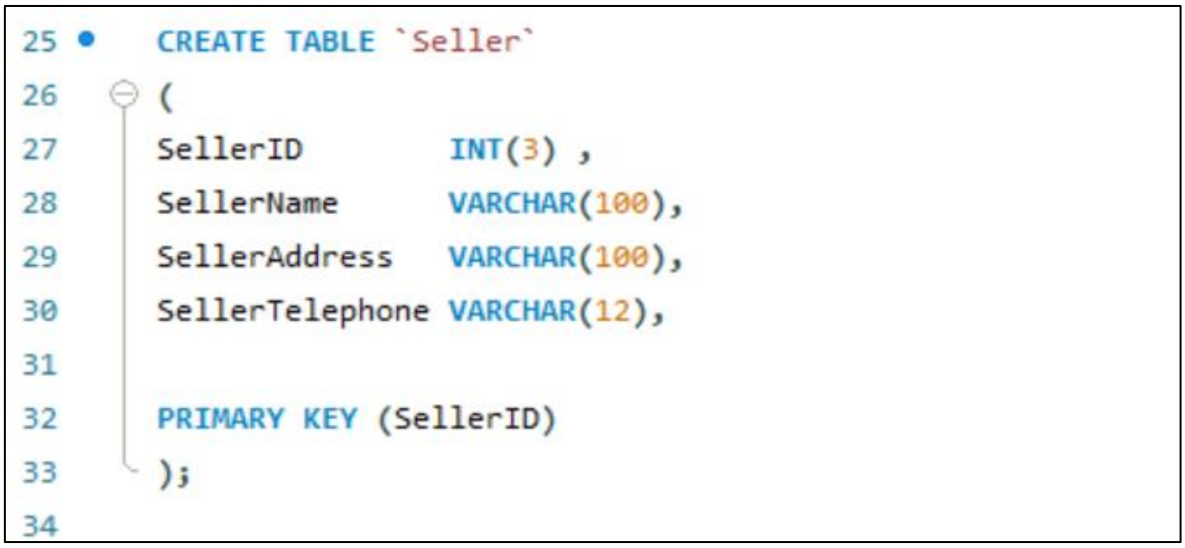
- **Staff**: The table "Staff" provides details on the employees that work at the auction house, such as the staff name, Staff identification number, address and their job title where the table Id is the primary key.



```
CREATE TABLE `Staff`
(
StaffID         VARCHAR(6) ,
Staffname       VARCHAR(100),
Staffaddress    VARCHAR(100),
Staffphone      VARCHAR(12),
Job_title       VARCHAR(50),

PRIMARY KEY (StaffID)
);
```

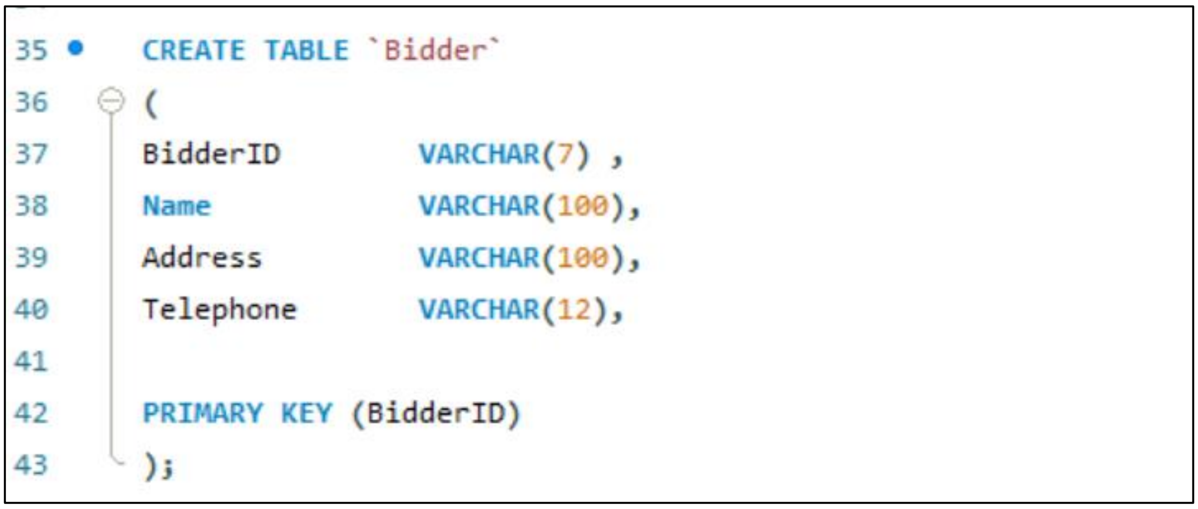
- **Seller**: The table “seller” holds information about the people bidding on goods at the auction, such as their name, ID, phone number, and address where the table Id is the primary key.



```
CREATE TABLE `Seller`
(
SellerID          INT(3) ,
SellerName        VARCHAR(100),
SellerAddress     VARCHAR(100),
SellerTelephone VARCHAR(12),

PRIMARY KEY (SellerID)
);
```

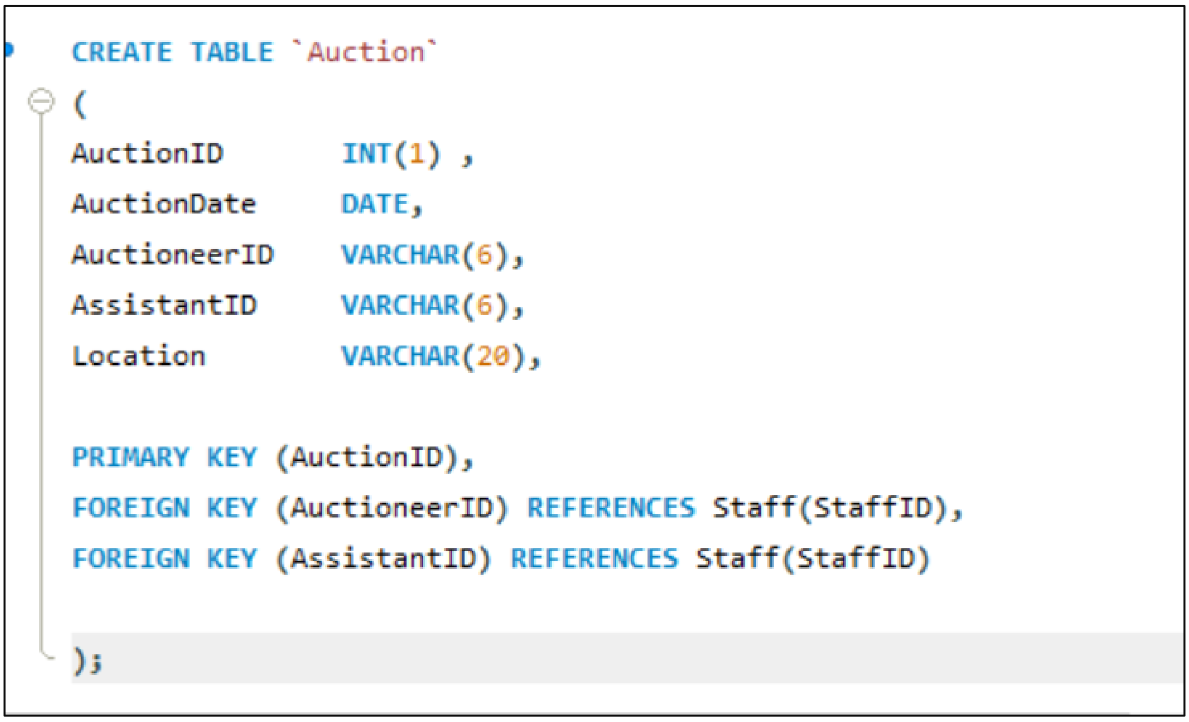
- **Bidder**: The table “bidder” holds data, such as ID, name, address, and phone number, regarding the bidders taking part in the auctions where the table Id is the primary key.



```
CREATE TABLE `Bidder`
(
BidderID          VARCHAR(7) ,
Name              VARCHAR(100),
Address           VARCHAR(100),
Telephone         VARCHAR(12),

PRIMARY KEY (BidderID)
);
```

- **Auction**: The table auction oversees the management of auction data, such as the ID of the auction, the date, the designated assistance and auctioneer referenced from the Staff , and the location.

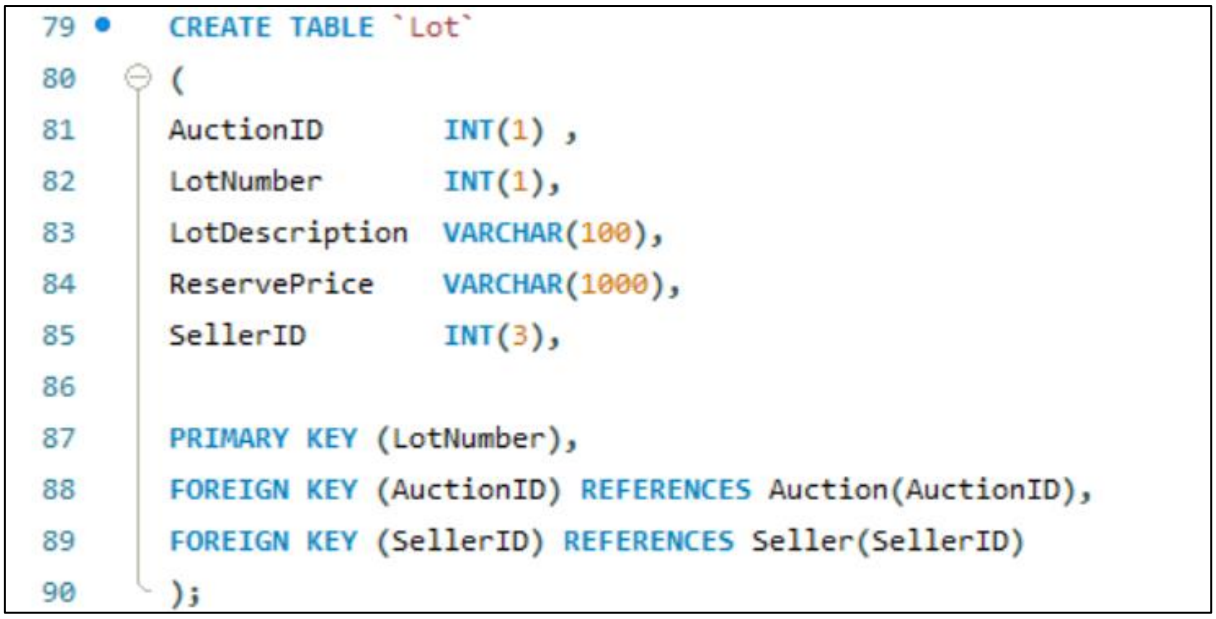


```
CREATE TABLE `Auction`
(
AuctionID        INT(1) ,
AuctionDate      DATE,
AuctioneerID     VARCHAR(6),
AssistantID      VARCHAR(6),
Location         VARCHAR(20),

PRIMARY KEY (AuctionID),
FOREIGN KEY (AuctioneerID) REFERENCES Staff(StaffID),
FOREIGN KEY (AssistantID) REFERENCES Staff(StaffID)

);
```

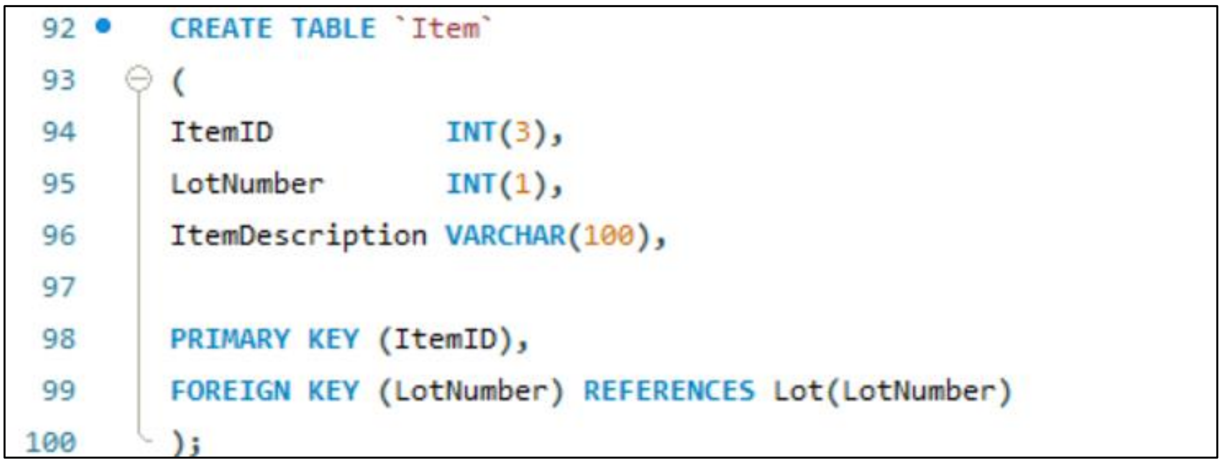
- **Lot**: The table “Lot” stores data associated to the lots selected for auction, including the lot number, description, reserve price, and the ID of the respective seller and the bidder.



```sql
CREATE TABLE `Lot`
(
AuctionID       INT(1) ,
LotNumber       INT(1),
LotDescription  VARCHAR(100),
ReservePrice    VARCHAR(1000),
SellerID        INT(3),

PRIMARY KEY (LotNumber),
FOREIGN KEY (AuctionID) REFERENCES Auction(AuctionID),
FOREIGN KEY (SellerID) REFERENCES Seller(SellerID)
);
```

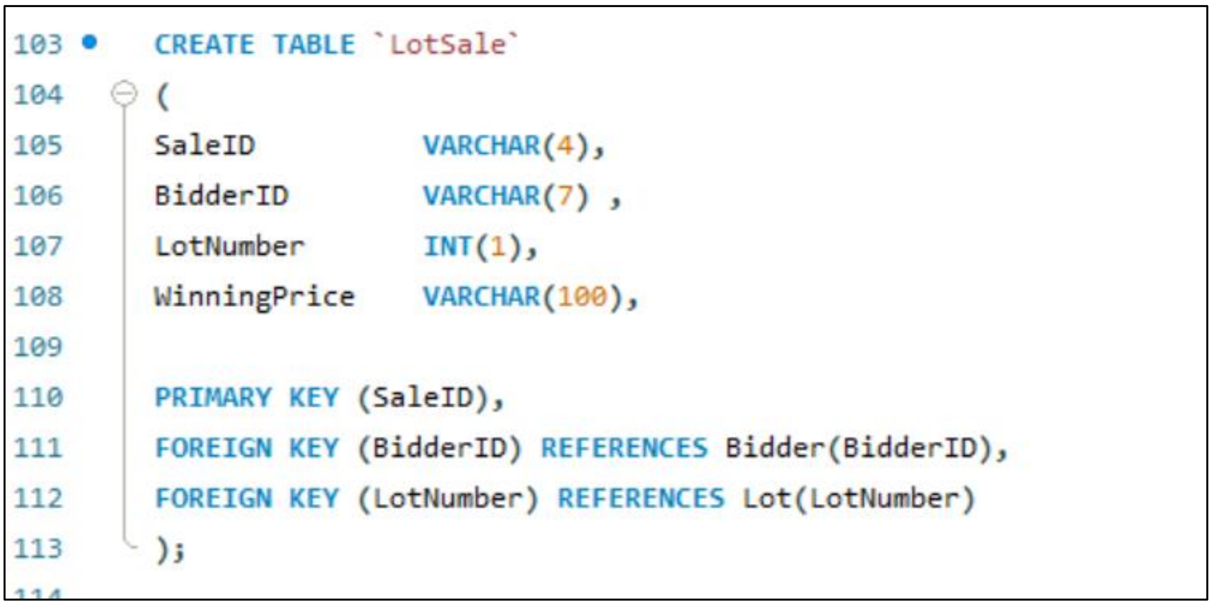
- **Item**: The table "Item" organises information about each lot's items, such as the item ID, lot number that goes with it, and description where lot number is taken as a foreign key from lot table.



```sql
CREATE TABLE `Item`
(
ItemID          INT(3),
LotNumber       INT(1),
ItemDescription VARCHAR(100),

PRIMARY KEY (ItemID),
FOREIGN KEY (LotNumber) REFERENCES Lot(LotNumber)
);
```

- **LotSale**: The table "LotSale" keeps track of the specifics of the profitable sales, such as the lot number, winning price, bidder ID, and sale ID where the bidder ID and lot number are taken as foreign key from their respective bidder and lot tables.



```sql
CREATE TABLE `LotSale`
(
SaleID          VARCHAR(4),
BidderID        VARCHAR(7) ,
LotNumber       INT(1),
WinningPrice    VARCHAR(100),

PRIMARY KEY (SaleID),
FOREIGN KEY (BidderID) REFERENCES Bidder(BidderID),
FOREIGN KEY (LotNumber) REFERENCES Lot(LotNumber)
);
```

MongoDB:

- The MongoDB database has five collections: Staff, Seller, Bidder, Lot Sale, and Auction.
- It uses a NoSQL methodology because MongoDB has a more flexible data format that supports dynamic and unstructured data storage, the data is kept as documents that resemble JSON.
- Collections have a less constraining and more dynamic structure than conventional SQL databases.
- The "Auction" collection in the given JSON script has nested arrays for "StaffID," "Lot," and "Items", these arrays represent embedded documents to the corresponding documents in other collections.

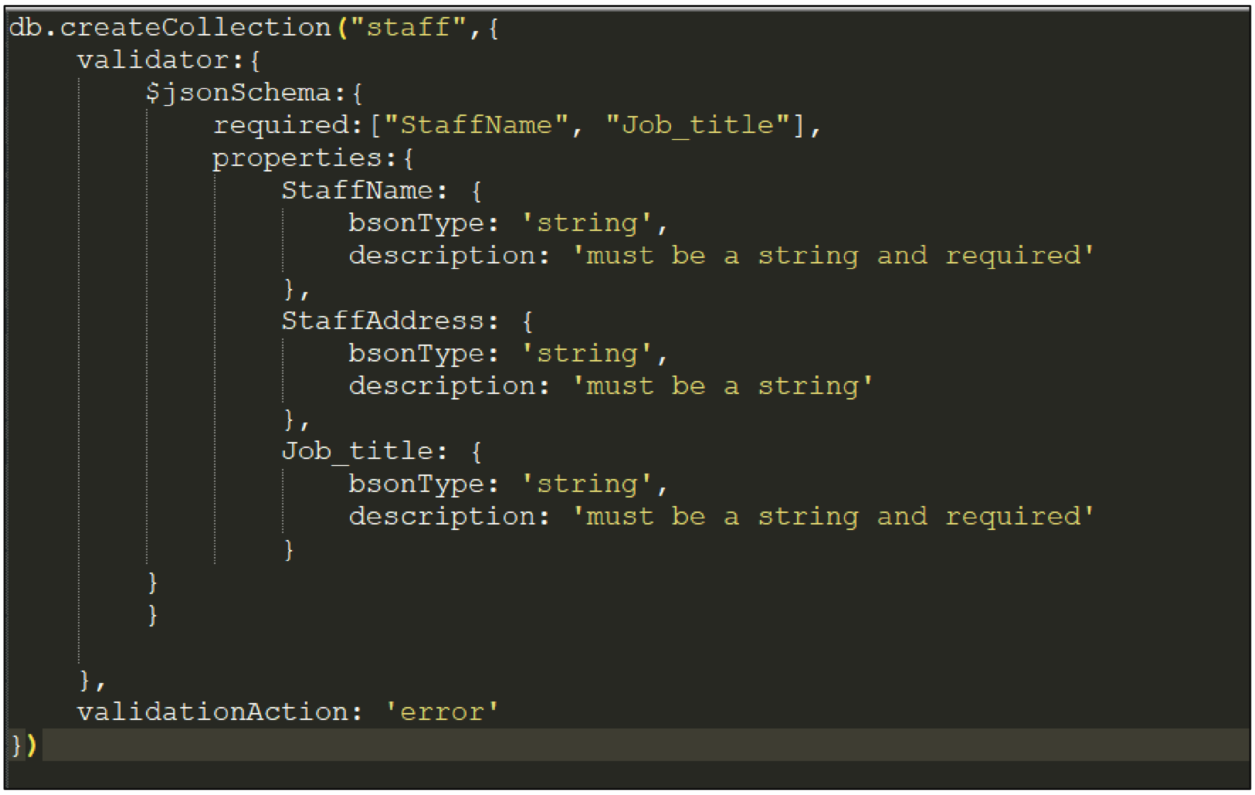
Creating the Collections and Validating the Respective Schema (mongodb, 2023)

1. Staff Collection

The purpose of this collection is to store data about the staff of the Bolton Auction House. Schema validation is essential in maintaining data security, consistency, and quality in MongoDB collections, which enhances the overall resilience and reliability of the database system.

- The collection's validation rules are specified using the validator key.
- The JSON schema that is utilised for validation is specified by the $jsonSchema key.
- The mandatory nature of the "StaffName" and "Job_title" elements is made apparent by the required array present in the JSON syntax.
- Any document that does not follow the specified validation rules will result in an error because the Validation Action key is set to 'error'.
- Every field is defined with its corresponding validation criteria under the properties key:

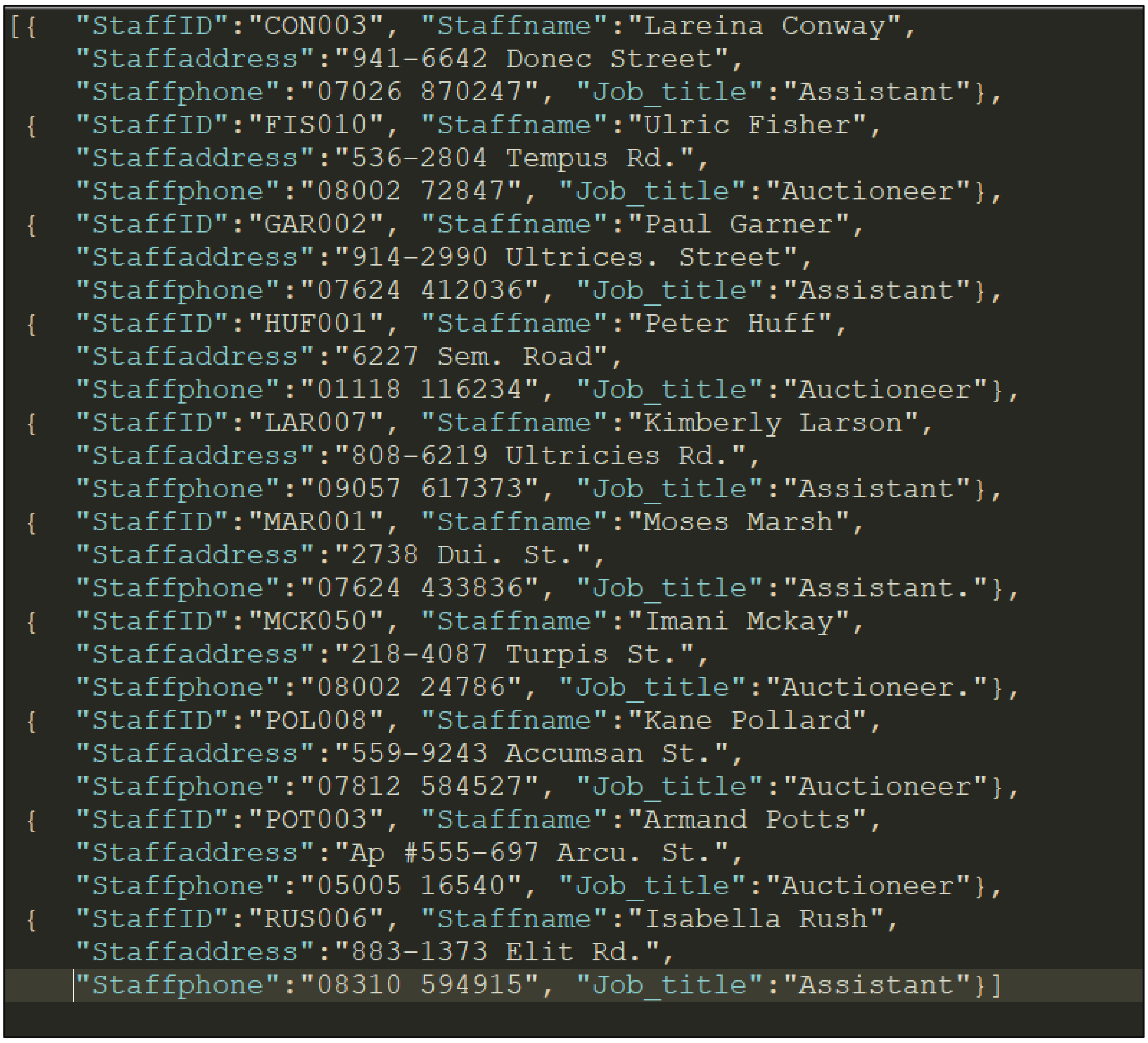
  - ➢ StaffName: This is a necessary field that needs to be of the BSON type “string”. For the field, an additional description is given.
  - ➢ StaffAddress: This is a mandatory field that needs to be of BSON type “string”.
  - ➢ Job_title: It is necessary and has to be of the BSON type “string”.



```
db.createCollection("staff",{
    validator:{
        $jsonSchema:{
            required:["StaffName", "Job_title"],
            properties:{
                StaffName: {
                    bsonType: 'string',
                    description: 'must be a string and required'
                },
                StaffAddress: {
                    bsonType: 'string',
                    description: 'must be a string'
                },
                Job_title: {
                    bsonType: 'string',
                    description: 'must be a string and required'
                }
        }
        }

    },
    validationAction: 'error'
})
```

Inserting the Staff Collection

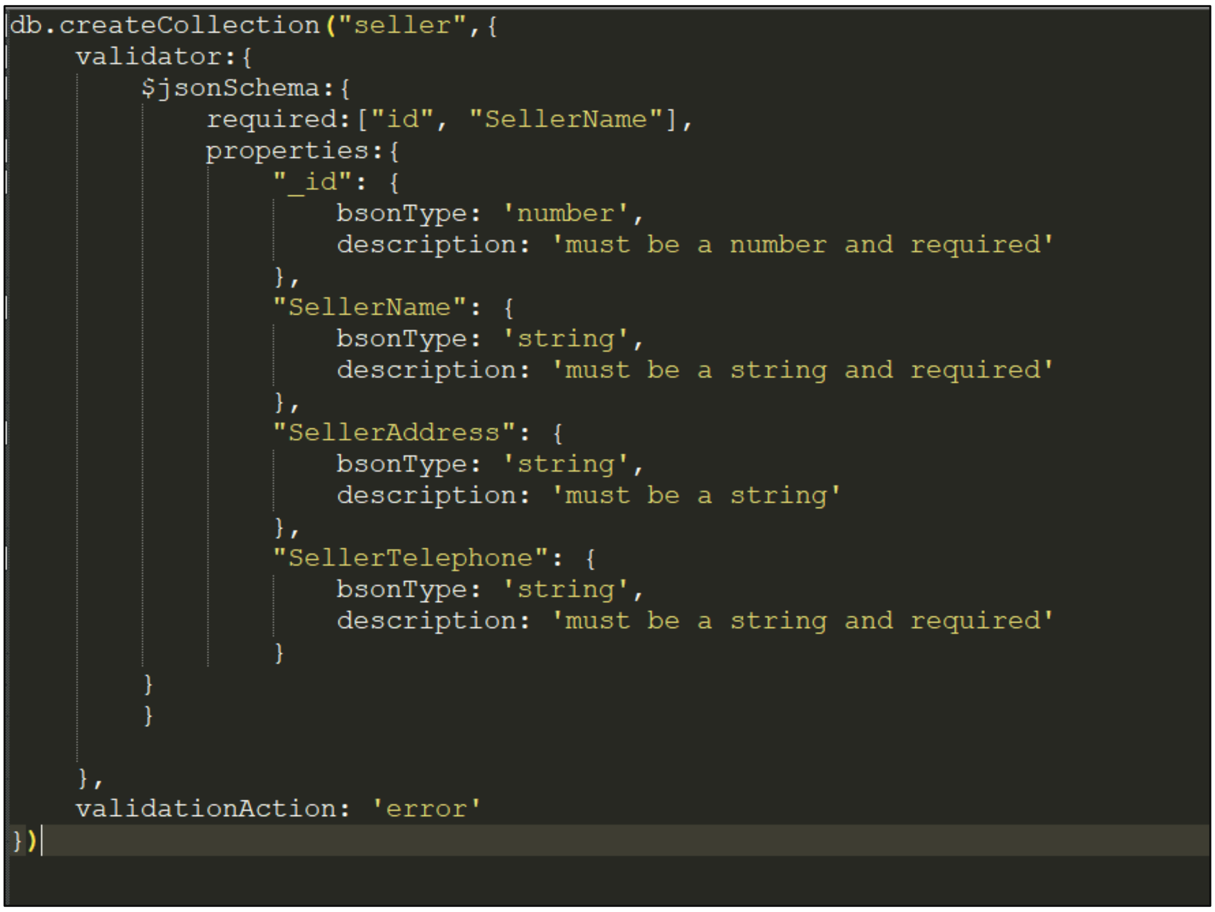

```
[{  "StaffID":"CON003", "Staffname":"Lareina Conway",
    "Staffaddress":"941-6642 Donec Street",
    "Staffphone":"07026 870247", "Job_title":"Assistant"},
 {  "StaffID":"FIS010", "Staffname":"Ulric Fisher",
    "Staffaddress":"536-2804 Tempus Rd.",
    "Staffphone":"08002 72847", "Job_title":"Auctioneer"},
 {  "StaffID":"GAR002", "Staffname":"Paul Garner",
    "Staffaddress":"914-2990 Ultrices. Street",
    "Staffphone":"07624 412036", "Job_title":"Assistant"},
 {  "StaffID":"HUF001", "Staffname":"Peter Huff",
    "Staffaddress":"6227 Sem. Road",
    "Staffphone":"01118 116234", "Job_title":"Auctioneer"},
 {  "StaffID":"LAR007", "Staffname":"Kimberly Larson",
    "Staffaddress":"808-6219 Ultricies Rd.",
    "Staffphone":"09057 617373", "Job_title":"Assistant"},
 {  "StaffID":"MAR001", "Staffname":"Moses Marsh",
    "Staffaddress":"2738 Dui. St.",
    "Staffphone":"07624 433836", "Job_title":"Assistant."},
 {  "StaffID":"MCK050", "Staffname":"Imani Mckay",
    "Staffaddress":"218-4087 Turpis St.",
    "Staffphone":"08002 24786", "Job_title":"Auctioneer."},
 {  "StaffID":"POL008", "Staffname":"Kane Pollard",
    "Staffaddress":"559-9243 Accumsan St.",
    "Staffphone":"07812 584527", "Job_title":"Auctioneer"},
 {  "StaffID":"POT003", "Staffname":"Armand Potts",
    "Staffaddress":"Ap #555-697 Arcu. St.",
    "Staffphone":"05005 16540", "Job_title":"Auctioneer"},
 {  "StaffID":"RUS006", "Staffname":"Isabella Rush",
    "Staffaddress":"883-1373 Elit Rd.",
    "Staffphone":"08310 594915", "Job_title":"Assistant"}]
```

2. Seller Collection

The purpose of this collection is to store data about the sellers of the Bolton Auction House.

The "_id" and "sellerName" are required fields in this collection.

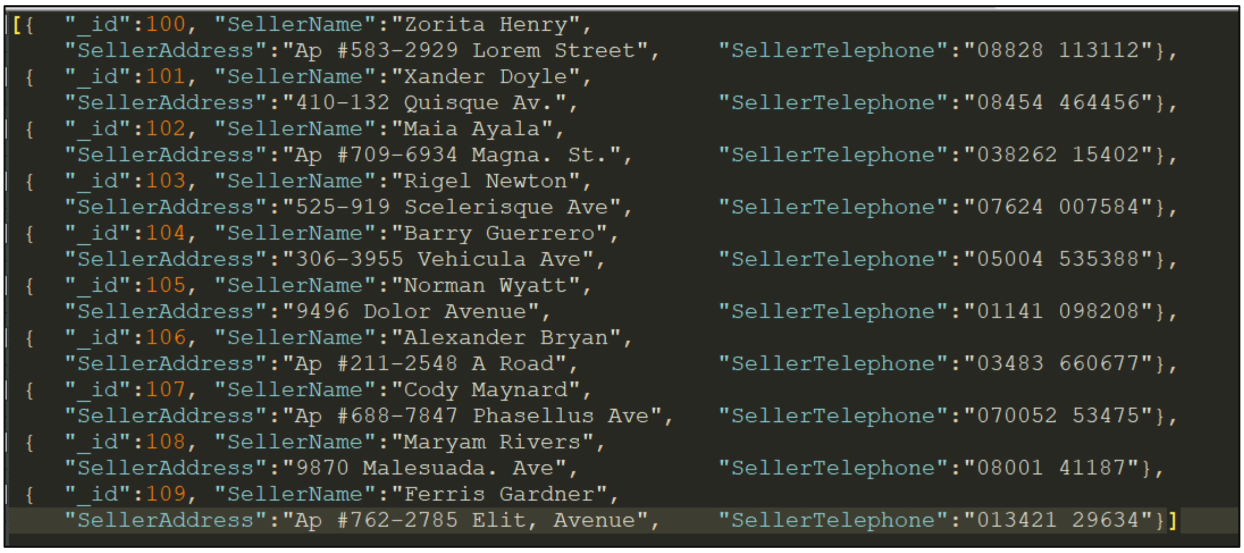
- **_id**: This is a necessary field that needs to be of BSON type "number." The _id here is not an object id rather the seller id is manually inserted in this field.
- **SellerName**: It is necessary and has to be of the BSON type "string."
- **SellerAddress**: This is a mandatory field that needs to be of BSON type'string'.
- **SellerTelephone**: It is necessary and needs to be of the BSON type "string."



```
db.createCollection("seller",{
    validator:{
        $jsonSchema:{
            required:["id", "SellerName"],
            properties:{
                "_id": {
                    bsonType: 'number',
                    description: 'must be a number and required'
                },
                "SellerName": {
                    bsonType: 'string',
                    description: 'must be a string and required'
                },
                "SellerAddress": {
                    bsonType: 'string',
                    description: 'must be a string'
                },
                "SellerTelephone": {
                    bsonType: 'string',
                    description: 'must be a string and required'
                }
        }
        }

    },
    validationAction: 'error'
})
```

Inserting the Seller Collection



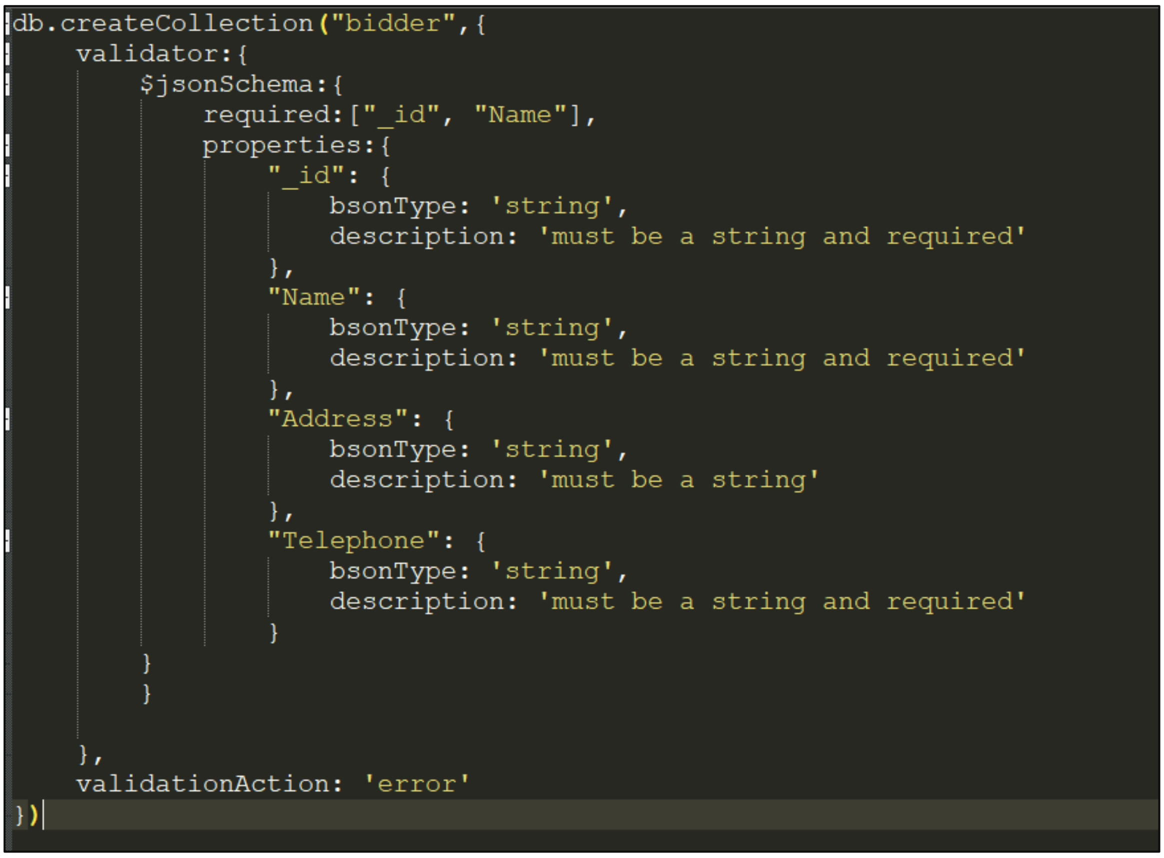
```
[{  "_id":100, "SellerName":"Zorita Henry",
    "SellerAddress":"Ap #583-2929 Lorem Street",    "SellerTelephone":"08828 113112"},
 {  "_id":101, "SellerName":"Xander Doyle",
    "SellerAddress":"410-132 Quisque Av.",          "SellerTelephone":"08454 464456"},
 {  "_id":102, "SellerName":"Maia Ayala",
    "SellerAddress":"Ap #709-6934 Magna. St.",      "SellerTelephone":"038262 15402"},
 {  "_id":103, "SellerName":"Rigel Newton",
    "SellerAddress":"525-919 Scelerisque Ave",      "SellerTelephone":"07624 007584"},
 {  "_id":104, "SellerName":"Barry Guerrero",
    "SellerAddress":"306-3955 Vehicula Ave",        "SellerTelephone":"05004 535388"},
 {  "_id":105, "SellerName":"Norman Wyatt",
    "SellerAddress":"9496 Dolor Avenue",            "SellerTelephone":"01141 098208"},
 {  "_id":106, "SellerName":"Alexander Bryan",
    "SellerAddress":"Ap #211-2548 A Road",          "SellerTelephone":"03483 660677"},
 {  "_id":107, "SellerName":"Cody Maynard",
    "SellerAddress":"Ap #688-7847 Phasellus Ave",   "SellerTelephone":"070052 53475"},
 {  "_id":108, "SellerName":"Maryam Rivers",
    "SellerAddress":"9870 Malesuada. Ave",          "SellerTelephone":"08001 41187"},
 {  "_id":109, "SellerName":"Ferris Gardner",
    "SellerAddress":"Ap #762-2785 Elit, Avenue",    "SellerTelephone":"013421 29634"}]
```

3. Bidder Collection

The purpose of this collection is to store data about the bidders of the Bolton Auction House.

The "_id" and "name" are required field here.

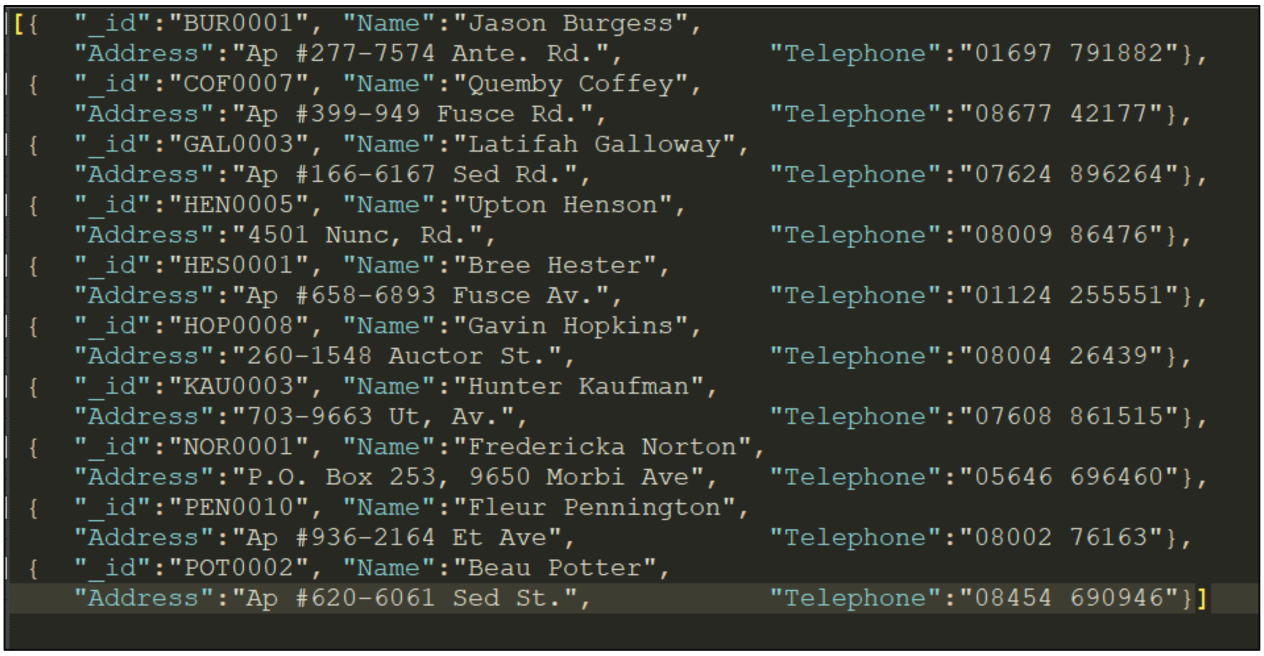
- **_id** - This is a necessary field that needs to be a string in BSON format. For the field, an additional description is given.
- **Name** - The needed field needs to be of BSON type'string'.
- **Address** - It is not necessary, although it needs to be of the BSON type "string."
- **Phone** - It is necessary and needs to be of the BSON type "string."



```
db.createCollection("bidder",{
    validator:{
        $jsonSchema:{
            required:["_id", "Name"],
            properties:{
                "_id": {
                    bsonType: 'string',
                    description: 'must be a string and required'
                },
                "Name": {
                    bsonType: 'string',
                    description: 'must be a string and required'
                },
                "Address": {
                    bsonType: 'string',
                    description: 'must be a string'
                },
                "Telephone": {
                    bsonType: 'string',
                    description: 'must be a string and required'
                }
        }
        }

    },
    validationAction: 'error'
})
```

Inserting the Bidder Collection



```
[{  "_id":"BUR0001", "Name":"Jason Burgess",
    "Address":"Ap #277-7574 Ante. Rd.",         "Telephone":"01697 791882"},
 {  "_id":"COF0007", "Name":"Quemby Coffey",
    "Address":"Ap #399-949 Fusce Rd.",          "Telephone":"08677 42177"},
 {  "_id":"GAL0003", "Name":"Latifah Galloway",
    "Address":"Ap #166-6167 Sed Rd.",           "Telephone":"07624 896264"},
 {  "_id":"HEN0005", "Name":"Upton Henson",
    "Address":"4501 Nunc, Rd.",                 "Telephone":"08009 86476"},
 {  "_id":"HES0001", "Name":"Bree Hester",
    "Address":"Ap #658-6893 Fusce Av.",         "Telephone":"01124 255551"},
 {  "_id":"HOP0008", "Name":"Gavin Hopkins",
    "Address":"260-1548 Auctor St.",            "Telephone":"08004 26439"},
 {  "_id":"KAU0003", "Name":"Hunter Kaufman",
    "Address":"703-9663 Ut, Av.",               "Telephone":"07608 861515"},
 {  "_id":"NOR0001", "Name":"Fredericka Norton",
    "Address":"P.O. Box 253, 9650 Morbi Ave",   "Telephone":"05646 696460"},
 {  "_id":"PEN0010", "Name":"Fleur Pennington",
    "Address":"Ap #936-2164 Et Ave",            "Telephone":"08002 76163"},
 {  "_id":"POT0002", "Name":"Beau Potter",
    "Address":"Ap #620-6061 Sed St.",           "Telephone":"08454 690946"}]
```

4. Auction Collection

The purpose of this table is to store data about the auctions that the Bolton Auction House holds. It probably keeps track of different information related to each purchase.

- **_id** - It is necessary and needs to be a number.
- **AuctionDate** - It is necessary and needs to be a string.
- **StaffID** – It is an array consisting of at least one string.
- **Location** – It is necessary to be a string.
- **Lot** – It is an array has four essential fields: "LotDescription," "LotNumber," "ReservePrice," and "SellerID." There is a specified data type for each of these fields.
- **Items**- It is an array of objects with the values "ItemID" and "ItemDescription"

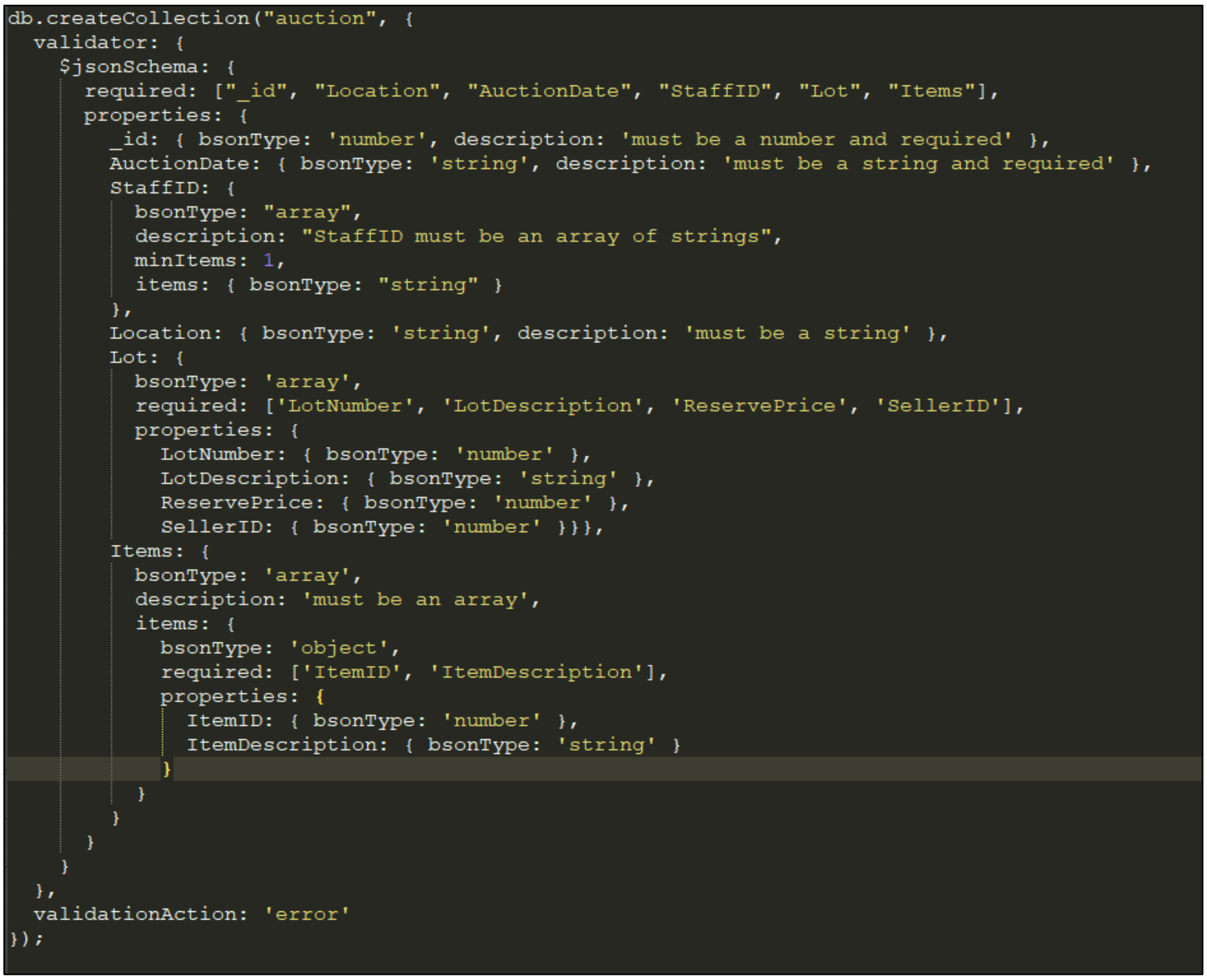
Embedding the Auction Collection

As demonstrated by the ‘Auction’ collection of the MongoDB database for the Bolton Auction System is embedded. According to (Copeland, 2013), embedding data within a document has a number of advantages that make it a well suited option for the auction sector. In this instance, it makes more sense to include “Lot” and “Items” in the “Auction” collection rather than establishing separate collections for them for the following reasons by (mongodb, 2023):

1. **Data Locality**: When relevant data is included into a single document, data locality is improved, making it possible to retrieve related information more quickly and effectively. Because “lots” and “items” of the Bolton auction database are directly linked to “auctions”, integrating them allows for faster access to relevant data, which minimises the need for expensive and time-consuming joins or lookups across several collections.
2. **Atomicity and Consistency**: When data is embedded, operations on it become precise, guaranteeing the data's consistency across the entire document. This preserves the integrity and coherence of the data linked to a particular auction by guaranteeing that related data remains together. Inconsistencies that could occur with distinct collections can be avoided by performing changes to an auction, its lots, and its objects as a single atomic action.
3. **Performance Optimisation**: Because embedding may reduce the number of queries needed to retrieve related data, it is advantageous for read-intensive activities. Improved efficiency and reduced latency result from keeping all pertinent data inside the 'Auction' page, which eliminates the need for separate queries to fetch related data from other collections.

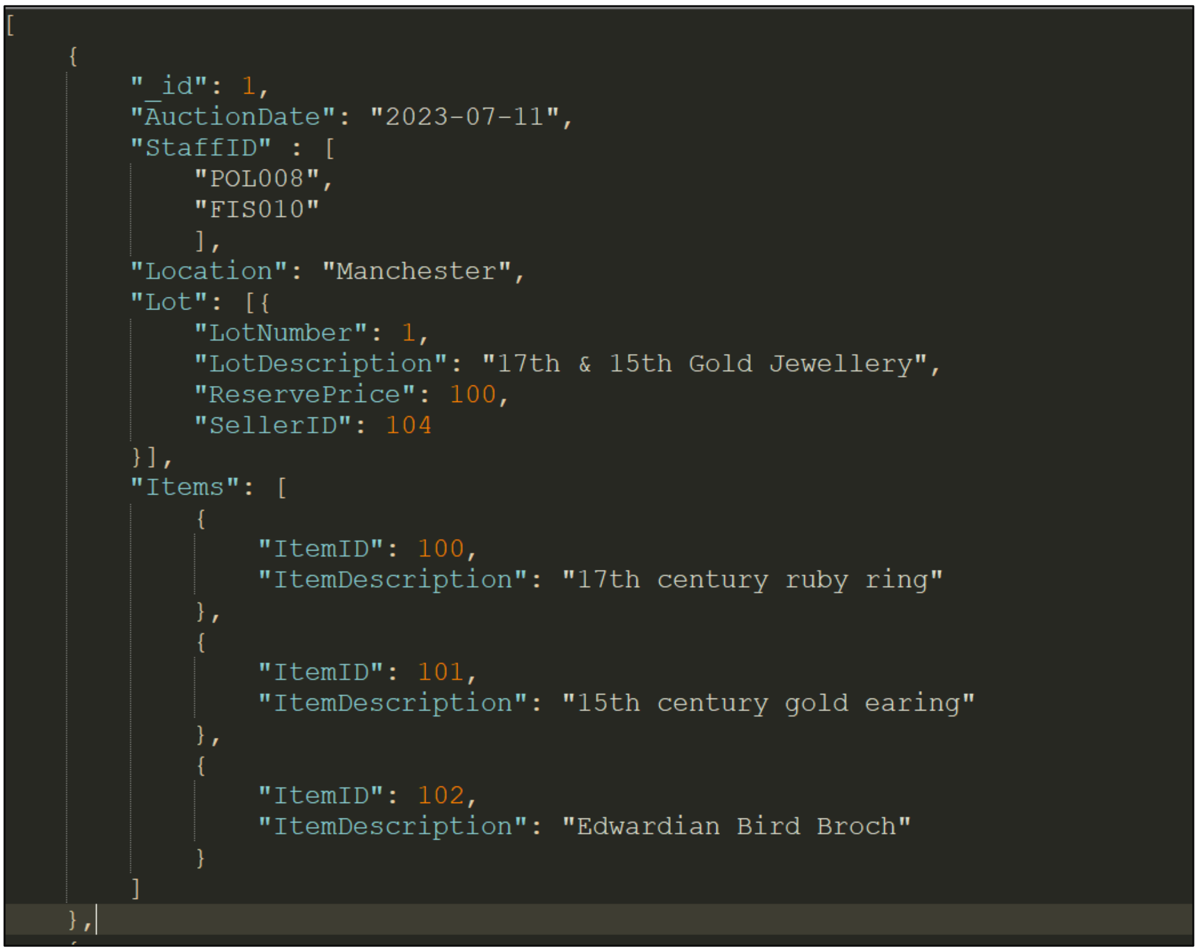
4. **Schema Flexibility**: Dynamic data structures can be stored in a single document because of the schema flexibility provided by embedded documents. Embedding makes it possible to store variable data within the same document during an auction, even in situations where the number of lots and items up for auctions may change. This reduces the need to make adjustments to the specified schema unlike RDBMS schema.
5. **Simpler Updates and Retrieval**: Embedded documents make it easier to update and retrieve relevant data. The ability to execute queries in a single operation to retrieve or update an auction's details, lots, and objects streamlines data access and reduces logical complexity in the application.

The use of embedding might have certain trade-offs such as document expansion and heavy work-load writing (geeksforgeeks, 2022). Considering our current scenario, given data and the advantages aforementioned, embedding is used in the Auction collection as follows:



```
db.createCollection("auction", {
  validator: {
    $jsonSchema: {
      required: ["_id", "Location", "AuctionDate", "StaffID", "Lot", "Items"],
      properties: {
        _id: { bsonType: 'number', description: 'must be a number and required' },
        AuctionDate: { bsonType: 'string', description: 'must be a string and required' },
        StaffID: {
          bsonType: "array",
          description: "StaffID must be an array of strings",
          minItems: 1,
          items: { bsonType: "string" }
        },
        Location: { bsonType: 'string', description: 'must be a string' },
        Lot: {
          bsonType: 'array',
          required: ['LotNumber', 'LotDescription', 'ReservePrice', 'SellerID'],
          properties: {
            LotNumber: { bsonType: 'number' },
            LotDescription: { bsonType: 'string' },
            ReservePrice: { bsonType: 'number' },
            SellerID: { bsonType: 'number' }}},
        Items: {
          bsonType: 'array',
          description: 'must be an array',
          items: {
            bsonType: 'object',
            required: ['ItemID', 'ItemDescription'],
            properties: {
              ItemID: { bsonType: 'number' },
              ItemDescription: { bsonType: 'string' }
            }
          }
        }
      }
    }
  },
  validationAction: 'error'
});
```

Inserting data into Auction Collection

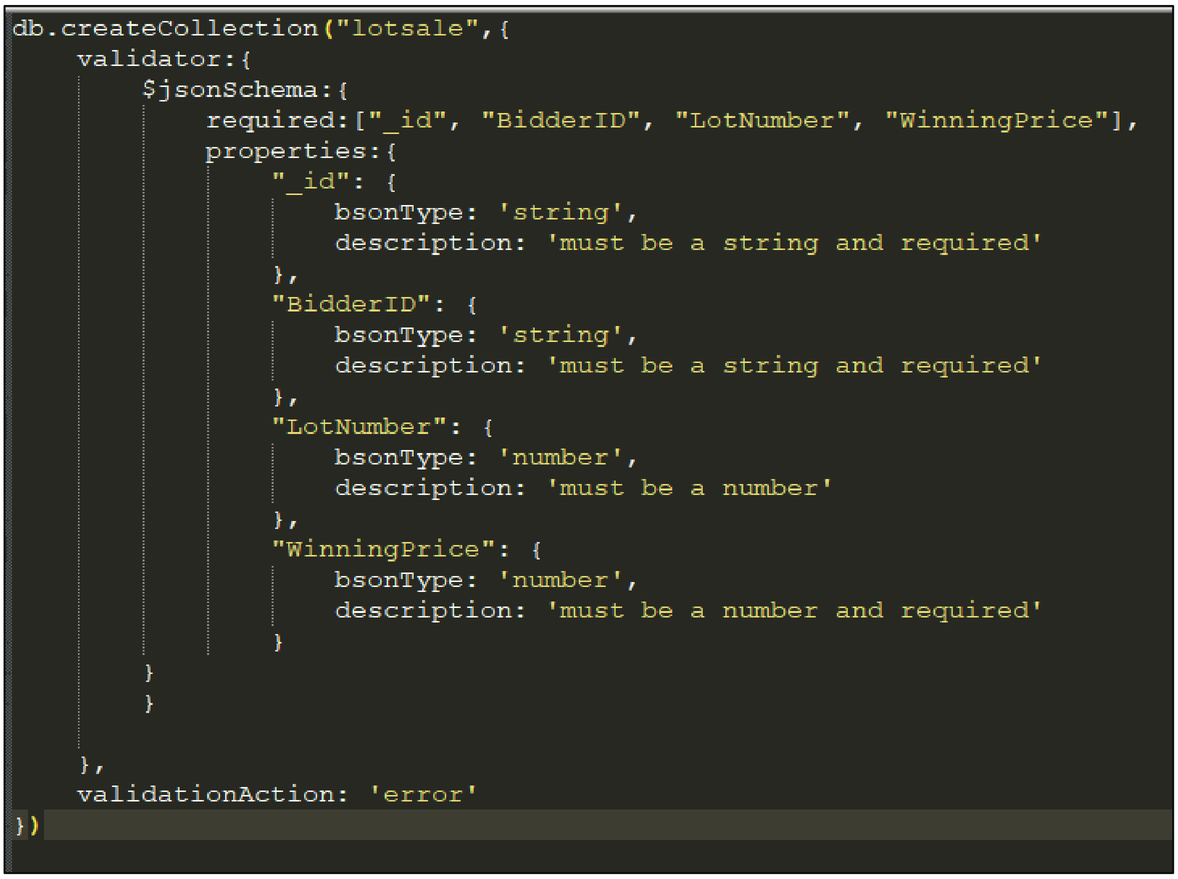


```
[
    {
        "_id": 1,
        "AuctionDate": "2023-07-11",
        "StaffID" : [
            "POL008",
            "FIS010"
            ],
        "Location": "Manchester",
        "Lot": [{
            "LotNumber": 1,
            "LotDescription": "17th & 15th Gold Jewellery",
            "ReservePrice": 100,
            "SellerID": 104
        }],
        "Items": [
            {
                "ItemID": 100,
                "ItemDescription": "17th century ruby ring"
            },
            {
                "ItemID": 101,
                "ItemDescription": "15th century gold earing"
            },
            {
                "ItemID": 102,
                "ItemDescription": "Edwardian Bird Broch"
            }
        ]
    },
```

6. Lot Sale Collection

The purpose of this table is to store data about the sales that the Bolton Auction House had.

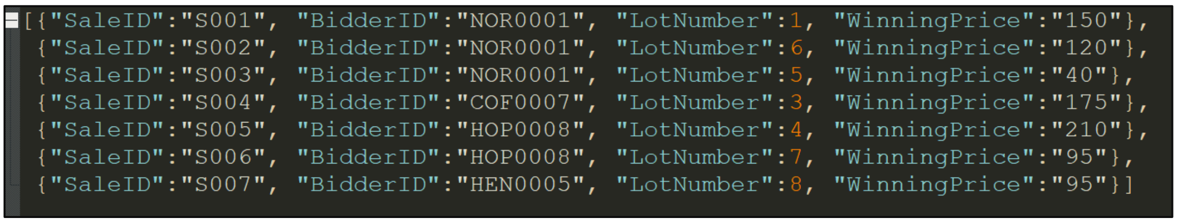
- **_id** - This is a necessary field that needs to be a string in BSON format. For the field, an additional description is given.
- **BidderID** - This is a necessary field that needs to be a string in BSON format.
- **LotNumber** - This is optional, however it must to be of the BSON type "number."
- **WinningPrice** - This is a necessary field that needs to be of the BSON type "number."



```
db.createCollection("lotsale",{
    validator:{
        $jsonSchema:{
            required:["_id", "BidderID", "LotNumber", "WinningPrice"],
            properties:{
                "_id": {
                    bsonType: 'string',
                    description: 'must be a string and required'
                },
                "BidderID": {
                    bsonType: 'string',
                    description: 'must be a string and required'
                },
                "LotNumber": {
                    bsonType: 'number',
                    description: 'must be a number'
                },
                "WinningPrice": {
                    bsonType: 'number',
                    description: 'must be a number and required'
                }
            }
        }

    },
    validationAction: 'error'
})
```

Inserting into Lot Sale Collection



```
[{"SaleID":"S001", "BidderID":"NOR0001", "LotNumber":1, "WinningPrice":"150"},
 {"SaleID":"S002", "BidderID":"NOR0001", "LotNumber":6, "WinningPrice":"120"},
 {"SaleID":"S003", "BidderID":"NOR0001", "LotNumber":5, "WinningPrice":"40"},
 {"SaleID":"S004", "BidderID":"COF0007", "LotNumber":3, "WinningPrice":"175"},
 {"SaleID":"S005", "BidderID":"HOP0008", "LotNumber":4, "WinningPrice":"210"},
 {"SaleID":"S006", "BidderID":"HOP0008", "LotNumber":7, "WinningPrice":"95"},
 {"SaleID":"S007", "BidderID":"HEN0005", "LotNumber":8, "WinningPrice":"95"}]
```

Efficiency and Scalability

The following is the comparison between SQL and NoSQL according to (Andreas Meier, 2019):

SQL:

Because of its predefined schema, MySQL might not perform well when processing large volume of data, despite being renowned for reliability and performance with structured data. As the volume of data increases, scaling could turn into a challenging and costly process.

NoSQL:

This NoSQL database allows for easy handling of massive amount of unstructured data and provides horizontal scalability. Because of its distributed architecture, it can accommodate more nodes, guaranteeing high availability and smooth scalability as the amount of data required by the application increases.

Data Consistency and Integrity

The following is the comparison between SQL and NoSQL according to (Andreas Meier, 2019):

SQL:

MySQL guarantees data consistency and integrity with transaction support and compliance with ACID (Atomicity, Consistency, Isolation, Durability). Its rigorous schema enforcement keeps erroneous data from being added, preserving the data's quality.

NoSQL:

MongoDB may compromise some aspects of data integrity in order to achieve scale and performance, even though it provides eventual consistency by default. Because of its flexible design, different data structures can be stored within the same collection, facilitating rapid iterations but perhaps resulting in inconsistent data if not handled carefully.

Flexibility and Evolution of Schemas

The following is the comparison between SQL and NoSQL according to (Andreas Meier, 2019):

SQL:

MySQL has a rigid, predetermined schema that needs to be carefully planned out before being used. In order to preserve data integrity, schema changes frequently require downtime and sensitive data movement techniques.

NoSQL:

MongoDB's adaptable design makes it simple to evolve and change data models without experiencing system failures. It is the best option for applications when the data model is updated and changed frequently since it easily handles schema changes.

Comparison based on Use Cases:

The following is the comparison between SQL and NoSQL according to (Andreas Meier, 2019):

SQL:

MySQL is appropriate for data storage, financial systems, and e-commerce platforms that have structured data and must strictly follow to predetermined schemas.

NoSQL:

Applications that require great scalability and flexibility, such content management systems, real-time analytics, and Internet of Things (IoT) applications, are well suited for MongoDB. These applications handle massive volumes of unstructured or semi-structured data.

Given the growing volume of auction data and the potential growth of online auction services, MongoDB's scalability and capacity to manage complex data structures without compromising performance make it an ideal choice. Furthermore, because of its support for dynamic schema and horizontal scaling, it can be easily modified to meet the changing needs of Bolton Auction House's expanding operations (Copeland, 2013).

Comparison based on the Query Operations:

The following is the comparison between SQL and NoSQL according to (Andreas Meier, 2019):

SQL:

Bolton Auction House is able to carry out complex tasks on structured data, such as retrieving, filtering, and aggregating data, because of MySQL's strong querying capabilities. Effective data retrieval and manipulation are made possible by its support for a variety of joins, subqueries, and analytical procedures. This is essential for managing the structured data pertaining to lots, bidders, staff, and auctions. Because of its standardised syntax and strong query optimisation, SQL is an asset for doing complex analytics on auction house data and generating insightful results.

NoSQL:

Text search, aggregation pipelines, and document-based queries are all comprehensive supported by MongoDB's query language. This improves Bolton Auction House's ability to manage unstructured data with dynamic schemas. The auction company can derive significant insights from its different data sets by using its efficient method of processing unstructured data, which includes performing CRUD operations on complex, nested documents and arrays.

Because of MongoDB's support for aggregation pipelines and real-time analytics, the auction house can conduct quick and flexible queries in order to extract insightful information from the frequently changing data. This adaptability and agility are essential for staying up to date with customer demands and industry developments as well as for making data-driven, well-informed decisions during auction events.

Limitations of SQL:

While SQL databases such as MySQL have their advantages, particularly in comparison to MongoDB, they might not be able to entirely meet the specific needs of the current Bolton Auction House scenario. The following justifies SQL's limitations in this context (geeksforgeeks, 2022):

- **Rigidity with Unstructured Data** : It might be difficult to manage the varied and unstructured data that is frequently seen in the operations of the auction house since SQL databases, like MySQL, require established schemas and structured data. Modifications to the schema require careful preparation and may be disruptive, which might impede the dynamic management of data.

- **Limited scalability for complicated Data**: Performance may be impacted by scaling the database to handle the growing amount of unstructured data and dynamic relationships.

- **Schema Alterations and Migration Issues:** Adding new data characteristics or altering an existing schema in a MySQL database may be tedious and complicated task, especially when working with unstructured or semi-structured data types. This rigidity in schema updates could make it more difficult for the auction house to rapidly adapt to evolving data needs.

- **Challenge with real-time analytics**: SQL databases may experience issues with processing large-scale data aggregation operations or complicated real-time analytics, particularly when dealing with rapid-changing data of the auction. Large-scale dataset queries and complex join queries may cause response times to lag and negatively affect user experience in general.

- **Restricted Horizontal Scalability:** When the Bolton Auction House's data expands rapidly, MySQL's vertical scaling strategy may run into issues. It may be difficult and expensive to scale MySQL to meet the growing data and operational needs.

Conclusion

In conclusion, based on specific needs for the given scenario of Bolton Auction, both MySQL and MongoDB have special advantages and work well for various use cases. While MongoDB's flexible schema and scalability make it an appealing choice for applications dealing with unstructured data and rapid schema expansion, MySQL offers good data integrity and is best suited for applications with structured data and complicated joins. Given its capacity to handle the dynamic nature of auction-related data and its potential for scalability without affecting performance and data integrity, MongoDB seems to be a more suitable option, especially in the context of the Bolton Auction System's requirement to manage complex auction data while also accommodating potential future scaling requirements.

(PORTFOLIO COMPONENT 2)

EXPLORING CURRENT AND EMERGING DATABASE TECHNOLOGIES FOR FUTURE INFORMATION SYSTEMS

With the rapidly evolving digital market, the cutting-edge database technology improve the services for the organisations. Bolton Auction House seeks to investigate the world of modern and developing database technologies beyond conventional RDBMS (MySQL) and NoSQL (MongoDB) in order to enhance the experiences of its sellers and customers. With consideration for their possible advantages, applications, and drawbacks, this whitepaper attempts to provide an in-depth examination for established and developing database technologies that meet the demands of the business for both in the future.

Creating Landscape for database technologies

According to (Lorincz & Huljić, 2020), although document-based data has shown to benefit from MongoDB, other NoSQL database formats have unique properties that are appropriate for a variety of applications.

Key-Value Stores: Key-value stores, such as Redis and DynamoDB, are efficient for handling big amounts of straightforward data and are great for fast data retrieval (amazon, n.d.).

Wide-Column Stores: Real-time applications and big data analytics benefit greatly from column-oriented databases like Cassandra and HBase, which are efficient at processing large datasets and analytical workloads (amazon, n.d.).

New and Emerging Database technologies

According to (Heudecker & Adrian, 2014), apart from the well-known NoSQL databases, a number of innovative database technologies are becoming more and more popular:

Graph databases: These are useful for social networks, fraud detection, and recommendation systems because they can handle complex relationships between data points. Examples of graph databases are Neo4j and Amazon Neptune (oracle, 2022).

Time-Series Databases: Time-series databases, such as Influx DB and Timescale DB, are essential for managing time-stamped data and are used in financial analysis, monitoring systems, and Internet of Things applications (amazon, 2023).

NewSQL Databases: Designed for high-performance transactional applications, NewSQL databases, such as Volt DB and Cockroach DB, combine the consistency of traditional SQL databases with the scalability of NoSQL (mdpi, n.d.).

Multi-Model Databases: These databases, which include Orient DB and Arango DB, enable several data models and give users the adaptability and freedom to work with different types of data within a single database system (Demurjian & Hsiao, 2002).

Evaluation of Database Technologies for Bolton Auction House

Considerations for Implementation :

According to (Kleppmann, 2017), the following elements need to be taken into account when choosing an appropriate database technology:

Scalability: The database's capacity to accommodate growing user loads and the quantities of data without sacrificing performance.

Flexibility: The database's capacity to support a variety of data formats and dynamic data structures.

Consistency and Availability: Making sure the database always stays accessible to users while preserving data consistency.

Security and Compliance: Observing strict guidelines for data security and following sector-specific laws.

Cost-Efficiency: Weighing the possible advantages of the database against the costs of implementation and upkeep.

Recommendation for Bolton Auction House

It is recommended that Bolton Auction House implement a multi-model database that is capable of handling a wide range of data types such as text, numbers, photos, and more data with efficiency. By doing this, the company would be able to simplify its data management procedures and guarantee an efficient and integrated handling of a variety of data types.

According to (Demurjian & Hsiao, 2002), the multi-model database gives strong support for complex interactions in addition to the flexibility needed to handle various data types. Bolton Auction House can efficiently handle complex relationships between buyers, sellers, and auction objects by using a graph database component. This feature improves knowledge of the auction ecosystem as a whole, which helps with customer relationship management and decision-making.

Furthermore, the multi-model framework's incorporation of a time-series database will make it possible to monitor changes in market dynamics and bidding patterns over time (Andreas Meier, 2019). For the auction house to obtain important insights on customer behaviour, market preferences, and demand changes, this skill is essential. Through the examination of past data patterns, the company may optimise its entire business operations by making well-informed judgements on pricing tactics, inventory control, and auction scheduling.

With Bolton Auction House's particular requirements, Arango DB is the suggested database solution. According to (Fowler, 2015), key-value, document, and graph data models are all supported by Arango DB, a multi-model database that uses a single database engine. It gives the auction house the flexibility it needs to manage its complex data requirements in an effective manner by allowing it to handle a variety of data formats. Arango DB's integrated graph database features make it possible to efficiently manage intricate interactions between bids, sellers, and products (arangodb, 2023). Furthermore, the organisation can easily analyse market dynamics and bidding patterns because of its support for time-series data processing, which makes data-based choices and strategic planning.

Bolton Auction House may take advantage of a complete database solution that meets its various data management demands by implementing ArangoDB into action. This will improve the company's customer experience, operational efficiency, and competitive edge in the auction market.

Analysis regarding the approach

NoSQL databases provide a number of benefits over traditional SQL databases in the context of Bolton Auction House, especially when taking into account the complex and varied nature of auction data and the requirement for scalable, adaptable, and high-performance data management (Fowler, 2015). For Bolton Auction House, a NoSQL database might be more appropriate in the following ways:

Flexible Data Modelling: ArangoDB and other NoSQL databases offer flexible data modelling features that enable the management and storing of a variety of data types, including semi-structured and unstructured data. This flexibility allows complex data structures to be stored without strict schema restrictions, which is essential for managing the variety of information related to auction items, bidders, and sellers.

Scalability and Performance: Bolton Auction House can efficiently manage escalating user loads and massive data volumes because of NoSQL databases' horizontal scalability design. The flexibility to scale horizontally guarantees that the database can handle growing data storage and processing requirements without sacrificing speed as the auction house develops and brings in more bids and sellers.

Complex Relationship Management: NoSQL technology makes it easier to manage the complex relationships that exist between bidders, sellers, and auction objects since it natively supports graph databases. This capacity is critical for comprehending the complex relationships that exist within the auction ecosystem, allowing the auction house to optimise its inventory management, bidding tactics, and relationship with customers through comprehensive data analysis.

Agile Development and Adaptability: NoSQL databases facilitate agile development techniques, which enable quicker iteration and adaption to shifting business requirements. According to (Sharma & Tripathi, 2023) ,the adaptability of NoSQL technology allows for the quick implementation of data model modifications and the addition of new data types as the auction industry develops and new data requirements appear. This guarantees that the database stays in line with the changing needs of the auction house and its customers.

ArangoDB is a NoSQL database that integrates document, graph, and key-value functionality into a single, integrated platform. It is adaptable and scalable. It was created to meet the increasingly complicated demands of contemporary data administration, providing a strong solution for managing many data kinds and intricate interactions within of a single database engine (Fowler, 2015). Arango DB's versatile query language and flexible data modelling features make it ideal for a wide range of use cases, such as social networks, content management systems, online applications, and more.

For companies like Bolton Auction looking for a versatile and effective database solution, ArangoDB is an excellent choice because of its powerful query language, scalability capabilities, and unique combination of key-value, document, and graph database functionalities. Because of its performance, adaptability, and simplicity of use, it can be used in a variety of application settings, enabling developers to create scalable, reliable, and successful data-driven systems (Fowler, 2015).

Model Flexibility for Data

Users may interact with key-value pairs, JSON documents, and graph data within a single database system because of Arango DB's support for diverse data models. Because of this versatility, developers may store and handle various kinds of data without requiring several database systems, which streamlines the data management process as a whole. Depending on the demands of their particular applications, users may easily transition between data models, guaranteeing maximum effectiveness and performance.

Multiple-Model Method

Arango DB's multi-model approach makes it easier to integrate diverse data models seamlessly, offering a unified data management solution for a range of application situations. ArangoDB simplifies development and lowers operational costs by merging the features of key-value, document, and graph databases, eliminating the headaches of managing several databases. This coherent methodology improves data integrity and streamlines data retrieval, making it easier for developers to create scalable and reliable systems.

Key-features of Arango DB

According to (Bayat, 2021), the following are the key features of Arango dB:

Document Store : Arango DB’s key feature enables users to store JSON documents and work with data using an adaptable and user-friendly query language. Applications handling semi-structured or unstructured data would benefit most from this feature, which offers effective methods for storing and retrieving data.

Graph Database : ArangoDB makes managing complex connections between data types easier by providing native support for graph data. Because of its sophisticated graph traversal and modification capabilities, it may be used in applications like fraud detection systems, recommendation engines, and social networks that call for in-depth analysis of interrelated data.

Query Language (AQL) : Users may easily accomplish complicated data retrieval and manipulation tasks with Arango DB’s query language, AQL. Graph traversals, joining, and filters are just a few of the many query types that AQL provides, giving developers the flexibility to describe complicated data operations clearly and effectively. Complex data retrieval and analysis are made possible by Arango DB’s robust query language, AQL (ArangoDB Query Language). The database's effective indexing and querying techniques maximise query performance, guaranteeing that Bolton Auction House can instantly glean insightful information from its data to support strategic planning and well-informed decision-making.

Replication & Scalability : ArangoDB has built-in scalability mechanisms that allow for horizontal scaling in response to increasing user demands and data volumes. Ensuring fault tolerance and data availability in distributed contexts, it facilitates automated data replication and sharding.

High Performance : ArangoDB offers low latency and excellent efficiency for data operations because of its native multi-threading architecture and effective indexing techniques. It is appropriate for applications that need real-time data processing and analysis since it is optimised to handle complicated queries and big datasets.

Strong Consistency : The ACID (Atomicity, Consistency, Isolation, Durability) compliance are provided by ArangoDB, guaranteeing data integrity and dependability for crucial auction transactions. This feature, which offers a reliable and safe platform for carrying out auction-related operations, is crucial for preserving the belief and trust of bidders and sellers

ArangoDB for Bolton Auction:

Considering Bolton Auction House's specific requirements, it is imperative to highlight additional elements necessary for the effective deployment of ArangoDB:

Flexibility and Customizability: Arango DB's adaptability to the particular requirements of the auction house is essential. Custom data models and customised database system functions can streamline data processing and administration and offer administrators and users a more customised experience.

Real-time Analytics and Reporting: By utilising Arango DB's real-time analytics and reporting features, valuable information into bidder behaviour, item preferences, and auction patterns may be obtained. The auction house may optimise income and client satisfaction by optimising auction methods, improving user experiences, and making data-driven choices through the implementation of powerful analytics tools.

Data Backup and Recovery Procedures: To guarantee data integrity and reduce the chance of data loss, thorough data backup and recovery procedures must be established. The auction house's vital data is protected by implementing frequent backups and efficient recovery methods, which also avert any possible interruptions in the case of unanticipated occurrences or system breakdowns.

Interaction with Third-Party products and Services: The auction house's operational efficiency may be improved by taking into account Arango DB’s smooth interface with a variety of third-party products and services. A more comprehensive approach to data management and analysis may be facilitated by integrating with CRM systems, data visualisation tools, and auction management software. These integrations can also increase data accessibility and expedite operations.

Comprehensive Training and assistance: Arango DB’s effective adoption and use by the auction house's technical staff depends on the provision of extensive training and ongoing support. Facilitating access to specialised training programmes, workshops, and support channels can enable the team to fully utilise Arango DB’s capabilities, efficiently handle technical issues, and guarantee a seamless and effective database management procedure.

Bolton Auction House may optimise the advantages of ArangoDB by integrating these factors into the implementation plan, guaranteeing a stable, expandable, and flexible database solution that meets the auction house's unique needs and long-term expansion goals.

Conclusion

To meet its varied data management requirements, Bolton Auction House should take into account a blend of multi-model, graph, and time-series databases in its search for a reliable and flexible database solution. By adopting these cutting-edge technologies, the business will be positioned as an innovative leader in the auction market and its information system's scalability and efficiency will be guaranteed. Bolton Auction House can improve its services, increase client happiness, and keep a competitive advantage in the ever-changing digital market by coordinating database technology like Arango DB with its company goals.

REFERENCES

- amazon. (2023). *amazon*. Retrieved from aws.amazon.com: https://aws.amazon.com/timestream/
- amazon. (n.d.). *amazon*. Retrieved from hoosing-an-aws-nosql-database/types-of-nosql-databases.html: https://docs.aws.amazon.com/whitepapers/latest/choosing-an-aws-nosql-database/types-of-nosql-databases.html
- Andreas Meier, M. K. (2019). *SQL & NoSQL Databases Models, Languages, Consistency Options and Architectures for Big Data Management.* Springer Fachmedien Wiesbaden.
- arangodb. (2023). *arangodb*. Retrieved from docs.arangodb: https://docs.arangodb.com/3.11/about-arangodb/
- Bayat, A. (2021). *medium*. Retrieved from arangodb-features: https://medium.com/@ali.bayat/arangodb-features-review-7db4d72824b3
- Copeland, R. (2013). *MongoDB Applied Design Patterns: Practical Use Cases with the Leading Nosql Database.* O'Reilly; Illustrated edition.
- Demurjian, S., & Hsiao, D. (2002). The multi model database system. *Eighth Annual International Phoenix Conference on Computers and Communications. 1989 Conference Proceedings.* Scottsdale, AZ, USA: IEEE.
- Fowler, A. (2015). Evaluating ArangoDB. In A. Fowler, *No SQL for Dummies* (p. 456). Dummies Tech.
- geeksforgeeks. (2022). *geeksforgeeks.org*. Retrieved from mongodb-embedded-documents: https://www.geeksforgeeks.org/mongodb-embedded-documents/
- geeksforgeeks. (2022). *geeksforgeeks.org*. Retrieved from advantages-and-disadvantages-of-sql: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-sql/
- Heudecker, N., & Adrian, M. (2014, February 28). *gartner.* Retrieved from www.gartner.com: https://www.gartner.com/en/documents/2673515
- Kleppmann, M. (2017). *Designing Data-Intensive Applications.* O'Reilly Media, Inc.
- Lorincz, J., & Huljić, V. (2020). Transforming Product Catalogue Relational into Graph Database: a Performance Comparison. *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO).* Opatija, Croatia: IEEE.
- mdpi. (n.d.). *mdpi*. Retrieved from https://www.mdpi.com/1999-5903/15/1/10#: https://www.mdpi.com/1999-5903/15/1/10#:~:text=On%20the%20other%20hand%2C%20we,terms%20of%20scalability%20and%20performance.

- mongodb. (2023). *mongodb*. Retrieved from manual/reference: https://www.mongodb.com/docs/manual/reference/operator/aggregation/unwind/
- mongodb. (2023). *mongodb*. Retrieved from schema-validation: https://www.mongodb.com/docs/manual/core/schema-validation/
- mongodb. (2023). *mongodb*. Retrieved from embedded-mongodb: https://www.mongodb.com/basics/embedded-mongodb#:~:text=Embedded%20documents%20are%20an%20efficient,only%20when%20they're%20worthwhile.
- oracle. (2022). *oracle*. Retrieved from autonomous-database/what-is-graph-database: https://www.oracle.com/uk/autonomous-database/what-is-graph-database/#:~:text=of%20graph%20databases-,Graph%20Database%20Defined,are%20not%20equipped%20to%20do.
- Sharma, G., & Tripathi, V. (2023). A systematic analysis of trending NOSQL database tools and techniques: A survey. *AIP Conference Proceedings* (p. 2782). Scopus.
- Valade, J. (2004). PHP and MySQL For Dummies. (p. 464). For Dummies.
- w3schools. (2023). *w3schools*. Retrieved from MySQL: https://www.w3schools.com/MySQL/default.asp

APPENDIX 1 – Part 1 MySQL Code

```
CREATE DATABASE IF NOT EXISTS boltonauction;

USE boltonauction;

DROP TABLE IF EXISTS `Staff` CASCADE;

DROP TABLE IF EXISTS `Seller` CASCADE;

DROP TABLE IF EXISTS `Bidder` CASCADE;

DROP TABLE IF EXISTS `Auction` CASCADE;

DROP TABLE IF EXISTS `Lot` CASCADE;



CREATE TABLE `Staff`

(

StaffID         VARCHAR(6) ,

Staffname               VARCHAR(100),

Staffaddress    VARCHAR(100),

Staffphone              VARCHAR(12),

Job_title               VARCHAR(50),


PRIMARY KEY (StaffID)

);
```

```
CREATE TABLE `Seller`

(

SellerID                INT(3) ,

SellerName              VARCHAR(100),

SellerAddress   VARCHAR(100),

SellerTelephone         VARCHAR(12),


PRIMARY KEY (SellerID)

);


CREATE TABLE `Bidder`

(

BidderID                VARCHAR(7) ,

Name                    VARCHAR(100),

Address                         VARCHAR(100),

Telephone               VARCHAR(12),


PRIMARY KEY (BidderID)

);



CREATE TABLE `Auction`

(

AuctionID               INT(1) ,

AuctionDate     DATE,

AuctioneerID    VARCHAR(6),

AssistantID             VARCHAR(6),

Location                VARCHAR(20),
```

```
PRIMARY KEY (AuctionID),

FOREIGN KEY (AuctioneerID) REFERENCES Staff(StaffID),

FOREIGN KEY (AssistantID) REFERENCES Staff(StaffID)


);


CREATE TABLE `Lot`

(

AuctionID               INT(1) ,

LotNumber               INT(1),

LotDescription VARCHAR(100),

ReservePrice    INT(4),

SellerID                INT(3),


PRIMARY KEY (LotNumber),

FOREIGN KEY (AuctionID) REFERENCES Auction(AuctionID),

FOREIGN KEY (SellerID) REFERENCES Seller(SellerID)

);


CREATE TABLE `Item`

(

ItemID                  INT(3),

LotNumber               INT(1),

ItemDescription         VARCHAR(100),


PRIMARY KEY (ItemID),

FOREIGN KEY (LotNumber) REFERENCES Lot(LotNumber)

);
```

[redacted]

```
CREATE TABLE `LotSale`

(

SaleID                    VARCHAR(4),

BidderID                  VARCHAR(7) ,

LotNumber                 INT(1),

WinningPrice   VARCHAR(100),


PRIMARY KEY (SaleID),

FOREIGN KEY (BidderID) REFERENCES Bidder(BidderID),

FOREIGN KEY (LotNumber) REFERENCES Lot(LotNumber)

);
```

```
INSERT INTO `Staff` (`StaffID`,`Staffname`,`Staffaddress`,`Staffphone`,`Job_title`)

VALUES

 ("MAR001",   "Moses Marsh",              "2738 Dui. St.",                    "07624
433836",            "Assistant."),

 ("LAR007",    "Kimberly Larson",     "808-6219 Ultricies Rd.",       "09057 617373",
          "Assistant"),

 ("POL008",    "Kane Pollard",               "559-9243 Accumsan St.",     "07812    584527",
                  "Auctioneer"),

 ("CON003",   "Lareina Conway",      "941-6642 Donec Street",      "07026 870247",
          "Assistant"),

 ("MCK050",   "Imani Mckay",                "218-4087 Turpis St.",          "08002      24786",
                  "Auctioneer."),

 ("GAR002",   "Paul Garner",            "914-2990 Ultrices. Street","07624 412036",
          "Assistant"),

 ("FIS010",      "Ulric Fisher",            "536-2804 Tempus Rd.",                "08002      72847",
                  "Auctioneer"),

 ("HUF001",   "Peter Huff",              "6227 Sem. Road",                         "01118    116234",
                  "Auctioneer"),

 ("RUS006",    "Isabella Rush",          "883-1373 Elit Rd.",                "08310 594915",
          "Assistant"),

 ("POT003",    "Armand Potts",               "Ap #555-697 Arcu. St.",      "05005      16540",
                  "Auctioneer");
```

```
INSERT INTO `Seller` (`SellerID`,`SellerName`,`SellerAddress`,`SellerTelephone`)
VALUES
 (100,        "Zorita Henry",          "Ap #583-2929 Lorem Street",
       "08828 113112"),
 (101,  "Xander Doyle",          "410-132 Quisque Av.",
       "08454 464456"),
 (102,  "Maia Ayala",       "Ap #709-6934 Magna. St.",          "038262 15402"),
 (103,  "Rigel Newton",          "525-919 Scelerisque Ave",           "07624
007584"),
 (104,  "Barry Guerrero",    "306-3955 Vehicula Ave",          "05004 535388"),
 (105,  "Norman Wyatt",          "9496 Dolor Avenue",           "01141
098208"),
 (106,  "Alexander Bryan",   "Ap #211-2548 A Road",           "03483
660677"),
 (107,  "Cody Maynard",          "Ap #688-7847 Phasellus Ave",        "070052
53475"),
 (108,  "Maryam Rivers",    "9870 Malesuada. Ave",           "08001
41187"),
 (109,  "Ferris Gardner",    "Ap #762-2785 Elit, Avenue",      "013421 29634");
 INSERT INTO `Bidder` (`BidderID`, `Name`,`Address`,`Telephone`)

VALUES

  ("NOR0001", "Fredericka Norton",   "P.O. Box 253, 9650 Morbi Ave",                  "05646
696460"),

  ("HES0001", "Bree Hester",             "Ap #658-6893 Fusce Av.",                        "01124
255551"),

 ("HEN0005", "Upton Henson",                    "4501 Nunc, Rd.",
           "08009 86476"),

 ("PEN0010", "Fleur Pennington",     "Ap #936-2164 Et Ave",
           "08002 76163"),

 ("COF0007", "Quemby Coffey",                   "Ap #399-949 Fusce Rd.",
           "08677 42177"),

 ("GAL0003", "Latifah Galloway",      "Ap #166-6167 Sed Rd.",
           "07624 896264"),

 ("POT0002", "Beau Potter",             "Ap #620-6061 Sed St.",
           "08454 690946"),

 ("BUR0001", "Jason Burgess",                   "Ap #277-7574 Ante. Rd.",
           "01697 791882"),

 ("KAU0003", "Hunter Kaufman",                  "703-9663 Ut, Av.",
           "07608 861515"),

 ("HOP0008", "Gavin Hopkins",                   "260-1548 Auctor St.",
           "08004 26439");
 INSERT INTO `Auction` (`AuctionID`, `AuctionDate`, `AuctioneerID`, `AssistantID`, `Location`)
VALUES
 (1,     '2023-07-11', "POL008", "FIS010", "Manchester"),
 (2,     '2023-07-08', "POL008", "FIS010", "Bury"),
 (3,     '2023-06-11', "POL008", "FIS010", "Burnley"),
 (4,     '2023-07-11', "MCK050", "GAR002", "Bolton"),
 (5,     '2023-06-13', "MCK050", "FIS010", "Preston"),
 (6,     '2023-04-13', "HUF001", "RUS006", "Preston");

 INSERT INTO `Lot` (`AuctionID`, `LotNumber`,`LotDescription`,`ReservePrice`,`SellerID`)
VALUES
 (1, 1, "17th & 15th Gold Jewellery", 100,104),
 (2, 6, "9ct gold cuff links",                   80, 106),
 (3, 4, "15th Century Toy Selection", 150, 108),
 (4, 5, "Flower Vase",                                  20, 102),
 (5, 3, "17th Century Blue China Tea Set",100, 104),
 (6, 7, "Earing and Broch Set",                      80, 104),
 (6, 8, "1960s toys",                                            30, 108);
 INSERT INTO `Item` (`ItemID`,`LotNumber`,`ItemDescription`)
VALUES
 (100,1,"17th century ruby ring"),
 (101,1,"15th century gold earing"),
 (102,1,"Edwardian Bird Broch"),
 (103,6,"16th century 9ct gold cuff links"),
 (104,4,"push along dog toy"),
 (105,4,"Victorian Doll"),
 (115,4,"Victorian Dolls House"),
 (106,5,"Floral Vase"),
 (107,3,"Blue Pattern China TeaPot, 17th Century"),
 (108,3,"Blue Pattern China Cup and Saucer, 17th Century"),
 (109,3,"Blue Pattern China Milk Jug, 17th Century"),
 (110,3,"Blue Pattern China Sugar Bowl, 17th Century"),
 (111,7,"12th century white gold earings"),
 (112,7,"Victoria Emerald Broch"),
 (113,8,"Jack in the Box, 1960"),
 (114,8,"spinning top, 1960");


 INSERT INTO LotSale (`SaleID`, `BidderID`, `LotNumber`, `WinningPrice`)
VALUES
 ("S001", "NOR0001", 1, 150),
 ("S002", "NOR0001", 6, 120),
 ("S003", "NOR0001", 5, 40),
 ("S004", "COF0007", 3, 175),
 ("S005", "HOP0008", 4, 210),
 ("S006", "HOP0008", 7, 95),
 ("S007", "HEN0005", 8, 95);
```

```
-- QUERIES

select * from boltonAuction.Staff;

select * from boltonAuction.Seller;

select * from boltonAuction.Bidder;

select * from boltonAuction.Auction;

select * from boltonAuction.Lot;

select * from boltonAuction.LotSale;

select*from boltonAuction.Item;


-- a) Write a query to show any lots which contain items which are Toys, you must display the
lot description and the date and location of the auction.


SELECT i.LotNumber, i.ItemDescription,  l.lotDescription, AuctionDate, Location

FROM item i

inner JOIN lot l

on ItemDescription LIKE '%toy%'

inner join auction a

on a.auctionID = l.auctionID

WHERE i.LotNumber= l.lotNumber;


-- (f.a) Query to fetch all those lots having "toy" in their description and return the
LotDescription, AuctionDate and Location.

SELECT l.LotDescription, a.AuctionDate, a.Location

FROM lot l

INNER JOIN

auction a

ON l.AuctionID = a.AuctionID

WHERE lotDescription LIKE '%toy%';
```

```
-- b.) Write a query to show seller name, telephone number, lot description and Reserve Price, where they have a lot where the reserve price is more than but less than £150 and only being auctioned in Manchester.


SELECT s.SellerName, s.SellerTelephone, l.LotDescription, l.ReservePrice, a.Location

FROM seller s

INNER JOIN lot l ON s.SellerID = l.SellerID

INNER JOIN auction a ON l.AuctionID = a.AuctionID

WHERE a.Location LIKE '%Manchester%' AND l.ReservePrice > 90 AND l.ReservePrice < 150;




-- c.) Write a query to show which customer has paid the highest total price for all successful bids, you should display the bidders name which should be labelled "Bidders Name" and the total price, which should be named as "Total Price".


SELECT b.Name, sum(ls.WinningPrice) as Total_price

FROM boltonAuction.Bidder b

INNER JOIN

lotSale ls

ON ls.BidderID = b.BidderID

Group by ls.BidderID

ORDER BY Total_price DESC LIMIT 1;
```

```
-- d.) Write a query to show the total number of successful bids per customer and their name, rename the total number of bids as "Total Bids".


SELECT  b.Name AS "Customer Name" , COUNT(ls.BidderID) as Total_Bids

FROM boltonAuction.Bidder b

INNER JOIN

lotSale ls

ON ls.BidderID = b.BidderID

Group by ls.BidderID

ORDER BY Total_Bids;


-- e.) Write a query to show the lowest reserve price for a seller, rename this column "Lowest Reserve Price" and show the sellers name, rename this column as "Seller Name".


SELECT DISTINCT s.SellerName AS "Seller Name", min(l.ReservePrice) AS Lowest_Reserve_Price

FROM seller s

INNER JOIN

lot l

ON l.sellerID = s.sellerID

GROUP BY s.sellerID

ORDER BY min(l.ReservePrice) ASC;


-- F.b Query to find auction with highest total revenue .

SELECT a.AuctionID, a.AuctionDate, a.Location,

  (SELECT SUM(ls.WinningPrice)

   FROM LotSale ls

   JOIN Lot l ON ls.LotNumber = l.LotNumber

   WHERE l.AuctionID = a.AuctionID) AS TotalRevenue

FROM Auction a
```

```
ORDER BY TotalRevenue DESC LIMIT 1;
```

```
-- Extra Queries

-- Query to fetch those auctions which have multiple lots.

SELECT AuctionID, COUNT(*) AS lot_count

FROM Lot

GROUP BY AuctionID

HAVING COUNT(*) > 1;



-- Query to find Seller with Multiple Lots

SELECT s.SellerName, a.AuctionID, COUNT(*) AS "Number of Lots"

FROM Seller s

JOIN Lot l ON s.SellerID = l.SellerID

JOIN (

    SELECT AuctionID, COUNT(*) AS lot_count

    FROM Lot

    GROUP BY AuctionID

    HAVING COUNT(*) > 1

) sub ON l.AuctionID = sub.AuctionID

JOIN Auction a ON l.AuctionID = a.AuctionID

GROUP BY s.SellerID, a.AuctionID;
```

APPENDIX 2 - Part 2 MongoDB Code

Schema Validation for Staff

```
db.createCollection("staff", {

  validator: {

    $jsonSchema: {

      required: ["Staffname", "Job_title"],

      properties: {

        StaffID: {

          bsonType: 'string',

          description: 'must be a string and required'

        },

        Staffname: {

          bsonType: 'string',

          description: 'must be a string and required'

        },

        Staffaddress: {

          bsonType: 'string',

          description: 'must be a string'

        },

        Staffphone: {

          bsonType: 'string',

          description: 'must be a string'

        },
            Job_title: {

               bsonType: 'string',

               description: 'must be a string and required'

            }

         }

      }

   },

   validationAction: 'error'

});
```

Schema Validation for Seller

```
db.createCollection("seller",{

          validator:{

                    $jsonSchema:{

                              required:["_id", "SellerName"],

                              properties:{

                                        "_id": {

                                                  bsonType: 'number',

                                                  description: 'must be a number and required'

                                        },

                                        "SellerName": {

                                                  bsonType: 'string',

                                                  description: 'must be a string and required'
                                        },
                                        "SellerAddress": {
                                                  bsonType: 'string',
                                                  description: 'must be a string'
                                        },
                                        "SellerTelephone": {
                                                  bsonType: 'string',
                                                  description: 'must be a string and required'
                                        }
                    }
                    }

          },
          validationAction: 'error'
})
```

Schema Validation for Bidder

```
db.createCollection("bidder",{
          validator:{
                    $jsonSchema:{
                              required:["_id", "Name"],
                              properties:{
                                        "_id": {
```

```
                                        bsonType: 'string',

                                        description: 'must be a string and required'

                              },

                              "Name": {

                                        bsonType: 'string',

                                        description: 'must be a string and required'

                              },

                              "Address": {

                                        bsonType: 'string',

                                        description: 'must be a string'

                              },

                              "Telephone": {

                                        bsonType: 'string',

                                        description: 'must be a string and required'

                              }

                    }

                    }


          },

          validationAction: 'error'

})
```

Schema Validation for Auction

```
db.createCollection("auction", {
```

```
validator: {
 $jsonSchema: {
  required: ["_id", "Location", "AuctionDate", "StaffID", "Lot", "Items"],
  properties: {
   _id: { bsonType: 'number', description: 'must be a number and required' },
   AuctionDate: { bsonType: 'string', description: 'must be a string and required' },
   StaffID: {
    bsonType: "array",
    description: "StaffID must be an array of strings",
    minItems: 1,
    items: { bsonType: "string" }
   },
   Location: { bsonType: 'string', description: 'must be a string' },
   Lot: {
    bsonType: 'array',
    required: ['LotNumber', 'LotDescription', 'ReservePrice', 'SellerID'],
    properties: {
     LotNumber: { bsonType: 'number' },
     LotDescription: { bsonType: 'string' },
     ReservePrice: { bsonType: 'string' },
     SellerID: { bsonType: 'number' }
    }
   },
    Items: {
      bsonType: 'array',
      description: 'must be an array',
      items: {
        bsonType: 'object',
        required: ['ItemID', 'ItemDescription'],
        properties: {
          ItemID: { bsonType: 'number' },
          ItemDescription: { bsonType: 'string' }
        }
      }
    }
   }
  }
 },
 validationAction: 'error'
});
```

Schema Validation for Lot Sale

```
db.createCollection("lotsale",{
validator:{
$jsonSchema:{
required:["_id", "BidderID", "LotNumber", "WinningPrice"],
properties:{

"_id": {

bsonType: 'string',

description: 'must be a string and required'

},

"BidderID": {

bsonType: 'string',

description: 'must be a string and required'

},

"LotNumber": {

bsonType: 'number',

description: 'must be a number'

},

"WinningPrice": {

bsonType: 'number',

description: 'must be a number and required'

}

}

}



},

validationAction: 'error'

})
```

Staff Collection JSON Script

```
[{"StaffID":"CON003",  "Staffname":"Lareina  Conway",  "Staffaddress":"941-6642  Donec
Street",         "Staffphone":"07026 870247",        "Job_title":"Assistant"},
 {"StaffID":"FIS010", "Staffname":"Ulric Fisher",      "Staffaddress":"536-2804  Tempus  Rd.",
                 "Staffphone":"08002 72847",         "Job_title":"Auctioneer"},
 {"StaffID":"GAR002", "Staffname":"Paul Garner",    "Staffaddress":"914-2990          Ultrices.
Street", "Staffphone":"07624 412036",        "Job_title":"Assistant"},
 {"StaffID":"HUF001", "Staffname":"Peter Huff",      "Staffaddress":"6227 Sem. Road",
                 "Staffphone":"01118 116234",        "Job_title":"Auctioneer"},
 {"StaffID":"LAR007", "Staffname":"Kimberly Larson","Staffaddress":"808-6219 Ultricies Rd.",
         "Staffphone":"09057 617373",        "Job_title":"Assistant"},
 {"StaffID":"MAR001", "Staffname":"Moses Marsh",          "Staffaddress":"2738   Dui.   St.",
                         "Staffphone":"07624 433836",        "Job_title":"Assistant."},
 {"StaffID":"MCK050", "Staffname":"Imani Mckay",  "Staffaddress":"218-4087   Turpis   St.",
                 "Staffphone":"08002 24786",         "Job_title":"Auctioneer."},
 {"StaffID":"POL008", "Staffname":"Kane Pollard",   "Staffaddress":"559-9243 Accumsan St.",
         "Staffphone":"07812 584527",        "Job_title":"Auctioneer"},
 {"StaffID":"POT003", "Staffname":"Armand Potts", "Staffaddress":"Ap  #555-697  Arcu.  St.",
         "Staffphone":"05005 16540",         "Job_title":"Auctioneer"},
{"StaffID":"RUS006", "Staffname":"Isabella Rush",   "Staffaddress":"883-1373 Elit Rd.",

         "Staffphone":"08310 594915",        "Job_title":"Assistant"}]
```

Seller Collection for JSON Script

```
[{"_id":100, "SellerName":"Zorita Henry",    "SellerAddress":"Ap  #583-2929  Lorem  Street",
         "SellerTelephone":"08828 113112"},
 {"_id":101, "SellerName":"Xander Doyle",    "SellerAddress":"410-132 Quisque Av.",
                 "SellerTelephone":"08454 464456"},
 {"_id":102, "SellerName":"Maia Ayala",      "SellerAddress":"Ap   #709-6934   Magna.   St.",
                 "SellerTelephone":"038262 15402"},
 {"_id":103, "SellerName":"Rigel Newton",    "SellerAddress":"525-919 Scelerisque Ave",
         "SellerTelephone":"07624 007584"},
 {"_id":104, "SellerName":"Barry Guerrero","SellerAddress":"306-3955 Vehicula Ave",
         "SellerTelephone":"05004 535388"},
 {"_id":105, "SellerName":"Norman Wyatt", "SellerAddress":"9496 Dolor Avenue",
                 "SellerTelephone":"01141 098208"},
 {"_id":106, "SellerName":"Alexander Bryan","SellerAddress":"Ap #211-2548 A Road",
         "SellerTelephone":"03483 660677"},
 {"_id":107, "SellerName":"Cody Maynard",  "SellerAddress":"Ap #688-7847  Phasellus  Ave",
         "SellerTelephone":"070052 53475"},
 {"_id":108, "SellerName":"Maryam Rivers", "SellerAddress":"9870 Malesuada. Ave",
                 "SellerTelephone":"08001 41187"},
{"_id":109,  "SellerName":"Ferris  Gardner","SellerAddress":"Ap  #762-2785  Elit,  Avenue",
          "SellerTelephone":"013421 29634"}]
```

Bidder Collection for JSON Script

```
[{"_id":"BUR0001", "Name":"Jason Burgess",          "Address":"Ap   #277-7574   Ante.   Rd.",
                    "Telephone":"01697 791882"},
 {"_id":"COF0007", "Name":"Quemby Coffey",        "Address":"Ap #399-949 Fusce Rd.",
                    "Telephone":"08677 42177"},
 {"_id":"GAL0003", "Name":"Latifah Galloway",       "Address":"Ap #166-6167 Sed Rd.",
                    "Telephone":"07624 896264"},
 {"_id":"HEN0005", "Name":"Upton Henson",                    "Address":"4501 Nunc, Rd.",
                              "Telephone":"08009 86476"},
 {"_id":"HES0001", "Name":"Bree Hester",             "Address":"Ap   #658-6893   Fusce   Av.",
                    "Telephone":"01124 255551"},
 {"_id":"HOP0008", "Name":"Gavin Hopkins",          "Address":"260-1548 Auctor St.",
                    "Telephone":"08004 26439"},
 {"_id":"KAU0003", "Name":"Hunter Kaufman",       "Address":"703-9663 Ut, Av.",
                              "Telephone":"07608 861515"},
 {"_id":"NOR0001", "Name":"Fredericka Norton", "Address":"P.O. Box 253, 9650 Morbi Ave",
          "Telephone":"05646 696460"},
 {"_id":"PEN0010", "Name":"Fleur Pennington",      "Address":"Ap #936-2164 Et Ave",
                    "Telephone":"08002 76163"},
 {"_id":"POT0002", "Name":"Beau Potter",            "Address":"Ap #620-6061 Sed St.",

                    "Telephone":"08454 690946"}]
```

Auction Collection for JSON Script

```
[
  {
     "_id": 1,
     "AuctionDate": "2023-07-11",
     "StaffID" : [
                         "POL008",
                         "FIS010"
                         ],
     "Location": "Manchester",
     "Lot": [{
       "LotNumber": 1,
       "LotDescription": "17th & 15th Gold Jewellery",
       "ReservePrice": 100,
       "SellerID": 104
```

```
    }],
    "Items": [
      {
        "ItemID": 100,
        "ItemDescription": "17th century ruby ring"
      },
      {
        "ItemID": 101,
        "ItemDescription": "15th century gold earing"
      },
      {
        "ItemID": 102,
        "ItemDescription": "Edwardian Bird Broch"
      }
    ]
  },
  {
    "_id": 2,
    "AuctionDate": "2023-07-08",
    "StaffID": [
                    "POL008",
                    "FIS010"
                    ],
    "Location": "Bury",
    "Lot": [{
      "LotNumber": 6,
      "LotDescription": "9ct gold cuff links",
      "ReservePrice": 80,
      "SellerID": 106
    }],
    "Items":[ {
      "ItemID": 103,
      "ItemDescription": "16th century 9ct gold cuff links"
    }]
  },
  {
    "_id": 3,
    "AuctionDate": "2023-06-11",
    "StaffID": [
                    "POL008",
                    "FIS010"
                    ],
    "Location": "Burnley",
    "Lot": [{
      "LotNumber": 4,
      "LotDescription": "15th Century Toy Selection",
      "ReservePrice": 150,
            "SellerID": 108
        }],
        "Items": [
            {
                "ItemID": 104,
                "ItemDescription": "push along dog toy"
            },
            {
                "ItemID": 105,
                "ItemDescription": "Victorian Doll"
            },
            {
                "ItemID": 115,
                "ItemDescription": "Victorian Dolls House"
            }
        ]
    },
    {
        "_id": 4,
        "AuctionDate": "2023-07-11",
        "StaffID": [
                                       "MCK050",
                                       "GAR002"
                                       ],
        "Location": "Bolton",
        "Lot": [{
            "LotNumber": 5,
            "LotDescription": "Flower Vase",
            "ReservePrice": 20,
            "SellerID": 102
        }],
        "Items": [{
            "ItemID": 106,
            "ItemDescription": "Floral Vase"
        }]
    },
    {
        "_id": 5,
        "AuctionDate": "2023-06-13",
         "StaffID": [
                                       "MCK050",
                                       "FIS010"
                                       ],
        "Location": "Preston",
        "Lot":[{
            "LotNumber": 3,
            "LotDescription": "17th Century Blue China Tea Set",
        "ReservePrice": 100,
        "SellerID": 104
    }],
    "Items": [
        {
            "ItemID": 107,
            "ItemDescription": "Blue Pattern China TeaPot, 17th Century"
        },
        {
            "ItemID": 108,
            "ItemDescription": "Blue Pattern China Cup and Saucer, 17th Century"
        },
        {
            "ItemID": 109,
            "ItemDescription": "Blue Pattern China Milk Jug, 17th Century"
        },
        {
            "ItemID": 110,
            "ItemDescription": "Blue Pattern China Sugar Bowl, 17th Century"
        }
    ]
},
{
    "_id": 6,
    "AuctionDate": "2023-04-13",
    "StaffID": [
                        "HUF001",
                        "RUS006"
                        ],
    "Location": "Preston",
    "Lot": [
        {
            "LotNumber": 7,
            "LotDescription": "Earing and Broch Set",
            "ReservePrice": 80,
            "SellerID": 104
        },
        {
            "LotNumber": 8,
            "LotDescription": "1960s toys",
            "ReservePrice": 30,
            "SellerID": 108
        }
    ],
    "Items": [
        {
            "ItemID": 111,
            "ItemDescription": "12th century white gold earings"
          },
          {
            "ItemID": 112,
            "ItemDescription": "Victoria Emerald Broch"
          },
          {
            "ItemID": 113,
            "ItemDescription": "Jack in the Box, 1960"
          },
          {
            "ItemID": 114,
            "ItemDescription": "spinning top, 1960"
          }
        ]
    }
]
```

LotSale Collection JSON Script

```
[{"_id":"S001", "BidderID":"NOR0001", "LotNumber":1, "WinningPrice":150},
 {"_id":"S002", "BidderID":"NOR0001", "LotNumber":6, "WinningPrice":120},
 {"_id":"S003", "BidderID":"NOR0001", "LotNumber":5, "WinningPrice":40},
 {"_id":"S004", "BidderID":"COF0007", "LotNumber":3, "WinningPrice":175},
 {"_id":"S005", "BidderID":"HOP0008", "LotNumber":4, "WinningPrice":210},
 {"_id":"S006", "BidderID":"HOP0008", "LotNumber":7, "WinningPrice":95},
 {"_id":"S007", "BidderID":"HEN0005", "LotNumber":8, "WinningPrice":95}]
```

Query A

```
db.auction.aggregate([

{ $unwind: "$Lot" },

{

$unwind: "$Items"

},

{

$match: {

"Items.ItemDescription": { $regex: /toy/i }

}
},
{
$project: {
_id:0,
LotNumber: "$Lot.LotNumber",
ItemDescription: "$Items.ItemDescription",
lotDescription: "$Lot.LotDescription",
AuctionDate: 1,
Location: 1
}
}
]);
```

Query B

```
db.auction.aggregate([
{
$unwind: "$Lot"
},
{
$match: {
$and: [
{ "Lot.ReservePrice": { $gt: 90, $lt: 150 } },
{ Location: "Manchester" }
]
}
},
{
$lookup: {
from: "seller",
localField: "Lot.SellerID",
foreignField: "_id",
as: "seller"
}
},
{
$unwind: "$seller"
},
{
$project: {
_id: 0,
SellerName: "$seller.SellerName",
Telephone: "$seller.SellerTelephone",
LotDescription: "$Lot.LotDescription",
ReservePrice: "$Lot.ReservePrice",
Location: "$Location"
}
}
]);
```

Query C

```
db.lotsale.aggregate([{$group:
{_id:"$BidderID",
Total_Winning_Price:{$sum:{$toDouble:"$WinningPrice"}
}
}},
{ $sort: { Total_Winning_Price: -1 } },  { $limit: 1 }])
```

Query D

```
db.lotsale.aggregate([
{
$group: {
_id: "$BidderID",
"Total Bids": { $sum: 1 }
}
},
{
$lookup: {
from: "bidder",
localField: "_id",
foreignField: "_id",
as: "bidder"
}
},
{ $unwind: "$bidder" },
{
$project: {
"bidder.Name": 1,
"Total Bids": 1,
_id: 0
}
}
]);
```

Query E

```
db.auction.aggregate([
{
```

```
$unwind: "$Lot"
},
{
$group: {
_id: "$Lot.SellerID",
"Lowest Reserve Price": { $min: "$Lot.ReservePrice" }
}
},
{
$lookup: {
from: "seller",
localField: "_id",
foreignField: "_id",
as: "seller"
}
},
{
$unwind: "$seller"
},
{
$project: {
_id: 0,
"Seller Name": "$seller.SellerName",
"Lowest Reserve Price": 1
}
}
]);
```

Query F-a.)

```
db.auction.aggregate([
{
```

```
$unwind: "$Lot"
},
{
$match: {
$or: [
{"Lot.LotDescription": /Toy/i},
{"Lot.Items.ItemDescription": /Toy/i}
]
}
},
{
$project: {
_id: 0,
"Lot.LotDescription": 1,
AuctionDate: 1,
Location: 1
}
}
])
```

Query F- b.)

```
db.auction.aggregate([
 {$lookup: {
    from: "lotsale",
    localField: "Lot.LotNumber",
    foreignField: "LotNumber",
    as: "lotSale"}},
 {$unwind: {
    path: "$lotSale"}},
 {
  $group: {_id: "$_id",
    AuctionID: { $first: "$_id" },
    AuctionDate: { $first: "$AuctionDate" },
    Location: { $first: "$Location" },
    TotalRevenue: { $sum: { $ifNull: [ { $toInt: "$lotSale.WinningPrice" }, 0 ] } }}},
 {$sort: {TotalRevenue: -1}
 },
 {
  $limit: 1
 }
]);
```