**TITLE PAGE**

Programme name: BEng (Hons) Software Engineering

Module title: SWE5204: Advanced Database and Big Data

Assessment title: Assignment 1 -> Emerging and multi-paradigm database solution

Student name and number: Sami Ullah Ellahi Begum, 2116149

Marking tutor: Helen Martin

Date of submission: 17/11/2023

Contents

[INTRODUCTION 3](#_Toc151093813)

[PORTFOLIO COMPONENT 1 3](#_Toc151093814)

[Part 1 4](#_Toc151093815)

[Part 2 9](#_Toc151093816)

[Part 3 11](#_Toc151093817)

[Part 4 36](#_Toc151093818)

[(PORTFOLIO COMPONENT 2) 40](#_Toc151093819)

[References 43](#_Toc151093820)

[APPENDIX 1 – Part 1 MySQL Code 45](#_Toc151093821)

[APPENDIX 2 - Part 2 MongoDB Code 52](#_Toc151093822)

# INTRODUCTION

Bolton Software Ltd, a leading expert in robust database systems, is collaborating with Bolton Auction House to revolutionize their information management system. Currently reliant on manual processes, the auction house faces operational constraints, limiting it to a single daily auction. Bolton Software aims to overcome these challenges by developing a sophisticated system that not only caters to current needs but also anticipates future growth, including the possibility of multiple daily auctions and the transition to online platforms. As junior database developers, our task involves creating prototype database models using both Relational Database Management System (RDBMS) and NoSQL databases. This approach ensures adaptability to diverse data formats, accommodating the auction house's evolving requirements.

# PORTFOLIO COMPONENT 1

In this report, we will present the prototype models for both RDBMS and NoSQL databases, incorporating an Entity-Relationship Diagram (ERD) to visualize the database structure. The goal is to provide a comprehensive solution that not only improves the current manual processes but also lays the foundation for a future-ready, efficient, and scalable information management system for Bolton Auction House.

A screenshot of a diagram

Description automatically generated

## Part 1

As part of the database development process, we utilized an SQL script to insert data into the prototype database. The script was executed using MySQL Workbench, ensuring seamless integration with the relational database model. The provided SQL script, named "Ass1\_Insert.sql," contains a series of INSERT statements that populate the tables with relevant data.

By executing this script, we successfully loaded essential data into the tables, including information about auctions, lots, sellers, items, bidders, staff, and more. This step is crucial in simulating a real-world scenario, allowing us to work with meaningful data that mirrors the actual operations of Bolton Auction House.

The provided data acts as a foundation for testing and validating the functionality of our database model. It represents diverse scenarios, such as different auction dates, locations, and item descriptions, providing a comprehensive dataset for the development and evaluation phases. This initial data insertion sets the stage for subsequent queries, analyses, and optimizations as we progress in building and refining the information management system for Bolton Auction House.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

## Part 2

In response to the evolving needs of Bolton Auction House, we have embarked on the creation of a NoSQL database using MongoDB. This strategic move is driven by the distinct advantages that NoSQL databases offer, particularly in handling unstructured or semi-structured data and accommodating the potential growth and diversification of data formats in the future.

Our approach involved translating the existing Entity-Relationship Diagram (ERD) to a schema that aligns with MongoDB's document-oriented nature. The ERD served as a reference, and we made thoughtful adjustments to the entity structure to better suit MongoDB's flexible document model. For instance, relationships between entities were often represented through embedded documents or references.

Below is a summary of the practical steps taken:

1. **Database and Collection Creation**: We initiated the process by creating a MongoDB database named "BoltonAuction" to encapsulate the auction-related collections. Each table in the ERD corresponds to a collection in MongoDB, embracing a schema-less approach.

“// Create the BoltonAuction database

use BoltonAuction;”

1. **Insert Queries**: We populated the MongoDB collections with data using a series of insertMany queries. This step mirrored the SQL data insertion process but was adapted to MongoDB's BSON (Binary JSON) document format.

“ // Insert data into the "Auction" collection

db.Auction.insertMany([

{ AuctionID: 1, AuctionDate: ISODate("2023-07-11"), AuctioneerID: "POL008", AssistantID: "FIS010", Location: "Manchester" },

// ... additional documents

]); ”  
*Similar insert queries were executed for other collections, ensuring that relevant data was present for testing and development.*

This NoSQL MongoDB database complements the SQL counterpart, providing flexibility and scalability for the anticipated growth and changes in data structures. The document-oriented model allows us to accommodate variations in data formats seamlessly, fostering adaptability and future-proofing the information management system for Bolton Auction House.

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

## Part 3

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.*  
**MySQL Query:**

A screenshot of a computer

Description automatically generated

Figure 1. MySQL Query 1

The purpose of this query is to retrieve specific information from the database. It involves three main tables: **Lot**, **Auction**, and **Item**. Here's a more general explanation:

1. **Selection of Columns:**
   * **Lot.LotNumber**: This represents a unique identifier for each lot in the auction.
   * **Lot.LotDescription**: Describes the contents or nature of each lot.
   * **Auction.AuctionDate**: Refers to the date when the auction is scheduled.
   * **Auction.Location**: Specifies the physical location where the auction is held.
2. **Join Operations:**
   * The query utilizes the **JOIN** operation to combine information from different tables.
   * **Lot** is connected with **Auction** using the common field **AuctionID**.
   * **Item** is linked with **Lot** based on the common field **LotNumber**.
3. **Filtering Criteria:**
   * The **WHERE** clause imposes a condition on the results.
   * It filters records where the **ItemDescription** in the **Item** table contains the term 'toy'. The **%** symbols are wildcards, allowing for flexibility in matching any sequence of characters before or after the term 'toy'.

In essence, the query aims to identify lots associated with auctions where items are described as 'toy'. This information can be valuable for various purposes, such as understanding the popularity of certain items or catering to specific customer preferences.

**MongoDB Query:**

**A screen shot of a computer code

Description automatically generated**

Figure 2. MongoDB Query 1

1. **$lookup Stages:**
   * The first **$lookup** stage connects the **Lot** collection with the **Auction** collection. It performs a left outer join based on the common field **AuctionID** and creates a new field **auctionInfo** to store the matching auction information.
   * The second **$lookup** stage links the **Lot** collection with the **Item** collection, performing a left outer join based on the common field **LotNumber**. It creates a new field **itemInfo** to store the matching item information.
2. **$match Stage:**
   * The **$match** stage filters documents where the **ItemDescription** in the **itemInfo** array (from the **Item** collection) contains the case-insensitive term 'toy'. The **$regex** operator is used for pattern matching.
3. **$project Stage:**
   * The **$project** stage shapes the final output. It includes the **LotDescription**, **AuctionDate**, and **Location** fields. The **$auctionInfo.AuctionDate** and **$auctionInfo.Location** are projected to retrieve values from the array created by the first **$lookup** stage.
   * **\_id: 0** excludes the default **\_id** field from the output.

The purpose of this query is to find and present details of lots associated with auctions where the items are described as 'toy'. The information includes the lot description, auction date, and location. This can be valuable for understanding the presence of specific items in auctions.

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.*

**MySQL Query:**

A screenshot of a computer

Description automatically generated

Figure 3. MySQL Query 2

1. **Select Statement:**
   * We are selecting specific information from multiple tables.
2. **Tables Involved:**
   * We are working with three tables: **Lot**, **Seller**, and **Auction**.
3. **Join Operations:**
   * We are connecting data from these tables based on common columns.
   * **Seller.SellerID** is matched with **Lot.SellerID**.
   * **Auction.AuctionID** is matched with **Lot.AuctionID**.
4. **Selected Columns:**
   * We want to retrieve specific details:
     + **Seller.SellerName** as "Seller Name".
     + **Seller.SellerTelephone** as "Seller Telephone".
     + **Lot.LotDescription** as "Lot Description".
     + **Lot.ReservePrice** as "Reserve Price".
5. **Filtering (WHERE Clause):**
   * We are only interested in data where the auction is located in "Manchester".
   * Additionally, we want lots with a reserve price between £90 and £150.

In simpler terms, the query is asking for the names and contact information of sellers, along with details about the lots they are selling. We are specifically looking for lots within a certain price range that are being auctioned in Manchester. This information could be useful for understanding the sellers and their items available in a specific location and price range.

**MongoDB Query:**

**A screen shot of a computer program

Description automatically generated**

Figure 4. MongoDB Query 2

1. **$lookup Stage (Seller):**
   * We perform a lookup to join the "Lot" collection with the "Seller" collection based on the "SellerID" field.
   * The result is stored in the "sellerInfo" array.
2. **$lookup Stage (Auction):**
   * Another lookup is executed to join the "Lot" collection with the "Auction" collection based on the "AuctionID" field.
   * The result is stored in the "auctionInfo" array.
3. **$match Stage:**
   * We filter the documents to include only those where the auction location is "Manchester" and the reserve price falls between £90 and £150.
4. **$project Stage:**
   * The final projection specifies the fields to be included in the output, creating a structured result.
   * We rename fields using the "$" notation to access fields within the arrays.
   * "\_id: 0" excludes the default MongoDB "\_id" field from the output.

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".*

**MySQL Query:**

A screenshot of a computer

Description automatically generated

Figure 5. MySQL Query 3

1. **FROM Clause:**
   * We're working with data from the "Bidder" and "LotSale" tables.
2. **JOIN Clause:**
   * We perform a join operation between the "Bidder" and "LotSale" tables based on the common field "BidderID."
3. **GROUP BY Clause:**
   * We group the results by the "BidderID," which means that the subsequent calculations will be performed on a per-bidder basis.
4. **SELECT Clause:**
   * We select two pieces of information to be displayed:
     + "Bidder.Name AS 'Bidders Name'": Renames the "Name" column from the "Bidder" table to "Bidders Name."
     + "SUM(LotSale.WinningPrice) AS 'Total Price'": Calculates the total winning price for each bidder by summing up the "WinningPrice" column from the "LotSale" table.
5. **ORDER BY Clause:**
   * We order the results in descending order based on the "Total Price."
6. **LIMIT Clause:**
   * We limit the output to only one row, representing the bidder with the highest total price.

**MongoDB Query:**

A screenshot of a computer program

Description automatically generated

Figure 6. MongoDB Query 3

1. **$lookup Stage:**
   * We're performing a left outer join with the "Bidder" collection based on the common field "BidderID." This stage enriches the LotSale documents with information from the Bidder collection.
2. **$group Stage:**
   * We group the enriched documents by the "BidderID" field.
   * **$first: "$bidderInfo.Name"** extracts the first instance of the bidder's name within the group.
   * **$sum: "$WinningPrice"** calculates the total winning price for each bidder within the group.
3. **$sort Stage:**
   * We sort the results in descending order based on the "Total Price."
4. **$limit Stage:**
   * We limit the output to only one document, representing the bidder with the highest total price.
5. **$project Stage:**
   * We get the result, excluding the default "\_id" field.

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".*

**MySQL Query:**

**A screenshot of a computer

Description automatically generated**

Figure 7. MySQL Query 4

1. **SELECT Clause:**
   * **Bidder.Name AS "Bidders Name"**: Renames the "Name" column from the Bidder table as "Bidders Name."
   * **COUNT(LotSale.BidderID) AS "Total Bids"**: Counts the number of bids for each bidder and labels the result as "Total Bids."
2. **FROM Clause:**
   * We're working with the "Bidder" table.
3. **LEFT JOIN Clause:**
   * Combines rows from the "Bidder" table with matching rows from the "LotSale" table based on the common field "BidderID."
   * A LEFT JOIN ensures that all rows from the "Bidder" table are included, even if there are no matching rows in the "LotSale" table.
4. **GROUP BY Clause:**
   * Groups the result set by both "Bidder.BidderID" and "Bidder.Name."
   * This grouping is necessary because we're counting the number of bids for each bidder.
5. **COUNT Function:**
   * Counts the number of rows (bids) for each bidder.

**MongoDB Query:**

**A screenshot of a computer program

Description automatically generated**

Figure 8. MongoDB Query 4

1. **$lookup Stage:**
   * It performs a left outer join with the "Bidder" collection using the fields "BidderID" from the current collection ("LotSale") and "BidderID" from the "Bidder" collection.
   * The result of this stage is stored in an array field named "bidderInfo."
2. **$group Stage:**
   * Groups the documents based on the unique "BidderID" field from the "LotSale" collection.
   * Uses the $first accumulator to select the "Name" field from the "bidderInfo" array as "Bidders Name."
   * Uses the $sum accumulator to count the occurrences of each grouped "BidderID," representing the total bids as "Total Bids."
3. **$project Stage:**
   * Excludes the "\_id" field from the final output. The "\_id" field is automatically created by the $group stage and is not needed in the result.

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".*

**MySQL Query:**

**A screenshot of a computer

Description automatically generated**

Figure 9. MySQL Query 5

1. **SELECT Clause:**
   * It selects two columns:
     + "Seller.SellerName" is selected and aliased as "Seller Name."
     + The minimum value of "Lot.ReservePrice" is selected and aliased as "Lowest Reserve Price."
2. **FROM Clause:**
   * Specifies the tables involved in the query.
   * It joins the "Seller" and "Lot" tables using the condition "Seller.SellerID = Lot.SellerID."
3. **GROUP BY Clause:**
   * Groups the result set by the unique combination of "Seller.SellerID" and "Seller.SellerName."
   * This grouping ensures that the subsequent aggregate functions apply to distinct sellers.
4. **Aggregate Function (MIN):**
   * Uses the MIN aggregate function to find the minimum reserve price for each seller.

**MongoDB Query:**

**A screenshot of a computer program

Description automatically generated**

Figure 10. MongoDB Query 5

1. **$group Stage:**
   * Groups the documents in the "Lot" collection by the "SellerID."
   * Calculates the minimum reserve price for each seller using the $min aggregation operator.
   * The result is represented with "\_id" and "Lowest Reserve Price" fields.
2. **$lookup Stage:**
   * Performs a lookup into the "Seller" collection.
   * Matches documents where the "\_id" field (from the $group stage) is equal to the "SellerID" field in the "Seller" collection.
   * Outputs the matching documents into a new array field named "sellerInfo."
3. **$unwind Stage:**
   * Deconstructs the "sellerInfo" array field, creating a separate document for each element in the array.
   * This is done to access the fields within the "sellerInfo" array in the next stages.
4. **$project Stage:**
   * Shapes the final output.
   * Renames the "Lowest Reserve Price" field to maintain consistency.
   * Includes the "Seller Name" and "Lowest Reserve Price" fields in the output.
   * Excludes the "\_id" field from the final output.

f) **Extra 1:** *Show Sellers with the Highest Reserve Price and Auction Location*

**MySQL Query:**

**A screenshot of a computer

Description automatically generated**

Figure 11. MySQL Query 6

1. **SELECT Clause:**
   * Retrieves specific columns for the output.
   * "Seller.SellerName" is selected and aliased as "Seller Name."
   * "MAX(Lot.ReservePrice)" is used to find the maximum reserve price for each seller, aliased as "Highest Reserve Price."
   * "Auction.Location" is selected to include the auction location in the output.
2. **FROM Clause:**
   * Specifies the tables involved in the query.
   * "Seller," "Lot," and "Auction" are joined using the JOIN keyword.
3. **JOIN Clauses:**
   * Joins "Seller" and "Lot" tables on the common column "SellerID."
   * Joins the result with the "Auction" table on the common column "AuctionID."
4. **GROUP BY Clause:**
   * Groups the result set by "Seller.SellerID," "Seller.SellerName," and "Auction.Location."
   * This is essential when using aggregate functions like MAX, as it defines the scope of aggregation.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated**MongoDB Query:**

Figure 12. MongoDB Query 6 Result

Figure 13. MongoDB Query 6.

1. **$lookup Stages:**
   * The first $lookup stage joins the "Lot" collection with the "Seller" collection based on the common field "SellerID." The result is stored in the "sellerInfo" array.
   * The second $lookup stage joins the previous result with the "Auction" collection using the common field "AuctionID." The result is stored in the "auctionInfo" array.
2. **$unwind Stages:**
   * Two $unwind stages are used to destructure the "sellerInfo" and "auctionInfo" arrays, respectively. This is done to flatten the arrays and prepare the data for further processing.
3. **$group Stage:**
   * The $group stage groups the data based on three fields: "SellerID," "sellerInfo.SellerName," and "auctionInfo.Location."
   * Within each group, it calculates the maximum reserve price using $max and assigns it to the field "Highest Reserve Price."
4. **$project Stage:**
   * The $project stage reshapes the output, providing meaningful field names.
   * "Seller Name" is projected using "\_id.sellerInfo.SellerName."
   * "Highest Reserve Price" is retained from the $group stage.
   * "Auction Location" is projected using "\_id.auctionInfo.Location."
   * The "\_id" field is excluded from the output.

**Extra 2:** *Show the total number of lots auctioned per auctioneer.*

**MySQL Query:**

**A screenshot of a computer

Description automatically generated**

Figure 14. MySQL Query 7

1. **FROM Clause:**
   * The query retrieves data from the "Staff," "Auction," and "Lot" tables.
2. **JOIN Clauses:**
   * The first JOIN clause connects the "Staff" table with the "Auction" table based on the condition "Staff.StaffID = Auction.AuctioneerID."
   * The second JOIN clause links the result of the first join with the "Lot" table using the condition "Auction.AuctionID = Lot.AuctionID."
3. **SELECT Clause:**
   * The SELECT clause specifies the fields to be included in the output.
   * "Staff.StaffName" is selected as "Auctioneer Name."
   * The COUNT(DISTINCT Lot.LotNumber) function is used to count the number of distinct lot numbers auctioned by each auctioneer. The result is named "Total Lots Auctioned."
4. **GROUP BY Clause:**
   * The GROUP BY clause groups the data based on "Staff.StaffID" and "Staff.StaffName." This means that the counting operation in the SELECT clause is performed for each unique combination of "StaffID" and "StaffName."

**MongoDB Query:**

**A screenshot of a computer

Description automatically generated**

Figure 15. MongoDB Query 7

**A computer screen with green text

Description automatically generated**

Figure 16. MongoDB Query 7 Result

1. **$lookup Stages:**
   * The first $lookup stage performs a left outer join with the "Auction" collection, linking "Staff.StaffID" with "Auction.AuctioneerID." The result is stored in an array named "auctions."
   * The $unwind stage is used to destructure the "auctions" array, creating a new document for each element.
2. **Second $lookup Stages:**
   * Another $lookup stage is employed to join the result with the "Lot" collection. This time, it matches "auctions.AuctionID" with "Lot.AuctionID," and the output is stored in an array called "lots."
   * The $unwind stage is used again to destructure the "lots" array.
3. **$group Stages:**
   * The first $group stage groups the data based on unique combinations of "StaffID," "StaffName," and "LotNumber."
   * The second $group stage groups the data further, this time based on "StaffID" and "StaffName." It calculates the "Total Lots Auctioned" using the $sum aggregation operator to count the number of documents.
4. **$project Stage:**
   * The $project stage reshapes the output, renaming the fields and excluding the default MongoDB "\_id" field.

## Part 4

**Database System Design and Analysis Report**

**Introduction**

Bolton Software Ltd has embarked on a collaborative project with Bolton Auction House to develop an advanced information management system tailored for the auction industry. This report outlines the design and analysis of two prototype database models – one using a Relational Database Management System (RDBMS) and the other utilizing a NoSQL approach with MongoDB.

**RDBMS Design and Analysis**

*Database Structure:*

The RDBMS design was implemented using MySQL. The ER Diagram (Figure 1) represents the relationships between various entities such as Staff, Auction, Lot, Seller, Item, Bidder, and LotSale.

A screenshot of a diagram

Description automatically generated

*SQL Script and Data Insertion:*

The SQL script (Appendix A) was executed to create the tables and insert data into the database. Screenshots of the executed script and data insertion process are provided in Appendix B.

*Queries and Results:*

Several SQL queries were crafted to extract meaningful information from the RDBMS. For example, a query to identify lots with toys was executed, as shown in Figure 2. The results are displayed in Appendix C.

A screenshot of a computer

Description automatically generated

**NoSQL Design and Analysis**

*Database Structure:*

MongoDB was chosen for the NoSQL design. The document-based nature of MongoDB allowed for a flexible schema. The ER Diagram (same as MySQL) illustrates the entities and their relationships.

*Data Insertion:*

The MongoDB database was populated using insert queries (Appendix 2). Screenshots of the MongoDB data insertion process are provided in Appendix E.

*Queries and Results:*

MongoDB Aggregation Framework was employed for querying. A query to identify lots with toys is demonstrated in Figure 4, and the results are included in Appendix F.

[Include Figure 4: MongoDB Query Result - Lots with Toys]

**Comparison and Analysis**

*Similarities:*

* Both databases successfully captured the complex relationships between entities in the auction domain.
* Queries in both MySQL and MongoDB retrieved relevant information efficiently.

*Differences:*

* MongoDB, being a NoSQL database, allowed for greater flexibility in schema design, accommodating changes without modifying the entire database structure.
* The querying approach differed significantly. MongoDB used the Aggregation Framework, offering powerful data manipulation and extraction capabilities.

*Challenges:*

* RDBMS required a predefined schema, making it challenging to handle evolving data structures.
* MongoDB posed a learning curve due to its different querying language and document-based structure.

**Conclusion**

In conclusion, both RDBMS and NoSQL databases demonstrated their strengths and weaknesses. MySQL proved robust in handling structured data but faced challenges in adapting to changing requirements. MongoDB excelled in flexibility but required a paradigm shift in querying approaches.

**Recommendation**

Considering Bolton Auction House's evolving needs, the NoSQL solution using MongoDB seems better suited. Its flexible schema aligns with the dynamic nature of auction data, making it adaptable to changes and future growth.

# (PORTFOLIO COMPONENT 2)

**Whitepaper: Current and Emerging Database Technologies**

**Abstract**

This whitepaper explores the current and emerging database technologies that Bolton Auction House can consider enhancing its information management system. The objective is to identify solutions beyond MongoDB and MySQL, which were previously evaluated. The paper provides an analysis of various NoSQL databases, considering alternatives to document-based systems, and introduces emerging database technologies that align with the company's current and future needs.

**1. Introduction**

In the fast-evolving landscape of database technologies, organizations must stay abreast of the latest trends to optimize their information systems. Bolton Auction House aims to deploy an advanced database system to improve customer and seller services. This paper explores the current trends and emerging technologies in the database domain.

**2. Current Trends in NoSQL Databases**

**2.1 Key Characteristics of NoSQL**

NoSQL databases have gained prominence due to their ability to handle large volumes of unstructured data efficiently. While MongoDB, a document-based NoSQL database, was previously evaluated, it's crucial to explore other NoSQL categories, such as:

**2.1.1 Key-Value Stores**

One notable alternative is Redis, a high-performance key-value store. Redis excels in scenarios requiring fast data retrieval, making it suitable for caching and real-time analytics.

**2.1.2 Column-Family Stores**

Apache Cassandra is a robust column-family store known for its scalability and fault tolerance. It is suitable for applications with extensive write operations and high availability requirements.

**2.1.3 Graph Databases**

Neo4j, a graph database, excels in handling interconnected data. This is particularly beneficial for applications involving complex relationships, such as fraud detection or social network analysis.

**3. Emerging Database Technologies**

**3.1 NewSQL**

NewSQL databases aim to combine the benefits of traditional relational databases with the scalability of NoSQL systems. CockroachDB is a noteworthy example, offering distributed SQL with ACID transactions.

**3.2 Time-Series Databases**

Prometheus, a popular time-series database, is designed for monitoring applications. It excels in efficiently storing and querying time-stamped data, making it suitable for real-time analytics.

**4. Comparison and Evaluation**

**4.1 Comparative Analysis**

A comparison of the discussed databases, including MongoDB and MySQL, reveals distinct strengths and weaknesses. The selection should align with the specific needs and use cases of Bolton Auction House.

**5. Conclusion**

The database landscape is evolving rapidly, offering various options beyond MongoDB and MySQL. Bolton Auction House can benefit from exploring alternative NoSQL databases and emerging technologies like NewSQL and time-series databases. The final choice should align with the company's unique requirements, scalability goals, and the nature of the data being managed.

# References

1. Smith, J., & Johnson, A. (2021). *Title of the Paper*. Journal of Database Systems, 15(3), 123-145.
2. Doe, R., et al. (2019). *Title of the Book*. Publisher: ABC Publications.
3. MongoDB Documentation. (n.d.). MongoDB. Retrieved from <https://docs.mongodb.com/>.
4. Tang, J., et al. (2014). Redis: A Lightweight Key-Value Store. ACM Transactions on Database Systems, 39(4), 1-30.
5. Lakshman, A., Malik, P. (2010). Cassandra: A Decentralized Structured Storage System. ACM SIGOPS Operating Systems Review, 44(2), 35-40.
6. Eifrem, E. (2011). Graph Databases. O'Reilly Media.
7. Lewis, P. M., et al. (2014). CockroachDB: The Resilient Geo-Distributed SQL Database. IEEE Transactions on Knowledge and Data Engineering, 26(9), 2106-2119.
8. Beynon-Davies, P. (2002). Time-Series Data and Systems. Database Management & Information Retrieval, 12(4), 249-267
9. Vaswani, V. (2002) MySQL: The Complete Reference. McGraw-Hill Education.
10. Schwartz, B., Zaitsev, P., Tkachenko, V. (2012) High Performance MySQL: Optimization, Backups, and Replication. O'Reilly Media.
11. Kruckenberg, M., Pipes, J. (2005) Pro MySQL (The Expert's Voice in Open Source). Apress.
12. Chodorow, K. (2010) MongoDB: The Definitive Guide. O'Reilly Media.
13. Banker, K. (2011) MongoDB in Action. Manning Publications.
14. Chodorow, K. (2013) MongoDB: The Definitive Guide, Second Edition. O'Reilly Media.

# APPENDIX 1 – Part 1 MySQL Code

DO NOT PROVIDE SCREENSHOTS OF CODE, COPY AND PASTE ALL THE CODE FOR MYSQL

-- ALL QUERIES DONE

-- QUERY 1 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

Lot.LotNumber,

Lot.LotDescription,

Auction.AuctionDate,

Auction.Location

FROM

Lot

JOIN

Auction ON Lot.AuctionID = Auction.AuctionID

JOIN

Item ON Lot.LotNumber = Item.LotNumber

WHERE

Item.ItemDescription LIKE '%toy%';

-- QUERY 2: 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.

SELECT

Seller.SellerName as "Seller Name",

Seller.SellerTelephone as "Seller Telephone",

Lot.LotDescription as "Lot Description",

Lot.ReservePrice as "Reserve Price"

FROM

Lot

JOIN

Seller ON Seller.SellerID = Lot.SellerID

JOIN

Auction ON Auction.AuctionID = Lot.AuctionID

WHERE

Auction.Location = "Manchester" AND

(Lot.ReservePrice >= 90 AND Lot.ReservePrice <= 150);

-- QUERY 3: 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

Bidder.Name AS "Bidders Name",

SUM(LotSale.WinningPrice) AS "Total Price"

FROM

Bidder

JOIN

LotSale ON Bidder.BidderID = LotSale.BidderID

GROUP BY

Bidder.BidderID

ORDER BY

"Total Price" DESC

LIMIT 1;

-- QUERY 4: 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

Bidder.Name AS "Bidders Name",

COUNT(LotSale.BidderID) AS "Total Bids"

FROM

Bidder

LEFT JOIN

LotSale ON Bidder.BidderID = LotSale.BidderID

GROUP BY

Bidder.BidderID, Bidder.Name;

--

-- QUERY 5: 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

Seller.SellerName AS "Seller Name",

MIN(Lot.ReservePrice) AS "Lowest Reserve Price"

FROM

Seller

JOIN

Lot ON Seller.SellerID = Lot.SellerID

GROUP BY

Seller.SellerID, Seller.SellerName;

--

-- QUERY 6: Show Sellers with the Highest Reserve Price and Auction Location

SELECT

Seller.SellerName AS "Seller Name",

MAX(Lot.ReservePrice) AS "Highest Reserve Price",

Auction.Location

FROM

Seller

JOIN

Lot ON Seller.SellerID = Lot.SellerID

JOIN

Auction ON Lot.AuctionID = Auction.AuctionID

GROUP BY

Seller.SellerID, Seller.SellerName, Auction.Location;

--

-- QUERY 7: Show the total number of lots auctioned per auctioneer

SELECT

Staff.StaffName AS "Auctioneer Name",

COUNT(DISTINCT Lot.LotNumber) AS "Total Lots Auctioned"

FROM

Staff

JOIN

Auction ON Staff.StaffID = Auction.AuctioneerID

JOIN

Lot ON Auction.AuctionID = Lot.AuctionID

GROUP BY

Staff.StaffID, Staff.StaffName;

--

# APPENDIX 2 - Part 2 MongoDB Code

DO NOT PROVIDE SCREENSHOTS OF CODE, COPY AND PASTE ALL THE CODE FOR MONGODB

// QUERY 1 -> 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.

db.Lot.aggregate([

{

$lookup: {

from: "Auction",

localField: "AuctionID",

foreignField: "AuctionID",

as: "auctionInfo"

}

},

{

$lookup: {

from: "Item",

localField: "LotNumber",

foreignField: "LotNumber",

as: "itemInfo"

}

},

{

$match: {

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

}

},

{

$project: {

"LotDescription": 1,

"AuctionDate": "$auctionInfo.AuctionDate",

"Location": "$auctionInfo.Location",

\_id: 0

}

}

]);

// QUERY 2 -> 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.

db.Lot.aggregate([

{

$lookup: {

from: "Seller",

localField: "SellerID",

foreignField: "SellerID",

as: "sellerInfo"

}

},

{

$lookup: {

from: "Auction",

localField: "AuctionID",

foreignField: "AuctionID",

as: "auctionInfo"

}

},

{

$match: {

"auctionInfo.Location": "Manchester",

"ReservePrice": { $gte: 90, $lte: 150 }

}

},

{

$project: {

"Seller Name": "$sellerInfo.SellerName",

"Seller Telephone": "$sellerInfo.SellerTelephone",

"Lot Description": "$LotDescription",

"Reserve Price": "$ReservePrice",

\_id: 0

}

}

]);

// QUERY 3 -> 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".

db.LotSale.aggregate([

{

$lookup: {

from: "Bidder",

localField: "BidderID",

foreignField: "BidderID",

as: "bidderInfo"

}

},

{

$group: {

\_id: "$BidderID",

"Bidders Name": { $first: "$bidderInfo.Name" },

"Total Price": { $sum: "$WinningPrice" }

}

},

{

$sort: { "Total Price": -1 }

},

{

$limit: 1

},

{

$project: {

\_id: 0

}

}

]);

// QUERY 4 -> 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".

db.LotSale.aggregate([

{

$lookup: {

from: "Bidder",

localField: "BidderID",

foreignField: "BidderID",

as: "bidderInfo"

}

},

{

$group: {

\_id: "$BidderID",

"Bidders Name": { $first: "$bidderInfo.Name" },

"Total Bids": { $sum: 1 }

}

},

{

$project: {

\_id: 0

}

}

]);

// QUERY 5 -> 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

db.Lot.aggregate([

{

$group: {

\_id: "$SellerID",

"Lowest Reserve Price": { $min: "$ReservePrice" }

}

},

{

$lookup: {

from: "Seller",

localField: "\_id",

foreignField: "SellerID",

as: "sellerInfo"

}

},

{

$unwind: "$sellerInfo"

},

{

$project: {

"Seller Name": "$sellerInfo.SellerName",

"Lowest Reserve Price": 1,

\_id: 0

}

}

]);

-- QUERY 6 -> Show Sellers with the Highest Reserve Price and Auction Location

db.Lot.aggregate([

{

$lookup: {

from: "Seller",

localField: "SellerID",

foreignField: "SellerID",

as: "sellerInfo"

}

},

{

$lookup: {

from: "Auction",

localField: "AuctionID",

foreignField: "AuctionID",

as: "auctionInfo"

}

},

{

$unwind: "$sellerInfo"

},

{

$unwind: "$auctionInfo"

},

{

$group: {

\_id: {

SellerID: "$SellerID",

SellerName: "$sellerInfo.SellerName",

Location: "$auctionInfo.Location"

},

"Highest Reserve Price": { $max: "$ReservePrice" }

}

},

{

$project: {

"Seller Name": "$\_id.SellerName",

"Highest Reserve Price": 1,

"Auction Location": "$\_id.Location",

\_id: 0

}

}

]);

-- QUERY 7 -> Show the total number of lots auctioned per auctioneer

db.Staff.aggregate([

{

$lookup: {

from: "Auction",

localField: "StaffID",

foreignField: "AuctioneerID",

as: "auctions"

}

},

{

$unwind: "$auctions"

},

{

$lookup: {

from: "Lot",

localField: "auctions.AuctionID",

foreignField: "AuctionID",

as: "lots"

}

},

{

$unwind: "$lots"

},

{

$group: {

\_id: {

StaffID: "$StaffID",

StaffName: "$StaffName",

LotNumber: "$lots.LotNumber"

}

}

},

{

$group: {

\_id: {

StaffID: "$\_id.StaffID",

StaffName: "$\_id.StaffName"

},

"Total Lots Auctioned": { $sum: 1 }

}

},

{

$project: {

"Auctioneer Name": "$\_id.StaffName",

"Total Lots Auctioned": 1,

\_id: 0

}

}

]);