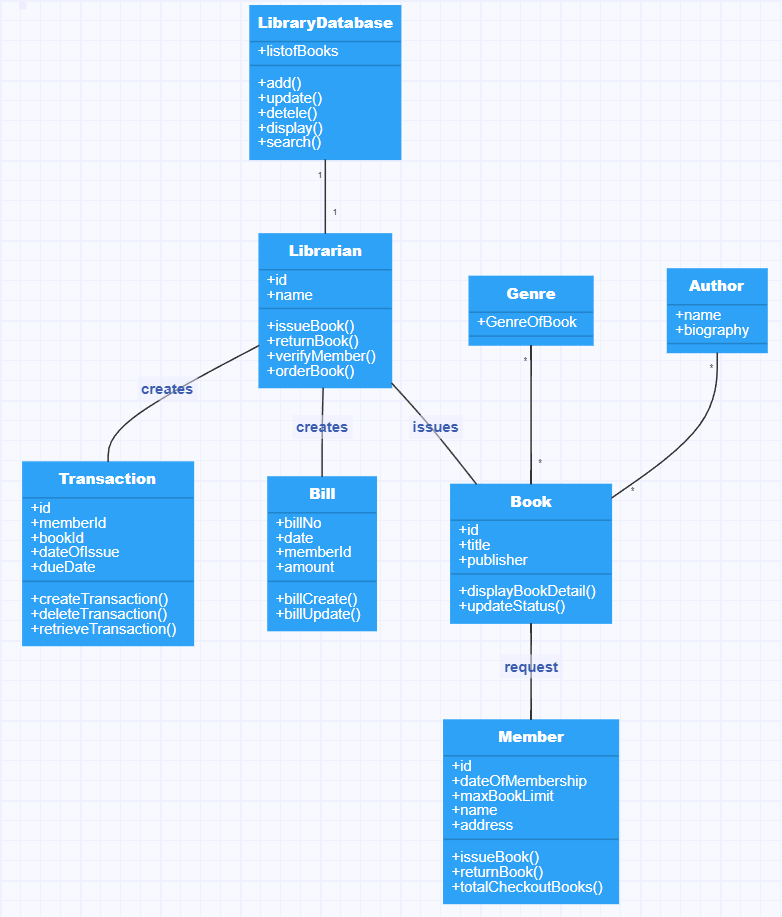
# \Task One;

UML Diagram of Library Management System.



# Task Two

|  |  |
| --- | --- |
| First dropping all the tables to ensure the database is empty. |  |
| Creating table named as “LibraryDatabse” and “Librarian” |  |
| Inserting data into table “Librarian” |  |
| Creating table named “Book” |  |
| Inserting data into table “Book” |  |
| Creating table named “Member” |  |
| Inserting data into table “Member” |  |
| Creating table named “Bill” |  |
| Inserting data into table “Bill” |  |
| Creating table named “Transaction” |  |
| Inserting data into table “Transaction” |  |

# Task Three

|  |  |
| --- | --- |
| Creating Collections in database name library. |  |
| Displaying the created Collections in MongoDB. |  |
| Inserting into Collection “Bill” |  |
| Displaying the inserted data in MongoDB. |  |
| Inserting into Collection “Book” |  |
| Displaying the inserted data in MongoDB. |  |
| Inserting into Collection “Librarian” |  |
| Displaying the inserted data in MongoDB. |  |
| Inserting into Collection “Member” |  |
| Displaying the inserted data in MongoDB. |  |
| Inserting into Collection “Transaction” |  |
| Displaying the inserted data in MongoDB. |  |

# Task Three:

|  |  |
| --- | --- |
| **Query a: A join of three or more tables** | |
| **SQL Code** | **MongoDB code** |
|  |  |
| ***Screenshots of output*** | |
|  |  |
| **Discussion:**  Both the **SQL** and **MongoDB** queries are best at their own side. For Oracle and MongoDB, I have implemented both the left outer join and inner join. Both Oracle and Mongo may join three or more separate tables, as was demonstrated above. In MongoDB, the process of filtering out documents that have the necessary information is known as a lookup. I joined four tables in Oracle named as "Member", "Bill" and "Book" and "Transaction". Oracle's pre-existing building elements for table connections made integrating two or more tables quite easy. Contrarily, MongoDB employed collections to hold structured data that could be quickly integrated from several collections using various techniques. However, maintaining the approach was difficult because the objective data had to cover a wide variety of categories. To combine the data as needed, a JOIN statement in Oracle was executed rather easily. This data was stored in a single collection, that means that many of their procedures were unnecessary in MongoDB since they were all stored in a single, huge container with other data. **(Ilić, Kopanja, Zlatković, Trajković, & Ćurguz, 2021)** | |

|  |  |
| --- | --- |
| **Query b: Nested Table** | |
| **SQL Code** | **MongoDB code** |
|  | db.Library.insert({libraryId: 1,Genre:["Action","Adventure","Horror"]}) |
| ***Screenshots of output*** |  |
|  |  |
| **Discussion:**  Oracle's nested functionality was implemented, but an exception occurred when entering data into the nested table. However, I layered the table data in MongoDB while maintaining the nested shape of the data. In the address column of the Genre collection, the same library ID’s addresses two data. I created and inserted data in nested table inside table name Library which contains Genre table. As contrast to Oracle, MongoDB makes it simpler to utilize nested table comparisons. In MongoDB, we may use nested to store many pieces of data in the same property. **(Martins, Tomé, Wanzeller, Sá, & Abbasi , 2021)** | |

|  |  |
| --- | --- |
| **Query c: Timestamps** | |
| **SQL Code** | **MongoDB code** |
| INSERT INTO Bill (billNo,createDate,memberId,amount) VALUES (1,CURRENT\_TIMESTAMP, 5 ,3000); | db.timestamp.insert(  {'id': 1,'date':new ISODate("1999-07-13T20:30:22Z") ,'currentDate':new Date()  })  //Aggregate  db.timestamp.aggregate([  {  $project:{HH: {$hour: "$date"},  MM:{$minute:"$date"},  SS:{$second:"$date"},  }  }  ]) |
| ***Screenshots of output*** |  |
|  |  |
| **Discussion:**  I used timestamp for the temporal feature in Oracle, and ISODate for the temporal feature in MongoDB. By filtering out dates of Book issue of the larger than a certain threshold, the result showed the issue date of book where we can apply due date. Data is stored as timestamps in a variety of formats, including year, month, hour, minute, and second. Similarly, ISODate is used in MongoDB to filter dates. Along with the date and time formats, ISODate also provides the date form. If we maintain the ISODate and send the date into it, output is automatically thrown along with the time and the date that we have supplied. For fast creating a new MongoDB datetime, it is a useful tool. Dates are recorded as signed 64-bit integers in the database, reflecting milliseconds since the Unix epoch. You can use ISODate to construct a genuine Date object in the database that you can use to run operations and rapidly calculate values for. Every time you needed to deal with a date, you would need to carry strings if you used a string as the date. | |

|  |  |
| --- | --- |
| **Query d: Query using Roll up** | |
| **SQL Code** | **MongoDB code** |
|  |  |
| ***Screenshots of output*** |  |
|  |  |
| **Discussion:**  As seen in the examples for rollup above, Oracle includes built-in OLAP methods that may be used to modify data obtained and add extra columns. Each of the statements in the sample had a similar use case, which involved calculating the total amount and categorizing the total amount males and females borrowing the books.  Utilizing more specific Oracle instructions for data execution and thorough data analysis using a mix of table data, relevant statistics may be provided in MongoDB along with grouping, sorting, and adding using aggregate. There are no comparable built-up OLAP operations in MongoDB.  However, same operations might be carried out utilizing a fruitless search and aggregation method. The need for a thorough understanding of the database structure makes query execution more challenging. For a specific set of dimensions, a SELECT query can utilize ROLLUP to calculate numerous layers of subtotals. The use of a cube provides a useful way to collect calculated and saved data with similar characteristics, such as dimension, aggregation rules, and so on. In addition to reducing the number of rows that the system must read for each user query, partitioning reduces the number of aggregations that the OLAP platform must compute on each cube refreshing. **(Liu, 2020)** | |