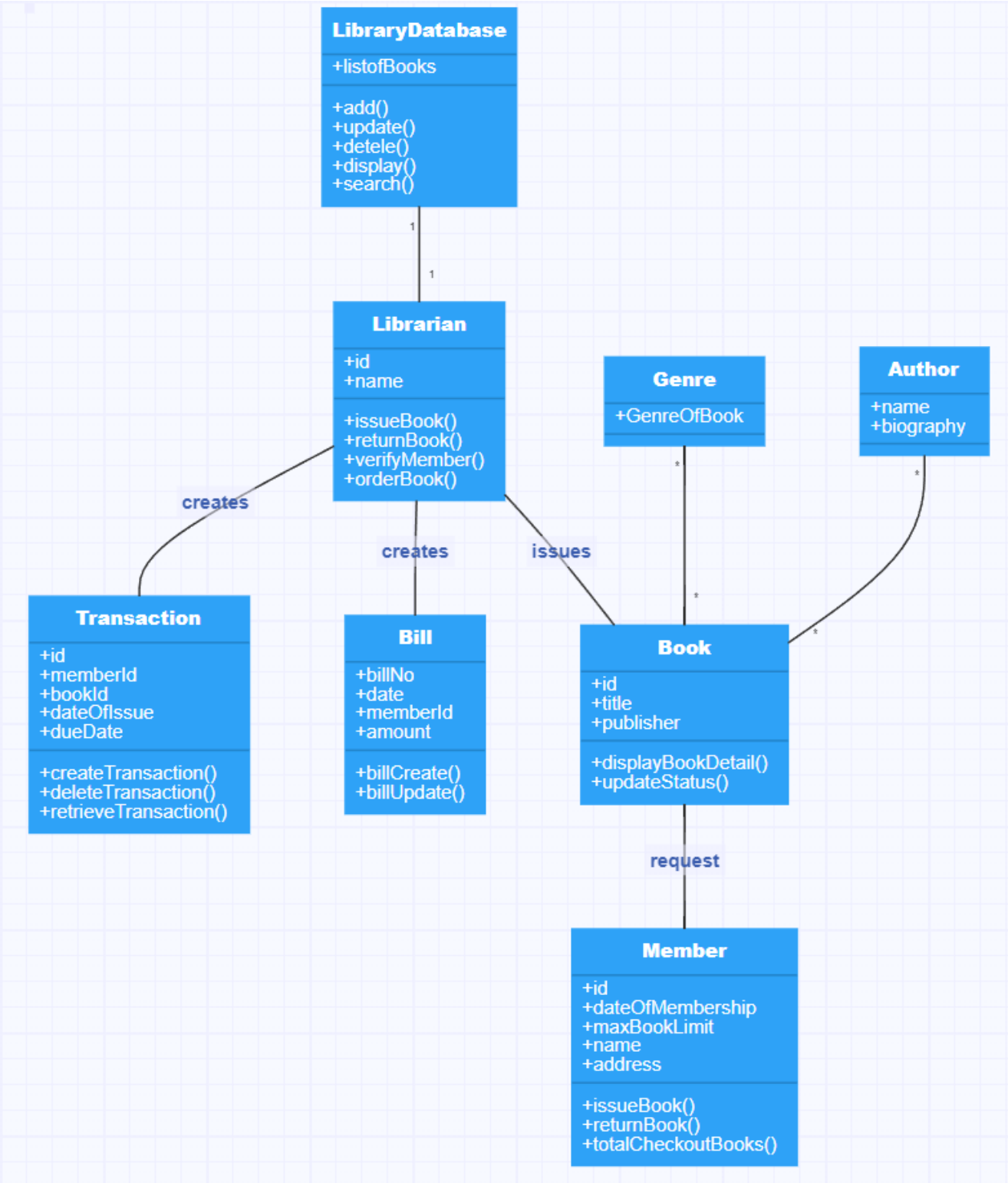
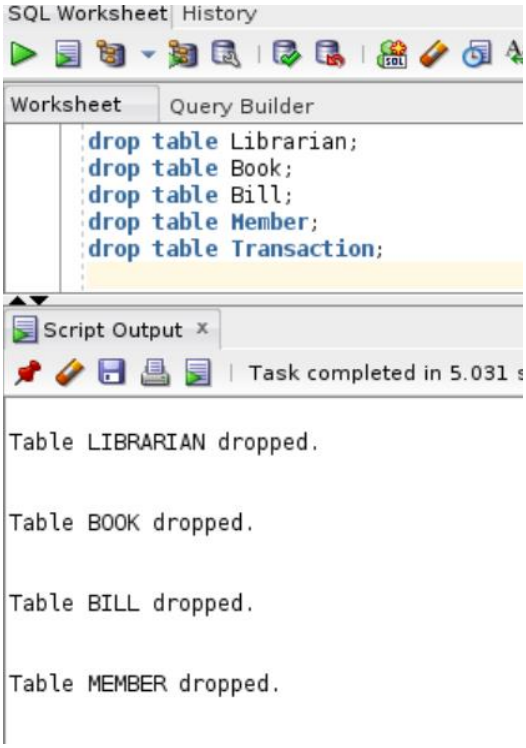
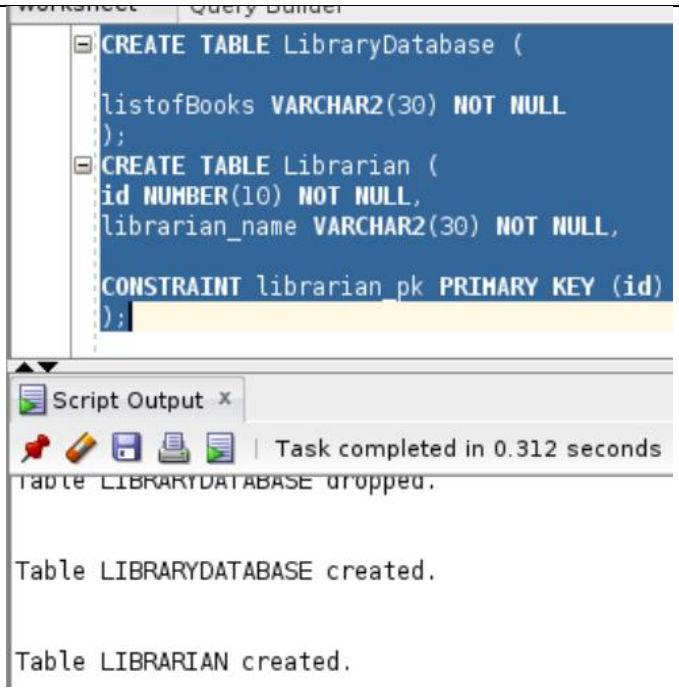
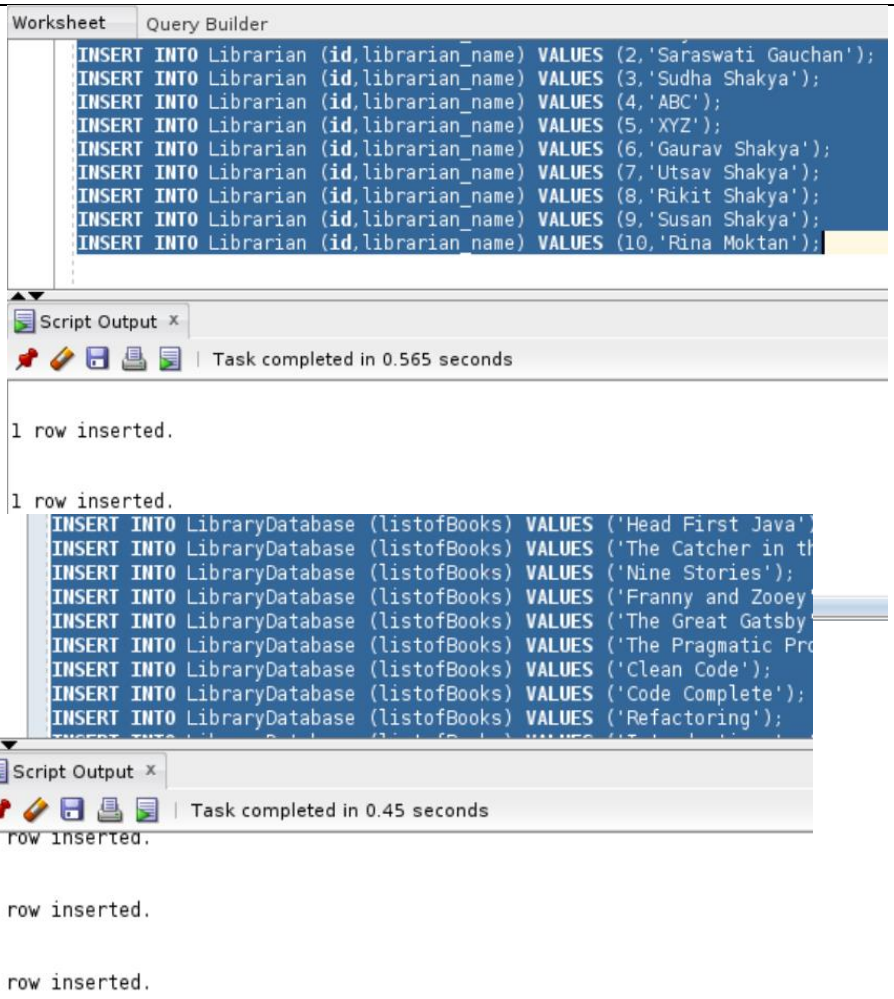
\Task One;

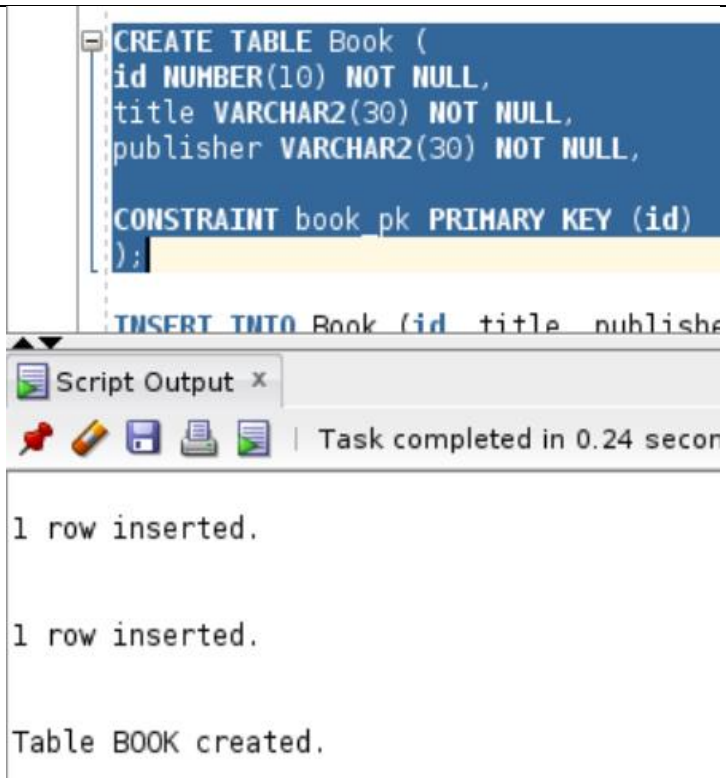
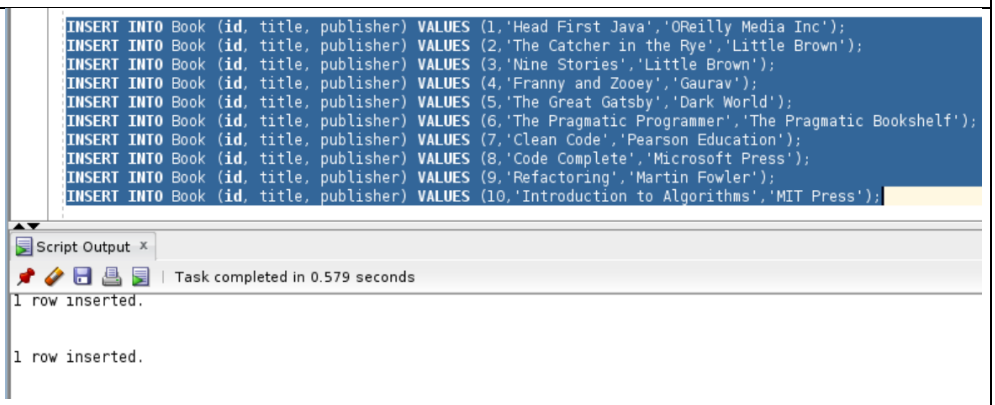
UML Diagram of Library Management System.

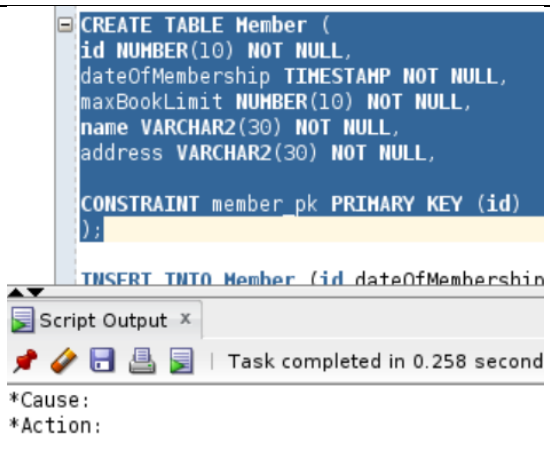
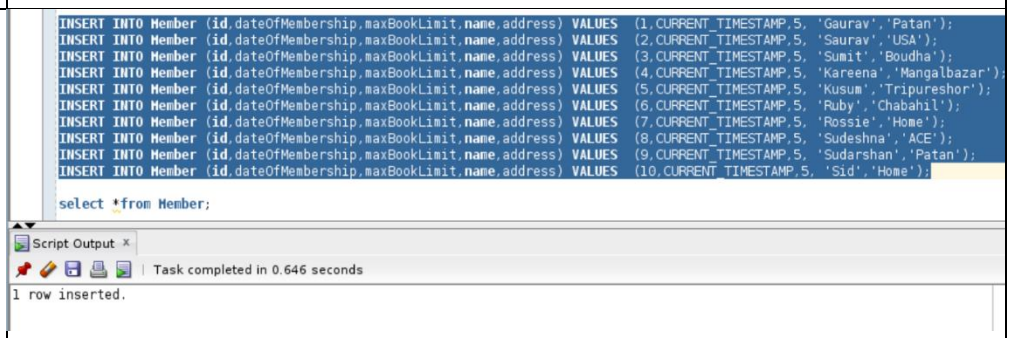
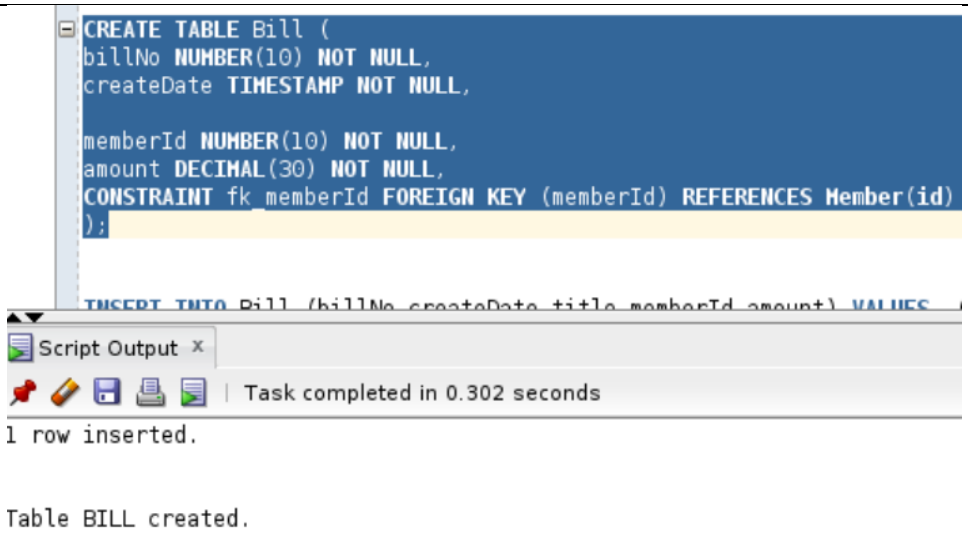


Task Two

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”	

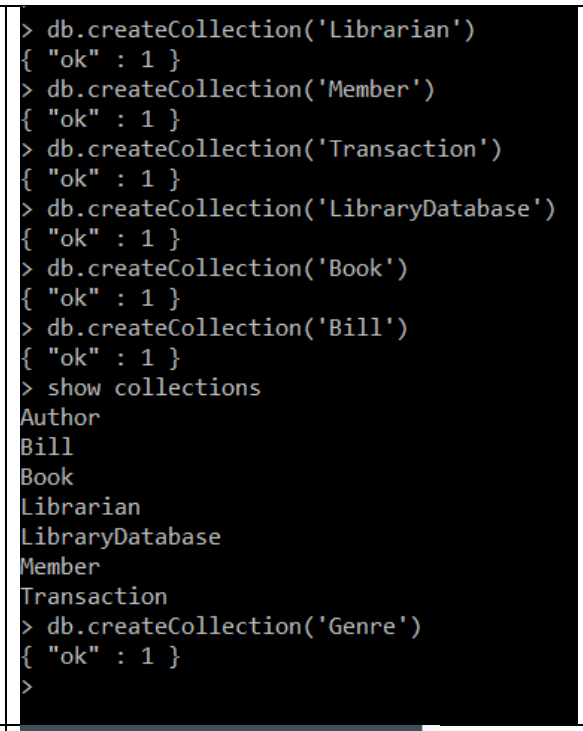
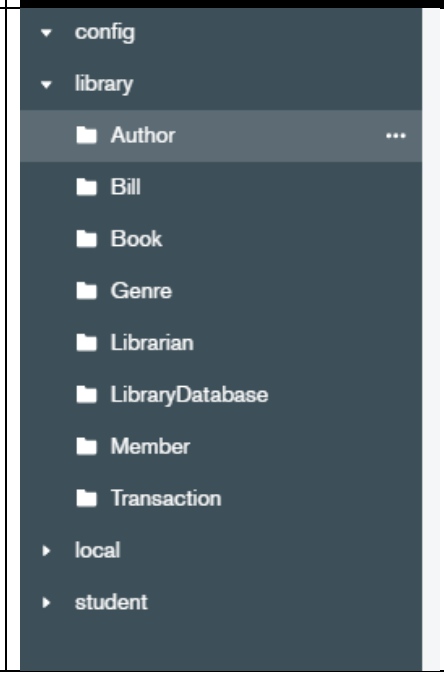
Creating table named "Member"	
Inserting data into table "Member"	
Creating table named "Bill"	

Inserting data into table “Bill”	`INSERT INTO Bill (billNo,createDate,memberId,amount) VALUES (1,CURRENT_TIMESTAMP, 5 ,3000);`
	`INSERT INTO Bill (billNo,createDate,memberId,amount) VALUES (2,CURRENT_TIMESTAMP, 6 ,3500);`
	`INSERT INTO Bill (billNo,createDate,memberId,amount) VALUES (3,CURRENT_TIMESTAMP, 3,4000);`
	`INSERT INTO Bill (billNo,createDate,memberId,amount) VALUES (4,CURRENT_TIMESTAMP, 2,1000);`
	`INSERT INTO Bill (billNo,createDate,memberId,amount) VALUES (5,CURRENT_TIMESTAMP, 7,500);`
	`INSERT INTO Bill (billNo,createDate,memberId,amount) VALUES (6,CURRENT_TIMESTAMP, 9,9000);`
	`INSERT INTO Bill (billNo,createDate,memberId,amount) VALUES (7,CURRENT_TIMESTAMP, 8,100);`
	Script Output
	Task completed in 0.505 seconds
	`1 row inserted.`

Creating table named “Transaction”	`CREATE TABLE Transaction (`
	`id NUMBER(10) NOT NULL,`
	`memberId NUMBER(10) NOT NULL,`
	`bookId NUMBER(10) NOT NULL,`
	`dateOfIssue TIMESTAMP NOT NULL,`
	`dueDate TIMESTAMP NULL,`
	`CONSTRAINT transcation_pk PRIMARY KEY (id),`
	`CONSTRAINT fkey_memberID FOREIGN KEY (memberId) REFERENCES Member(id),`
	`CONSTRAINT forkey_memberID FOREIGN KEY (bookId) REFERENCES Book(id)`
	`);`
	Script Output
	Task completed in 0.162 seconds
	`Table TRANSACTION created.`

Inserting data into table “Transaction”	`INSERT INTO Transaction (id,memberId,bookId,dateOfIssue,dueDate) VALUES (1,1,4,CURRENT_TIMESTAMP,'')`
	`INSERT INTO Transaction (id,memberId,bookId,dateOfIssue,dueDate) VALUES (2,2,7,CURRENT_TIMESTAMP,'')`
	`INSERT INTO Transaction (id,memberId,bookId,dateOfIssue,dueDate) VALUES (3,3,3,CURRENT_TIMESTAMP,'')`
	`INSERT INTO Transaction (id,memberId,bookId,dateOfIssue,dueDate) VALUES (4,4,1,CURRENT_TIMESTAMP,'')`
	`INSERT INTO Transaction (id,memberId,bookId,dateOfIssue,dueDate) VALUES (5,5,2,CURRENT_TIMESTAMP,'')`
	`INSERT INTO Transaction (id,memberId,bookId,dateOfIssue,dueDate) VALUES (6,6,5,CURRENT_TIMESTAMP,'')`
	`INSERT INTO Transaction (id,memberId,bookId,dateOfIssue,dueDate) VALUES (7,7,6,CURRENT_TIMESTAMP,'')`
	Script Output
	Task completed in 0.372 seconds
	`1 row inserted.`
	`1 row inserted.`

Task Three

Creating Collections in database name library.	
Displaying the created Collections in MongoDB.	

Inserting into Collection "Bill"	

```
> db.Bill.insert({'billNo': 1,'date':new ISODate("2020-05-18T14:10:30Z") , 'memberId': 5, 'amount': 3000
WriteResult({ "nInserted" : 1 })
> db.Bill.insert({'billNo': 2,'date':new ISODate("2020-06-20T20:30:21Z") , 'memberId': 6, 'amount': 3500
WriteResult({ "nInserted" : 1 })
> db.Bill.insert({'billNo': 3,'date': new ISODate("2020-06-18T20:15:36Z"), 'memberId': 3, 'amount': 4000
WriteResult({ "nInserted" : 1 })
> db.Bill.insert({'billNo': 4,'date':new ISODate("2020-07-13T20:30:22Z"), 'memberId': 2, 'amount': 1000
WriteResult({ "nInserted" : 1 })
>
> db.Bill.insert({'billNo': 5,'date': new ISODate("2020-08-18T20:12:32Z"), 'memberId': 7, 'amount': 500
...
...
> db.Bill.insert({'billNo': 5,'date': new ISODate("2020-08-18T20:12:32Z"), 'memberId': 7, 'amount': 500
...
...
> db.Bill.insert({'billNo': 5,'date': new ISODate("2020-08-18T20:12:32Z"), 'memberId': 7, 'amount': 500
WriteResult({ "nInserted" : 1 })
> db.Bill.insert({'billNo': 6,'date': new ISODate("2021-02-10T20:24:30Z"), 'memberId': 9, 'amount': 9000
WriteResult({ "nInserted" : 1 })
> db.Bill.insert({'billNo': 7,'date': new ISODate("2021-1-05T20:10:30Z"), 'memberId': 8, 'amount': 100}
uncaught exception: Error: invalid ISO date: 2021-1-05T20:10:30Z :
ISODate@src/mongo/shell/types.js:85:15
@(shell):1:37
> db.Bill.insert({'billNo': 7,'date': new ISODate("2021-01-05T20:10:30Z"), 'memberId': 8, 'amount': 100
WriteResult({ "nInserted" : 1 })
>
```

Displaying the inserted data in MongoDB.	`_id: ObjectId('62c7f4e32225846184c7fb67')`
	`billNo: 1`
	`date: 2020-05-18T14:10:30.000+00:00`
	`memberId: 5`
	`amount: 3000`
	
	`_id: ObjectId('62c7f6542225846184c7fb68')`
	`billNo: 2`
	`date: 2020-06-20T20:30:21.000+00:00`
	`memberId: 6`
	`amount: 3500`
	
	`_id: ObjectId('62c7f65d2225846184c7fb69')`
	`billNo: 3`
	`date: 2020-06-18T20:15:36.000+00:00`
	`memberId: 3`
	`amount: 4000`
	
	`_id: ObjectId('62c7f6622225846184c7fb6a')`
	`billNo: 4`
	`date: 2020-07-13T20:30:22.000+00:00`
	`memberId: 2`
	`amount: 1000`
	
	`_id: ObjectId('62c7f68c2225846184c7fb6b')`
	`billNo: 5`
	`date: 2020-08-18T20:12:32.000+00:00`
	`memberId: 7`
	`amount: 500`

Inserting into Collection "Book"	`> db.Book.insert({'id': 1,'title': 'Head First Java', 'publisher': 'OReilly Media Inc'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Book.insert({'id': 2,'title': 'The Catcher in the Rye', 'publisher': 'Little Brown'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Book.insert({'id': 3,'title': 'Nine Stories', 'publisher': 'Little Brown'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Book.insert({'id': 4,'title': 'Franny and Zooey', 'publisher': 'Gaurav'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Book.insert({'id': 5,'title': 'The Great Gatsby', 'publisher': 'Dark World'})`
	`WriteResult({ "nInserted" : 1 })`
	`>`
	`> db.Book.insert({'id': 6,'title': 'The Pragmatic Programmer', 'publisher': 'The Pragmatic Bookshelf'}`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Book.insert({'id': 6,'title': 'The Pragmatic Programmer', 'publisher': 'The Pragmatic Bookshelf'}`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Book.insert({'id': 7,'title': 'Clean Code', 'publisher': 'Pearson Education'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Book.insert({'id': 8,'title': 'Code Complete', 'publisher': 'Microsoft Press'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Book.insert({'id': 9,'title': 'Refactoring', 'publisher': 'Martin Fowler'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Book.insert({'id': 10,'title': 'Introduction to Algorithms', 'publisher': 'MIT Press'})`
	`WriteResult({ "nInserted" : 1 })`

Displaying the inserted data in MongoDB.	`_id: ObjectId('62c7ee942225846184c7fb5c')`
	`id: 1`
	`title: "Head First Java"`
	`publisher: "OReilly Media Inc"`
	
	`_id: ObjectId('62c7ee9f2225846184c7fb5d')`
	`id: 2`
	`title: "The Catcher in the Rye"`
	`publisher: "Little Brown"`
	
	`_id: ObjectId('62c7eeb32225846184c7fb5e')`
	`id: 3`
	`title: "Nine Stories"`
	`publisher: "Little Brown"`
	
	`_id: ObjectId('62c7eec02225846184c7fb5f')`
	`id: 4`
	`title: "Franny and Zooey"`
	`publisher: "Gaurav"`
	
	`_id: ObjectId('62c7eec42225846184c7fb60')`
	`id: 5`
	`title: "The Great Gatsby"`
	`publisher: "Dark World"`

Inserting into Collection "Librarian"	`> db.Librarian.insert({'id': 1,'librarian_name': 'Anjana Moktan'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Librarian.insert({'id': 2,'librarian_name': 'Saraswati Gauchan'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Librarian.insert({'id': 3,'librarian_name': 'Sudha Shakya'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Librarian.insert({'id': 4,'librarian_name': 'ABC'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Librarian.insert({'id': 5,'librarian_name': 'XYZ'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Librarian.insert({'id': 6,'librarian_name': 'Gaurav Shakya'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Librarian.insert({'id': 7,'librarian_name': 'Utsav Shakya'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Librarian.insert({'id': 8,'librarian_name': 'Rikit Shakya'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Librarian.insert({'id': 9,'librarian_name': 'Susan Shakya'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Librarian.insert({'id': 10,'librarian_name': 'Rina Moktan'})`
	`WriteResult({ "nInserted" : 1 })`

Displaying the inserted data in MongoDB.	`_id: ObjectId('62c7f93a2225846184c7fb6e')`
	`id: 1`
	`librarian_name: "Anjana Moktan"`
	
	`_id: ObjectId('62c7f9412225846184c7fb6f')`
	`id: 2`
	`librarian_name: "Saraswati Gauchan"`
	
	`_id: ObjectId('62c7f9442225846184c7fb70')`
	`id: 3`
	`librarian_name: "Sudha Shakya"`
	
	`_id: ObjectId('62c7f9482225846184c7fb71')`
	`id: 4`
	`librarian_name: "ABC"`
	
	`_id: ObjectId('62c7f94c2225846184c7fb72')`
	`id: 5`
	`librarian_name: "XYZ"`

Inserting into Collection "Member"	`> db.Member.insert({'id':1 ,'dateOfMembership':new ISODate("2019-01-08T12:00:00Z") , 'maxBookLimit': 5, 'name': 'Gaurav' ,'address':'Patan' })`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Member.insert({'id':2 ,'dateOfMembership':new ISODate("2019-01-09T13:00:00Z") , 'maxBookLimit': 5, 'name':'Saurav' ,'address':'USA' })`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Member.insert({'id':3 ,'dateOfMembership':new ISODate("2019-02-18T20:00:00Z") , 'maxBookLimit': 5, 'name': 'Sumit','address':'Boudha' })`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Member.insert({'id':4 ,'dateOfMembership':new ISODate("2019-02-20T10:00:00Z") , 'maxBookLimit': 5, 'name':'Kareena' ,'address': 'Mangalbaza`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Member.insert({'id':5 ,'dateOfMembership':new ISODate("2019-02-26T20:00:00Z") , 'maxBookLimit': 5, 'name':'Kusum','address':'Tripureshor'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Member.insert({'id': 6,'dateOfMembership':new ISODate("2019-03-12T20:00:00Z") , 'maxBookLimit': 5, 'name': 'Ruby','address': 'Chabahil'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Member.insert({'id': 7,'dateOfMembership':new ISODate("2019-04-05T20:00:00Z") , 'maxBookLimit': 5, 'name':'Rossie' ,'address': 'Home'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Member.insert({'id': 8,'dateOfMembership':new ISODate("2019-07-12T15:00:00Z") , 'maxBookLimit': 5, 'name': 'Sudeshna','address': 'ACE'})`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Member.insert({'id':9 ,'dateOfMembership':new ISODate("2019-08-01T20:00:00Z") , 'maxBookLimit': 5, 'name':'Sudarshan' ,'address':'Patan' })`
	`WriteResult({ "nInserted" : 1 })`
	`> db.Member.insert({'id': 10,'dateOfMembership':new ISODate("2019-08-05T12:00:00Z") , 'maxBookLimit': 5, 'name': 'Sid','address':'Home' })`
	`WriteResult({ "nInserted" : 1 })`

Displaying the inserted data in MongoDB.	_id: ObjectId('62c7fc8c2225846184c7fb78') id: 1 dateOfMembership: 2019-01-08T12:00:00.000+00:00 maxBookLimit: 5 name: "Gaurav" address: "Patan" _id: ObjectId('62c7fc942225846184c7fb79') id: 2 dateOfMembership: 2019-01-09T13:00:00.000+00:00 maxBookLimit: 5 name: "Saurav" address: "USA" _id: ObjectId('62c7fc992225846184c7fb7a') id: 3 dateOfMembership: 2019-02-18T20:00:00.000+00:00 maxBookLimit: 5 name: "Sumit" address: "Boudha"
Inserting into Collection "Transaction"	> db.Transaction.insert({'id': 2,'memberId': 2,'bookId': 7,'dateOfIssue':new ISODate("2020-07-13T20:30:22Z") ,'dueDate': ''}) WriteResult({ "nInserted" : 1 }) > db.Transaction.insert({'id': 5,'memberId': 5,'bookId': 2,'dateOfIssue':new ISODate("2020-05-18T14:10:30Z") ,'dueDate':'' }) WriteResult({ "nInserted" : 1 }) > db.Transaction.insert({'id': 6,'memberId': 6,'bookId': 5,'dateOfIssue':new ISODate("2020-06-20T20:30:21Z") ,'dueDate': ''}) WriteResult({ "nInserted" : 1 }) > db.Transaction.insert({'id': 7,'memberId': 7,'bookId': 6,'dateOfIssue':new ISODate("2020-08-18T20:12:32Z") ,'dueDate':'' }) WriteResult({ "nInserted" : 1 }) > db.Transaction.insert({'id': 1,'memberId': 1,'bookId': 4,'dateOfIssue':new ISODate("2020-08-05T12:00:00Z") ,'dueDate':'' }) WriteResult({ "nInserted" : 1 }) > db.Transaction.insert({'id': 3,'memberId': 3,'bookId': 3,'dateOfIssue':new ISODate("2020-06-18T20:15:36Z") ,'dueDate': ''}) WriteResult({ "nInserted" : 1 })
Displaying the inserted data in MongoDB.	_id: ObjectId('62c8008b2225846184c7fb82') id: 2 memberId: 2 bookId: 7 dateOfIssue: 2020-07-13T20:30:22.000+00:00 dueDate: "" _id: ObjectId('62c8009d2225846184c7fb83') id: 5 memberId: 5 bookId: 2 dateOfIssue: 2020-05-18T14:10:30.000+00:00 dueDate: "" _id: ObjectId('62c800a32225846184c7fb84') id: 6 memberId: 6 bookId: 5 dateOfIssue: 2020-06-20T20:30:21.000+00:00 dueDate: ""

Task Three:

Query a: A join of three or more tables

SQL Code

```
SELECT Member.name, Book.title, Bill.amount,
Transaction.dateOfIssue from Member

left outer join Bill on Member.id = Bill.memberId
inner join Transaction on Member.id = Transaction.bookId
join Book on Transaction.bookId = Book.id
order by Bill.amount asc
```

MongoDB code

```
> db.Member.aggregate([ {$lookup:{ from:"Bill", localField:"id",
foreignField:"memberId"  , as:"MemberBill" } }, {$lookup:{ from:"
Book", localField:"MemberBill.id", foreignField:"bookId", as:"Mem
berDetail"}}]).pretty()
```

Screenshots of output

NAME	TITLE	AM...	DATEOFISSUE
Rossie	Clean Code	500	08-JUL-22 04.17.21.14577..
Saurav	The Catcher i...	1000	08-JUL-22 04.17.21.25568..
Kusum	The Great Gatsby	3000	08-JUL-22 04.17.21.28663..
Ruby	The Pragmatic...	3500	08-JUL-22 04.17.21.32454..
Sumit	Nine Stories	4000	08-JUL-22 04.17.21.18346..
Gaurav	Head First Java	(null)	08-JUL-22 04.17.21.22149..
Kareena	Franny and Zooey	(null)	08-JUL-22 04.17.21.07423..


```
"_id" : ObjectId("62c7fc942225846184c7fb79"),
"id" : 2,
"dateOfMembership" : ISODate("2019-01-09T13:00:00Z"),
"maxBookLimit" : 5,
"name" : "Saurav",
"address" : "USA",
"MemberBill" : [
        {
                "_id" : ObjectId("62c7f6622225846184c7fb6a"),
                "billNo" : 4,
                "date" : ISODate("2020-07-13T20:30:22Z"),
                "memberId" : 2,
                "amount" : 1000
        }
],
"MemberDetail" : [
        {
                "_id" : ObjectId("62c7ee942225846184c7fb5c"),
                "id" : 1,
                "title" : "Head First Java",
                "publisher" : "OReilly Media Inc"
        },
        {
                "_id" : ObjectId("62c7ee9f2225846184c7fb5d"),
                "id" : 2,
                "title" : "The Catcher in the Rye",
                "publisher" : "Little Brown"
        },
        {
                "_id" : ObjectId("62c7eeb32225846184c7fb5e"),
                "id" : 3,
                "title" : "Nine Stories",
                "publisher" : "Little Brown"
        },
        {
                "_id" : ObjectId("62c7eec02225846184c7fb5f"),
                "id" : 4,
                "title" : "Franny and Zooey",
                "publisher" : "Gaurav"
        },
        {
```


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

```oracle
CREATE TYPE nested_table IS TABLE OF VARCHAR2(2);
/
CREATE TABLE Library(
    libraryId NUMBER(20),
    genre nested_table
)
NESTED TABLE genre STORE AS nested_tab;
/
select *from Library;
insert into Library values (1,nested_table());

insert into table(select l.genre from Library l
where l.libraryId =  1)
values('Action');

insert into table(select l.genre from Library l
where l.libraryId =  1)
values('Adventure');

insert into table(select l.genre from Library l
where l.libraryId =  1)
values('Horro');
```

MongoDB code

```mongodb
db.Library.insert({libraryId:
1,Genre:["Action","Adventure","Horror"]})
```

Screenshots of output

```
 LIBRARYID
----------
GENRE
--------------------------------------------
         1
NESTED_TABLE('Action', 'Adventure', 'Horro')
```

```
  _id: ObjectId('62c8647a2225846184c7fb8a')
  libraryId: 1
  Genre: Array
    0: "Action"
    1: "Adventure"
    2: "Horror"
```

Discussion:

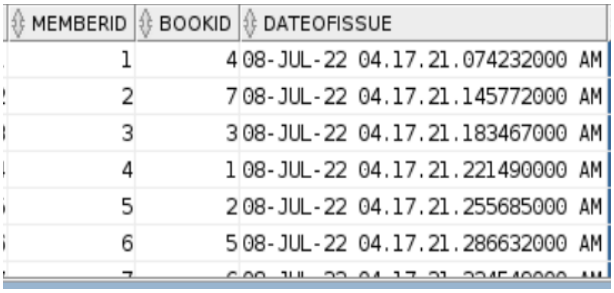
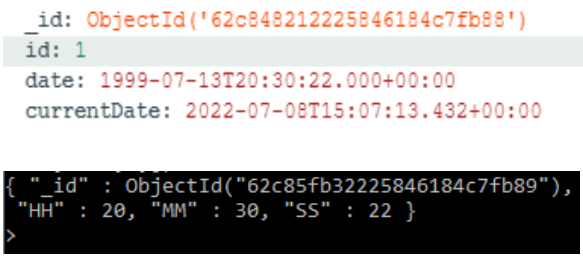
Oracle's nested functionality was implemented, but an exception occurred when entering data

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	
	

MEMBERID	BOOKID	DATEOFISSUE
1	4	08-JUL-22 04.17.21.074232000 AM
2	7	08-JUL-22 04.17.21.145772000 AM
3	3	08-JUL-22 04.17.21.183467000 AM
4	1	08-JUL-22 04.17.21.221490000 AM
5	2	08-JUL-22 04.17.21.255685000 AM
6	5	08-JUL-22 04.17.21.286632000 AM

```
_id: ObjectId('62c848212225846184c7fb88')
id: 1
date: 1999-07-13T20:30:22.000+00:00
currentDate: 2022-07-08T15:07:13.432+00:00
```

```
{ "_id" : ObjectId("62c85fb32225846184c7fb89"),
 "HH" : 20, "MM" : 30, "SS" : 22 }
>
```

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**
`SELECT Gender, SUM(amount) AS Total_Amount FROM Bill GROUP BY ROLLUP (gender)`	`> db.Bill.aggregate([ { $group: { _id: "$gender", TotalAmount: {$sum: "$amount"} }} ])`
Screenshots of output	
Script Output Task cor ---------- ---------- female 13500 male 7600 21100	{ "_id" : "female", "TotalAmount" : 13500 } { "_id" : "male", "TotalAmount" : 7600 }

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)**