MADHA ENGINEERING COLLEGE

(Affiliated to Anna University and Approved by AICTE, New Delhi) Madha Nagar, Kundrathur, Chennai-600069

DEPARTMENT OF Master of Computer Application

MC4211

Advanced Database Technology

Laboratory

R-2021

LAB MANUAL

4	2	1	2	2	2	2
5	2	1	2	2	2	2
Avg	2	1	2	2	2	2

MC4211 ADVANCED DATABASE TECHNOLOGY LABORATORY L T P C
0 0 4 2

COURSE OBJECTIVES:

- To understand the process of distributing tables across multiple systems
- To understand the process of storing, retrieving spatial and temporal data
- To understand the process of storing, retrieving objects in a database
- To understand the process of storing and retrieving data from a XML Database
- To use the open source database for building a mobile application

LIST OF EXPERIMENTS:

1. NOSQL Exercises
   a. MongoDB – CRUD operations, Indexing, Sharding
   b. Cassandra: Table Operations, CRUD Operations, CQL Types
   c. HIVE: Data types, Database Operations, Partitioning – HiveQL
   d. OrientDB Graph database – OrientDB Features
2. MySQL Database Creation, Table Creation, Query
3. MySQL Replication – Distributed Databases
4. Spatial data storage and retrieval in MySQL
5. Temporal data storage and retrieval in MySQL
6. Object storage and retrieval in MySQL
7. XML Databases , XML table creation, XQuery FLWOR expression
8. Mobile Database Query Processing using open source DB (MongoDB/MySQL etc)

TOTAL: 60 PERIODS

SOFTWARE REQUIREMENTS

1. Java / Python / R / Scala
2. Oracle, MySQL, MongoDB, Casandra, Hive

COURSE OUTCOMES:

On completion of the course, the student will be able to:

CO1: Design and implement advanced databases.
CO2: Use big data frameworks and tools.
CO3: Formulate complex queries using SQL.
CO4: Create an XML document and perform Xquery.
CO5: Query processing in Mobile databases using open source tools.

CO-PO Mapping

CO	**POs**					
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**
1	2	1	2	2	2	2

EX No: Date :	NOSQL EXERCISES MongoDB – CRUD Operations, Indexing, Sharding, Deployment

AIM:
To execute the queries to perform CRUD operations, Indexing, Sharding, Deployment in MongoDB.

PROCEDURE:

Step 1: Start the mongo deamon and run it in behind
Step 2: Start the mongo client
Step 3: Create the database
Step 4: Perform basic queries to perform CRUD (Create, Read, Update, Delete) operations.
Step 5: Perform basic queries for Indexing, Sharding and Deployment.

QUERIES:

//to start mongo deamon
C:/> mongod

//to start mongo client
C:/> mongo

//to list out database names
> show dbs

CRUD Queries:

//to create database
> use db1

//to check in which database I am working
> db

//to drop database in which I am working
> db.dropDatabase()

//To create collection
> db.createCollection('stud')

//to list out collection names
> show collections

//create collection by inserting document
> db.emp.insert({rno:1,name:'Bhavana'})

//Every row/document can be different than other

```
> db.emp.insert({name:'Amit',rno:2})
> db.emp.insert({rno:3, email_id:'a@gmail.com'})
```

// To display data from collection

```
> db.emp.find()
{ "_id" : ObjectId("5d7d3daf315728b4998f522e"), "rno" : 1, "name" : "Bhavana" }
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }
{ "_id" : ObjectId("5d7d3f56315728b4998f5230"), "rno" : 3, "email_id" : "a@gmail.com" }
```

//insert data by providing _id value

```
> db.emp.insert({_id:1,rno:4,name:"Akash"})

> db.emp.find()
{ "_id" : ObjectId("5d7d3daf315728b4998f522e"), "rno" : 1, "name" : "Bhavana" }
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }
{ "_id" : ObjectId("5d7d3f56315728b4998f5230"), "rno" : 3, "email_id" : "a@gmail.com" }
{ "_id" : 1, "rno" : 4, "name" : "Akash" }
```

// trying to insert data with duplicate _id, it will not accept as _id is primary key field

```
> db.emp.insert({_id:1,rno:5,name:"Reena"})
E11000 duplicate key error index: db1.emp.$_id_  dup key: { : 1.0 }
```

//Insert multiple documents at once

```
> db.emp.insert([{rno:7,name:'a'},{rno:8,name:'b'},{rno:8,name:'c'}])

> db.emp.find()
{ "_id" : ObjectId("5d7d3daf315728b4998f522e"), "rno" : 1, "name" : "Bhavana" }
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }
{ "_id" : ObjectId("5d7d3f56315728b4998f5230"), "rno" : 3, "email_id" : "a@gmail.com" }
{ "_id" : 1, "rno" : 4, "name" : "Akash" }
{ "_id" : 2, "rno" : 5, "name" : "Reena" }
{ "_id" : ObjectId("5d7d4244315728b4998f5231"), "rno" : 7, "name" : "a" }
{ "_id" : ObjectId("5d7d4244315728b4998f5232"), "rno" : 8, "name" : "b" }
{ "_id" : ObjectId("5d7d4244315728b4998f5233"), "rno" : 8, "name" : "c" }
```

// to insert multiple values for one key using []

```
> db.emp.insert({rno:10,name:'Ankit',hobbies:['singing','cricket','swimming'],age:21})

> db.emp.find()
{ "_id" : ObjectId("5d7d3daf315728b4998f522e"), "rno" : 1, "name" : "Bhavana" }
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }
{ "_id" : ObjectId("5d7d3f56315728b4998f5230"), "rno" : 3, "email_id" : "a@gmail.com" }
{ "_id" : 1, "rno" : 4, "name" : "Akash" }
{ "_id" : 2, "rno" : 5, "name" : "Reena" }
{ "_id" : ObjectId("5d7d4244315728b4998f5231"), "rno" : 7, "name" : "a" }
```

{ "_id" : ObjectId("5d7d4244315728b4998f5232"), "rno" : 8, "name" : "b" }
{ "_id" : ObjectId("5d7d4244315728b4998f5233"), "rno" : 8, "name" : "c" }
{ "_id" : ObjectId("5d7d433a315728b4998f5234"), "rno" : 10, "name" : "Ankit", "hobbies" : ["singing", "cricket", "swimming"], "age" : 21 }

// Embedded document example

> db.emp.insert({rno:11, Name: {Fname:"Bhavana", Mname:"Amit", Lname:"Khivsara"}})
> db.emp.insert({rno:12, Name: "Janvi", Address:{Flat:501, Building:"Sai Appart", area:"Tidke colony", city: "Nashik", state:"MH", pin:423101}, age:22})

// To insert date use ISODate function

> db.emp.insert({rno:15, name:'Ravina', dob: ISODate("2019-09-14")})

> db.emp.find()
{ "_id" : ObjectId("5d7d3daf315728b4998f522e"), "rno" : 1, "name" : "Bhavana" }
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }
{ "_id" : ObjectId("5d7d3f56315728b4998f5230"), "rno" : 3, "email_id" : "a@gmail.com" }
{ "_id" : 1, "rno" : 4, "name" : "Akash" }
{ "_id" : 2, "rno" : 5, "name" : "Reena" }
{ "_id" : ObjectId("5d7d4244315728b4998f5231"), "rno" : 7, "name" : "a" }
{ "_id" : ObjectId("5d7d4244315728b4998f5232"), "rno" : 8, "name" : "b" }
{ "_id" : ObjectId("5d7d4244315728b4998f5233"), "rno" : 8, "name" : "c" }
{ "_id" : ObjectId("5d7d433a315728b4998f5234"), "rno" : 10, "name" : "Ankit", "hobbies" : ["singing", "cricket", "swimming"], "age" : 21 }
{ "_id" : ObjectId("5d7d4462315728b4998f5235"), "rno" : 11, "Name" : { "Fname" : "Bhavana", "Mname" : "Amit", "Lname" : "Khivsara" } }
{ "_id" : ObjectId("5d7d4574315728b4998f5236"), "rno" : 12, "Name" : "Janvi", "Address" : { "Flat" : 501, "Building" : "Sai Appart", "area" : "Tidke colony", "city" : "Nashik", "state" : "MH", "pin" : 423101 }, "age" : 22 }
{ "_id" : ObjectId("5d7d465d315728b4998f5237"), "rno" : 15, "name" : "Ravina", "dob" : ISODate("2019-09-14T00:00:00Z") }
>

// Multi embedded document with data function

> db.emp.insert({rno:17, name:"Ashika",date:Date(), awards:[{name:"Best c -Designer", year:2010, prize:"winner"},{name:"Wen site competition",year:2012,prize:"Runner-up"},{name:"Fashion show", year:2015,prize:"winner"}], city:"Nashik"})

// ouput using pretty command

> db.emp.find().pretty()
{
	"_id" : ObjectId("5d7d3daf315728b4998f522e"),
	"rno" : 1,
	"name" : "Bhavana"
}
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }

```
{
        "_id" : ObjectId("5d7d3f56315728b4998f5230"),
        "rno" : 3,
        "email_id" : "a@gmail.com"
}
{ "_id" : 1, "rno" : 4, "name" : "Akash" }
{ "_id" : 2, "rno" : 5, "name" : "Reena" }
{ "_id" : ObjectId("5d7d4244315728b4998f5231"), "rno" : 7, "name" : "a" }
{ "_id" : ObjectId("5d7d4244315728b4998f5232"), "rno" : 8, "name" : "b" }
{ "_id" : ObjectId("5d7d4244315728b4998f5233"), "rno" : 8, "name" : "c" }
{
        "_id" : ObjectId("5d7d433a315728b4998f5234"),
        "rno" : 10,
        "name" : "Ankit",
        "hobbies" : [
                "singing",
                "cricket",
                "swimming"
        ],
        "age" : 21
}
{
        "_id" : ObjectId("5d7d4462315728b4998f5235"),
        "rno" : 11,
        "Name" : {
                "Fname" : "Bhavana",
                "Mname" : "Amit",
                "Lname" : "Khivsara"
        }
}
{
        "_id" : ObjectId("5d7d4574315728b4998f5236"),
        "rno" : 12,
        "Name" : "Janvi",
        "Address" : {
                "Flat" : 501,
                "Building" : "Sai Appart",
                "area" : "Tidke colony",
                "city" : "Nashik",
                "state" : "MH",
                "pin" : 423101
        },
        "age" : 22
}
{
        "_id" : ObjectId("5d7d465d315728b4998f5237"),
```

```
        "rno" : 15,
        "name" : "Ravina",
        "dob" : ISODate("2019-09-14T00:00:00Z")
}
{
        "_id" : ObjectId("5d7d4aa7315728b4998f5238"),
        "rno" : 17,
        "name" : "Ashika",
        "date" : "Sat Sep 14 2019 16:16:39 GMT-0400 (EDT)",
        "awards" : [
                {
                        "name" : "Best C-designer",
                        "year" : 2010,
                        "prize" : "winner"
                },
                {
                        "name" : "Wen site competition",
                        "year" : 2012,
                        "prize" : "Runner-up"
                },
                {
                        "name" : "Fashion show",
                        "year" : 2015,
                        "prize" : "winner"
                }
        ],
        "city" : "Nashik"
}
```

// New collection for Find operation

```
> db.stud.insert([{rno:1, name:'Ashiti'}, {rno:2,name:'Savita'}, {rno:3,name:'Sagar'},
{rno:4,name:'Reena'},{rno:5,name:'Jivan'}])
```

//Simple Find Command

```
> db.stud.find()
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
```

//Find command with Condition

```
> db.stud.find({rno:5})
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
```

//Find command with condition with giving name field only to show

```
> db.stud.find({rno:5},{name:1})
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "name" : "Jivan" }
```

//Find command with condition with giving name field only to show and _id to hide

```
> db.stud.find({rno:5},{name:1,_id:0})
{ "name" : "Jivan" }
```

// Find command to show only names without condition

```
> db.stud.find({},{name:1,_id:0})
{ "name" : "Ashiti" }
{ "name" : "Savita" }
{ "name" : "Sagar" }
{ "name" : "Reena" }
{ "name" : "Jivan" }
```

// To display data whose rno is greater than 2

```
> db.stud.find({rno:{$gt:2}})
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
```

// To display data whose rno is less than equal to 2

```
> db.stud.find({rno:{$lte:2}})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }
```

// To display data whose rno is less than 2

```
> db.stud.find({rno:{$lt:2}})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
```

// To display data whose rno is not equal to 2

```
> db.stud.find({rno:{$ne:2}})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
```

// To display data whose rno is either 1 or 3 or 5 using in operator

```
> db.stud.find({rno:{$in:[1,3,5]}})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
```

// To display data whose rno is either 1 or 3 or 5 or 7 or 9 using in operator

```
> db.stud.find({rno:{$in:[1,3,5,7,9]}})
```

{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }

//Sorting Command -1 is for Descending

> db.stud.find().sort({rno:-1})
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }

//Sorting Command 1 is for Ascending

> db.stud.find().sort({name:1})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }

//Dispay rno & name whose rno is greater than 2. Show output in decending order by rno

> db.stud.find({rno:{$gt:2}},{_id:0}).sort({rno:-1})
{ "rno" : 5, "name" : "Jivan" }
{ "rno" : 4, "name" : "Reena" }
{ "rno" : 3, "name" : "Sagar" }

//Collection with 3 and 5 rollno as duplicate values

> db.stud.find()
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
{ "_id" : ObjectId("5d83b8d9a44331f62bcd836e"), "rno" : 5, "name" : "Radhika" }
{ "_id" : ObjectId("5d83b8eba44331f62bcd836f"), "rno" : 3, "name" : "Manioj" }

//Distinct command to show only unique values for roll no

> db.stud.distinct("rno")
[1, 2, 3, 4, 5]

// Limit use to show only some records from starting- following command shows only first 2 records from collection

> db.stud.find().limit(2)
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }

{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }

// Skip use to show all records after skipping some records- following command shows all records after first 2 records from collection

```
> db.stud.find().skip(2)
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
{ "_id" : ObjectId("5d83b8d9a44331f62bcd836e"), "rno" : 5, "name" : "Radhika" }
{ "_id" : ObjectId("5d83b8eba44331f62bcd836f"), "rno" : 3, "name" : "Manioj" }
```

// Shows documents where name starting with A

```
> db.stud.find({name:/^A/})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
```

// Shows documents where name ending with i

```
> db.stud.find({name:/i$/})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
```

// Shows documents where name having letter a anywhere

```
> db.stud.find({name:/a/})
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
{ "_id" : ObjectId("5d83b8d9a44331f62bcd836e"), "rno" : 5, "name" : "Radhika" }
{ "_id" : ObjectId("5d83b8eba44331f62bcd836f"), "rno" : 3, "name" : "Manioj" }
>
```

//findOne to show only first record

```
> db.stud.findOne()
{
        "_id" : ObjectId("5d83af5aa44331f62bcd8369"),
        "rno" : 1,
        "name" : "Ashiti"
}
```

// count to show number of documents in collection

```
> db.stud.find().count()
7

> db.stud.find({rno:{$gt:2}}).count()
5
```

//Insert one embedded document(for address)

```
> db.stud.insert({rno:8,address:{area:"College Road",city:"Nashik",state:"MH"},name:"Arya"})
```

//To find document having city as Nashik(as city is key of address key specify "address.city"

```
> db.stud.find({"address.city":"Nashik"})
{ "_id" : ObjectId("5d83c04aa44331f62bcd8370"), "rno" : 8, "address" : { "area" : "College Road", "city" : "Nashik", "state" : "MH" }, "name" : "Arya" }
```

//Insert one document with multiple values(eg hobbies)

```
> db.stud.insert({rno:9,hobbies:['singing','dancing','cricket']})
```

//To use find command on multi values attribute(eg hobbies)

```
> db.stud.find({hobbies:'dancing'})
{ "_id" : ObjectId("5d83c165a44331f62bcd8371"), "rno" : 9, "hobbies" : [  "singing",  "dancing",  "cricket" ] }
```

//$unset will remove the column rno from document matching the given condition

```
> db.stud.update({rno:1},{$unset:{rno:1}})
```

//$set to update the value of rno

```
>db.stud.update({rno:2},{$set:{rno:22}})
```

//upsert use to update document if condition found otherwise insert document with updates values.

```
> db.stud.update({rno:50},{$set:{rno:55}},{upsert:true})
```

//multi:true used to update in multiple documents

```
> db.stud.update({rno:5},{$set:{rno:15}},{multiple:true})
```

//It will remove record having rno as 4

```
> db.stud.remove({rno:4})
```

//It will remove only one record having rno as 4

```
> db.stud.remove({rno:4},1)
```

//It will remove all records

```
> db.stud.remove({})
```

Indexing:

//To create index on rno in ascending order(1)- //Single field Index example

```
>db.stud.createIndex({rno:1})
```

//To show the list of Index , v is version, key is on which field you created index
//ns-name space(database name.collection name), name- Name of index given by mongodb

```
>db.stud.getIndexes()
[
        {
```

```
                "v" : 1,
                "key" : {
                        "_id" : 1
                },
                "ns" : "db1.stud",
                "name" : "_id_"
        },
        {
                "v" : 1,
                "key" : {
                        "rno" : 1
                },
                "ns" : "db1.stud",
                "name" : "rno_1"
        }
]
```

//Compound Index Example (-1 is descending & 1 is ascending)

```
>db.stud.createIndex({rno:-1,name:1})

>db.stud.getIndexes()
[
        {
                "v" : 1,
                "key" : {
                        "_id" : 1
                },
                "ns" : "db1.stud",
                "name" : "_id_"
        },
        {
                "v" : 1,
                "key" : {
                        "rno" : 1
                },
                "ns" : "db1.stud",
                "name" : "rno_1"
        },
        {
                "v" : 1,
                "key" : {
                        "rno" : -1,
                        "name" : 1
                },
                "ns" : "db1.stud",
                "name" : "rno_-1_name_1"
        }
]
```

// To drop single index

```
>db.stud.dropIndex({rno:1})
{ "nIndexesWas" : 3, "ok" : 1 }
```

// To drop all indexes at a time

```
>db.stud.dropIndexes()
{
        "nIndexesWas" : 2,
        "msg" : "non-_id indexes dropped for collection",
        "ok" : 1
}
```

RESULT:

The queries to perform CRUD, Indexing, Sharding and Deployment was executed successfully.

EX No:	NOSQL EXERCISES Cassandra: Table Operations, CRUD Operations, CQL Types.
Date :	

AIM:
To execute the queries to perform Table Operations, CRUD operations, CQL Types in Cassandra.

PROCEDURE:

Step 1: Start the Cassandra server and run it in behind.
Step 2: Start the CQL client shell.
Step 3: Create the database
Step 4: Perform basic queries to perform Table operations.
Step 5: Perform basic queries to perform CRUD (Create, Read, Update, Delete) operations.
Step 5: Perform basic queries for CQL Types.

QUERIES:

1. **Create Keyspace:**

```
cqlsh.> CREATE KEYSPACE stud WITH replication = {'class':'SimpleStrategy', 'replication_factor'
: 3};

cqlsh> CREATE KEYSPACE test WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy',
'datacenter1' : 3 } AND DURABLE_WRITES = false;

cqlsh> USE test;
cqlsh:test>
```

2. **Table Operations**

```
cqlsh:test> CREATE TABLE emp( emp_id int PRIMARY KEY, emp_name text, emp_city text,
  emp_sal varint, emp_phone varint);

cqlsh:test> ALTER TABLE emp ADD emp_email text;

cqlsh:test> ALTER TABLE emp DROP emp_email;

cqlsh:test> DROP TABLE emp;

cqlsh:test> TRUNCATE student;
```

3. **CRUD Operations**
   1. **Create data**

```
cqlsh:test> INSERT INTO emp (emp_id, emp_name, emp_city,
  emp_phone, emp_sal) VALUES (1,'ram', 'Hyderabad', 9848022338, 50000);

cqlsh:test> INSERT INTO emp (emp_id, emp_name, emp_city,
  emp_phone, emp_sal) VALUES(2,'robin', 'Hyderabad', 9848022339, 40000);

cqlsh:test> INSERT INTO emp (emp_id, emp_name, emp_city,
  emp_phone, emp_sal) VALUES(3,'rahman', 'Chennai', 9848022330, 45000);
```

2. To read all data from table

```
cqlsh:test> SELECT * FROM emp;

 emp_id |  emp_city | emp_name |  emp_phone | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 | 50000
      2 | Hyderabad |    robin | 9848022339 | 40000
      3 |   Chennai |   rahman | 9848022330 | 45000
 (3   rows)
```

3. To update data in a table

```
cqlsh:test> UPDATE emp SET emp_city='Delhi',emp_sal=50000 WHERE emp_id=2;
cqlsh:test> select * from emp;

 emp_id |  emp_city | emp_name |  emp_phone | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 | 50000
      2 |     Delhi |    robin | 9848022339 | 50000
      3 |   Chennai |   rahman | 9848022330 | 45000
 (3 rows)

cqlsh:test> SELECT emp_name, emp_sal from emp;

 emp_name | emp_sal
----------+---------
      ram | 50000
    robin | 50000
   rajeev | 30000
   rahman | 50000
(4 rows)
```

4. To create index

```
cqlsh:test> CREATE INDEX sal ON emp(emp_sal);
cqlsh:test> SELECT * FROM emp WHERE emp_sal=50000;

 emp_id |  emp_city | emp_name |  emp_phone | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 | 50000
      2 |      null |    robin | 9848022339 | 50000
      3 |   Chennai |   rahman | 9848022330 | 50000
```

5. To drop index

```
cqlsh:test> drop index sal;
```

6. To Deleting Data from a Table

```
cqlsh:tutorialspoint> DELETE emp_sal FROM emp WHERE emp_id=3;
cqlsh:test> select * from emp;

 emp_id |  emp_city | emp_name |  emp_phone | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 | 50000
      2 |     Delhi |    robin | 9848022339 | 50000
      3 |   Chennai |   rahman | 9848022330 | null
(3 rows)
```

To Delete a complete row in a column

```
cqlsh:test> DELETE FROM emp WHERE emp_id=3;
cqlsh:test> select * from emp;

 emp_id |  emp_city | emp_name |  emp_phone | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 | 50000
      2 |     Delhi |    robin | 9848022339 | 50000
(2 rows)
```

4. CQL Types

CQL provides a rich set of built-in data types, including collection types. Along with these data types, users can also create their own custom data types. The following table provides a list of built-in data types available in CQL.

Data Type	Constants	Description
Ascii	Strings	Represents ASCII character string
bigint	Bigint	Represents 64-bit signed long
blob	Blobs	Represents arbitrary bytes
Boolean	Booleans	Represents true or false
counter	Integers	Represents counter column
decimal	integers, floats	Represents variable-precision decimal
double	Integers	Represents 64-bit IEEE-754 floating point
float	integers, floats	Represents 32-bit IEEE-754 floating point
inet	Strings	Represents an IP address, IPv4 or IPv6
int	Integers	Represents 32-bit signed int
text	Strings	Represents UTF8 encoded string
timestamp	integers, strings	Represents a timestamp
timeuuid	Uuids	Represents type 1 UUID
uuid	uuids	Represents type 1 or type 4 UUID
varchar	Strings	Represents uTF8 encoded string
varint	Integers	Represents arbitrary-precision integer

Collection Types

Cassandra Query Language also provides a collection data types. The following table provides a list of Collections available in CQL.

Collection	Description
List	A list is a collection of one or more ordered elements.
Map	A map is a collection of key-value pairs.
Set	A set is a collection of one or more elements.

User-defined datatypes

Cqlsh provides users a facility of creating their own data types. Given below are the commands used while dealing with user defined data types.

- CREATE TYPE − Creates a user-defined data type.
- ALTER TYPE − Modifies a user-defined data type.
- DROP TYPE − Drops a user-defined data type.
- DESCRIBE TYPE − Describes a user-defined data type.
- DESCRIBE TYPES − Describes user-defined data types.

1. **Create a user-defined type named address.**

```
CREATE TYPE mykeyspace.address (
  street text,
  city text,
  zip_code int,
  phones set<text> );
```

2. **Create a user-defined type for the name of a user.**

```
CREATE TYPE mykeyspace.fullname (
  firstname text,
  lastname text);
```

3. **Create a table for storing user data in columns of type fullname and address.**
   Use the frozen keyword in the definition of the user-defined type column.

```
CREATE TABLE mykeyspace.users (
  id uuid PRIMARY KEY,
  name frozen <fullname>,
  direct_reports set<frozen <fullname>>,     // a collection set
  addresses map<text, frozen <address>>      // a collection map);
```

4. **Insert a user's name into the fullname column.**

```
INSERT INTO mykeyspace.users (id, name) VALUES (62c36092-82a1-3a00-93d1-
   46196ee77204, {firstname: 'Marie-Claude', lastname: 'Josset'});
```

5. **Insert an address labeled home into the table.**

```
UPDATE mykeyspace.users SET addresses = addresses + {'home': { street: '191 Rue St.
Charles', city: 'Paris', zip_code: 75015, phones: {'33 6 78 90 12 34'}}} WHERE
id=62c36092-82a1-3a00-93d1-46196ee77204;
```

6. **Retrieve the full name of a user.**

```
SELECT name FROM mykeyspace.users WHERE id=62c36092-82a1-3a00-93d1-46196ee77204;
name
------------------------------------------------
 {firstname: 'Marie-Claude', lastname: 'Josset'}
```

7. **Using dot notation, you can retrieve a component of the user-defined type column**.

```
 SELECT name.lastname FROM mykeyspace.users WHERE id=62c36092-82a1-3a00-93d1-46196ee77204;
 name.lastname
---------------
   Josset
```

8. **To create index**

```
CREATE INDEX on mykeyspace.users (name);
SELECT id FROM mykeyspace.users WHERE name = {firstname: 'Marie-Claude', lastname: 'Josset'};

 id
--------------------------------------
 62c36092-82a1-3a00-93d1-46196ee77204
```

9. To update a complete UDT

```
UPDATE mykeyspace.users SET direct_reports = { ( 'Naoko', 'Murai'), ( 'Sompom', 'Peh') } WHERE id=62c36092-82a1-3a00-93d1-46196ee77204;

INSERT INTO mykeyspace.users (id, direct_reports) VALUES ( 7db1a490-5878-11e2-bcfd-0800200c9a66, { ('Jeiranan', 'Thongnopneua') } );

SELECT direct_reports FROM mykeyspace.users;

 direct_reports
-------------------------------------------------------------------------------
                        {{firstname: 'Jeiranan', lastname: 'Thongnopneua'}}
 {{firstname: 'Naoko', lastname: 'Murai'}, {firstname: 'Sompom', lastname: 'Peh'}}
```

RESULT:

The queries to create keyspace, create table and CQL types have been executed successfully.

EX No: Date :	NOSQL EXERCISES HIVE: Data types, Database Operations, Partitioning – HiveQL

AIM:
To create a database for executing database operations , partitioning and execute hive queries in HIVE.

PROCEDURE:
Step 1: Start the hadoop and run it in behind.
Step 2: Start the derby server and let it run in background.
Step 3: Start yarn/ MapReduce
Step 4: Start network server (use 0.0.0.0 as host address)
Step 5: Start hive. Create the database .Perform basic queries to perform database operations.
Step 5: Perform basic queries to perform partitioning. Perform basic queries for HiveQL .

THEORY and QUERY:

1. **Data types**

All the data types in Hive are classified into four types, given as follows:

- Column Types
- Literals
- Null Values
- Complex Types

Column Types
Column type are used as column data types of Hive. They are as follows:

Integral Types
Integer type data can be specified using integral data types, INT. When the data range exceeds the range of INT, you need to use BIGINT and if the data range is smaller than the INT, you use SMALLINT. TINYINT is smaller than SMALLINT.

The following table depicts various INT data types:

Type	Postfix	Example
TINYINT	Y	10Y
SMALLINT	S	10S
INT	-	10
BIGINT	L	10L

String Types
String type data types can be specified using single quotes (' ') or double quotes (" "). It contains two data types: VARCHAR and CHAR. Hive follows C-types escape characters.

The following table depicts various CHAR data types:

Data Type	Length
VARCHAR	1 to 65355
CHAR	255

Timestamp
It supports traditional UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format “YYYY-MM-DD HH:MM:SS.fffffffff” and format “yyyy-mm-dd hh:mm:ss.fffffffff”.

Dates
DATE values are described in year/month/day format in the form {{YYYY-MM-DD}}.

Decimals
The DECIMAL type in Hive is as same as Big Decimal format of Java. It is used for representing immutable arbitrary precision. The syntax and example is as follows:

DECIMAL (precision, scale)
Decimal (10,0)

Union Types
Union is a collection of heterogeneous data types. You can create an instance using create union. The syntax and example is as follows:

UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>
{0:1}
{1:2.0}
{2:["three","four"]}
{3:{"a":5,"b":"five"}}
{2:["six","seven"]}
{3:{"a":8,"b":"eight"}}
{0:9}
{1:10.0}

Literals
The following literals are used in Hive:

Floating Point Types
Floating point types are nothing but numbers with decimal points. Generally, this type of data is composed of DOUBLE data type.

Decimal Type
Decimal type data is nothing but floating point value with higher range than DOUBLE data type. The range of decimal type is approximately -10-308 to 10308.
Null Value
Missing values are represented by the special value NULL.

Complex Types
The Hive complex data types are as follows:

Arrays
Arrays in Hive are used the same way they are used in Java.
Syntax: ARRAY<data_type>

Maps
Maps in Hive are similar to Java Maps.
Syntax: MAP<primitive_type, data_type>

Structs
Structs in Hive is similar to using complex data with comment.
Syntax: STRUCT<col_name : data_type [COMMENT col_comment], ...>

2. Database Operations and partitions

To create Database:

```
hive> CREATE DATABASE financials;

hive> CREATE DATABASE IF NOT EXISTS financials;
```

To list the Database:

```
hive> SHOW DATABASES;
default
financials

hive> CREATE DATABASE human_resources;

hive> SHOW DATABASES LIKE 'h.*';
human_resources
```

To create database by specifying its location to store

```
hive> CREATE DATABASE financials
   > LOCATION '/my/preferred/directory';
```

To create database with a comment

```
hive> DESCRIBE DATABASE financials;
financials   Holds all financial tables
  hdfs://master-server/user/hive/warehouse/financials.db
```

To create database with additional properties

```
hive> CREATE DATABASE financials
   > WITH DBPROPERTIES ('creator' = 'Mark Moneybags', 'date' = '2021-01-02');
```

To describe the database design

```
hive> DESCRIBE DATABASE financials;
financials   hdfs://master-server/user/hive/warehouse/financials.db

hive> DESCRIBE DATABASE EXTENDED financials;
financials   hdfs://master-server/user/hive/warehouse/financials.db
 {date=2021-01-02, creator=Mark Moneybags);
```

To set a database

```
hive> USE financials;
```

To delete a database

```
hive> DROP DATABASE IF EXISTS financials;
```

To drop a table inside the database before deleting the database

```
hive> DROP DATABASE IF EXISTS financials CASCADE;
```

To alter the database properties

```
hive> ALTER DATABASE financials SET DBPROPERTIES ('edited-by' = 'Joe Dba');
```

To create table

```
CREATE TABLE IF NOT EXISTS mydb.employees (
  name         STRING COMMENT 'Employee name',
  salary       FLOAT  COMMENT 'Employee salary',
```

```
  subordinates ARRAY<STRING> COMMENT 'Names of subordinates',
  deductions   MAP<STRING, FLOAT>
          COMMENT 'Keys are deductions names, values are percentages',
  address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
          COMMENT 'Home address')
COMMENT 'Description of the table'
TBLPROPERTIES ('creator'='me', 'created_at'='2021-01-02 10:00:00', ...)
LOCATION '/user/hive/warehouse/mydb.db/employees';
```

To copy a schema

```
CREATE TABLE IF NOT EXISTS mydb.employees2
LIKE mydb.employees;
```

To list out the tables

```
hive> USE mydb;

hive> SHOW TABLES;
employees
table1
table2

hive> USE default;

hive> SHOW TABLES IN mydb;
employees
table1
table2
```

To describe the table schema

```
hive> DESCRIBE EXTENDED mydb.employees;
name    string  Employee name
salary  float   Employee salary
subordinates    array<string>   Names of subordinates
deductions      map<string,float> Keys are deductions names, values are percentages
address struct<street:string,city:string,state:string,zip:int>  Home address

Detailed Table Information Table(tableName:employees, dbName:mydb, owner:me,
...
location:hdfs://master-server/user/hive/warehouse/mydb.db/employees,
parameters:{creator=me, created_at='2021-01-02 10:00:00',
          last_modified_user=me, last_modified_time=1337544510,
          comment:Description of the table, ...}, ...)
```

To describe the schema of a particular column

```
hive> DESCRIBE mydb.employees.salary;
salary  float   Employee salary
```

To partition the data first by country and then by state:

```
CREATE TABLE employees (
  name         STRING,
  salary       FLOAT,
  subordinates ARRAY<STRING>,
  deductions   MAP<STRING, FLOAT>,
  address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
```

)
PARTITIONED BY (country STRING, state STRING);

To show the partitions
hive> SHOW PARTITIONS employees;
...
Country=CA/state=AB
country=CA/state=BC
...
country=US/state=AL
country=US/state=AK
...

To describe the partitioned extended table
hive> DESCRIBE EXTENDED employees;
name string,
salary float,
...
address struct<...>,
country string,
state string

Detailed Table Information...
partitionKeys:[FieldSchema(name:country, type:string, comment:null),
FieldSchema(name:state, type:string, comment:null)],
...

To delete a table
DROP TABLE IF EXISTS employees;

RESULT:
The database its operations , partitioning and hive queries have been executed successfully in HIVE.

EX No: Date :	NOSQL EXERCISES OrientDB Graph database – OrientDB Features

AIM:
To study about the OrientDB Graph database and its features.

THEORY:
Introduction
OrientDB is an Open Source NoSQL Database Management System, which contains the features of traditional DBMS along with the new features of both Document and Graph DBMS. It is written in Java and is amazingly fast. It can store 220,000 records per second on commodity hardware. OrientDB, is one of the best open-source, multi-model, next generation NoSQL product.

OrientDB is an Open Source NoSQL Database Management System. NoSQL Database provides a mechanism for storing and retrieving NO-relation or NON-relational data that refers to data other than tabular data such as document data or graph data. NoSQL databases are increasingly used in Big Data and real-time web applications. NoSQL systems are also sometimes called "Not Only SQL" to emphasize that they may support SQL-like query languages.

OrientDB also belongs to the NoSQL family. OrientDB is a second generation Distributed Graph Database with the flexibility of Documents in one product with an open source of Apache 2 license.

Features	**MongoDB**	**OrientDB**
Relationships	Uses the RDBMS JOINS to create relationship between entities. It has high runtime cost and does not scale when database scale increases.	Embeds and connects documents like relational database. It uses direct, super-fast links taken from graph database world.
Fetch Plan	Costly JOIN operations.	Easily returns complete graph with interconnected documents.
Transactions	Doesn't support ACID transactions, but it supports atomic operations.	Supports ACID transactions as well as atomic operations.
Query language	Has its own language based on JSON.	Query language is built on SQL.
Indexes	Uses the B-Tree algorithm for all indexes.	Supports three different indexing algorithms so that the user can achieve best performance.
Storage engine	Uses memory mapping technique.	Uses the storage engine name LOCAL and PLOCAL.

OrientDB is the first Multi-Model open source NoSQL DBMS that brings together the power of graphs and flexibility of documents into a scalable high-performance operational database.

The main feature of OrientDB is to support multi-model objects, i.e. it supports different models like Document, Graph, Key/Value and Real Object. It contains a separate API to support all these four models.

Document Model

The terminology Document model belongs to NoSQL database. It means the data is stored in the Documents and the group of Documents are called as Collection. Technically, document means a set of key/value pairs or also referred to as fields or properties.

OrientDB uses the concepts such as classes, clusters, and link for storing, grouping, and analyzing the documents.

The following table illustrates the comparison between relational model, document model, and OrientDB document model

Relational Model	**Document Model**	**OrientDB Document Model**
Table	Collection	Class or Cluster
Row	Document	Document
Column	Key/value pair	Document field
Relationship	Not available	Link

Graph Model

A graph data structure is a data model that can store data in the form of Vertices (Nodes) interconnected by Edges (Arcs). The idea of OrientDB graph database came from property graph. The vertex and edge are the main artifacts of the Graph model. They contain the properties, which can make these appear similar to documents.

The following table shows a comparison between graph model, relational data model, and OrientDB graph model.

Relational Model	**Graph Model**	**OrientDB Graph Model**
Table	Vertex and Edge Class	Class that extends "V" (for Vertex) and "E" (for Edges)
Row	Vertex	Vertex
Column	Vertex and Edge property	Vertex and Edge property
Relationship	Edge	Edge

The Key/Value Model

The Key/Value model means that data can be stored in the form of key/value pair where the values can be of simple and complex types. It can support documents and graph elements as values.

The following table illustrates the comparison between relational model, key/value model, and OrientDB key/value model.

Relational Model	Key/Value Model	OrientDB Key/Value Model
Table	Bucket	Class or Cluster
Row	Key/Value pair	Document
Column	Not available	Document field or Vertex/Edge property
Relationship	Not available	Link

The Object Model
This model has been inherited by Object Oriented programming and supports **Inheritance** between types (sub-types extends the super-types), **Polymorphism** when you refer to a base class and **Direct binding** from/to Objects used in programming languages.

The following table illustrates the comparison between relational model, Object model, and OrientDB Object model.

Relational Model	Object Model	OrientDB Object Model
Table	Class	Class or Cluster
Row	Object	Document or Vertex
Column	Object property	Document field or Vertex/Edge property
Relationship	Pointer	Link

Following are some of the important terminologies in OrientDB.
Record
The smallest unit that you can load from and store in the database. Records can be stored in four types.

- Document
- Record Bytes
- Vertex
- Edge

Record ID
When OrientDB generates a record, the database server automatically assigns a unit identifier to the record, called RecordID (RID). The RID looks like #<cluster>:<position>. <cluster> means cluster identification number and the <position> means absolute position of the record in the cluster.

Documents
The Document is the most flexible record type available in OrientDB. Documents are softly typed and are defined by schema classes with defined constraint, but you can also insert the document without any schema, i.e. it supports schema-less mode too.
Documents can be easily handled by export and import in JSON format. For example, take a look at the following JSON sample document. It defines the document details.

```
{
  "id"       : "1201",
  "name"     : "Jay",
  "job"      : "Developer",
```

```
  "creations" : [
    {
      "name"    : "Amiga",
      "company" : "Commodore Inc."
    },

    {
      "name"    : "Amiga 500",
      "company" : "Commodore Inc."
    }
  ]
}
```

RecordBytes
Record Type is the same as BLOB type in RDBMS. OrientDB can load and store document Record type along with binary data.

Vertex
OrientDB database is not only a Document database but also a Graph database. The new concepts such as Vertex and Edge are used to store the data in the form of graph. In graph databases, the most basic unit of data is node, which in OrientDB is called a vertex. The Vertex stores information for the database.

Edge
There is a separate record type called the Edge that connects one vertex to another. Edges are bidirectional and can only connect two vertices. There are two types of edges in OrientDB, one is regular and another one lightweight.

Class
The class is a type of data model and the concept drawn from the Object-oriented programming paradigm. Based on the traditional document database model, data is stored in the form of collection, while in the Relational database model data is stored in tables. OrientDB follows the Document API along with OPPS paradigm. As a concept, the class in OrientDB has the closest relationship with the table in relational databases, but (unlike tables) classes can be schema-less, schema-full or mixed. Classes can inherit from other classes, creating trees of classes. Each class has its own cluster or clusters, (created by default, if none are defined).

Cluster
Cluster is an important concept which is used to store records, documents, or vertices. In simple words, Cluster is a place where a group of records are stored. By default, OrientDB will create one cluster per class. All the records of a class are stored in the same cluster having the same name as the class. You can create up to 32,767(2^15-1) clusters in a database.
The CREATE class is a command used to create a cluster with specific name. Once the cluster is created you can use the cluster to save records by specifying the name during the creation of any data model.

Relationships
OrientDB supports two kinds of relationships: referenced and embedded. Referenced relationships means it stores direct link to the target objects of the relationships. Embedded relationships means it stores the relationship within the record that embeds it. This relationship is stronger than the reference relationship.

Database

The database is an interface to access the real storage. IT understands high-level concepts such as queries, schemas, metadata, indices, and so on. OrientDB also provides multiple database types. For more information on these types, see Database Types.

RESULT:
The OrientDB Graph database concepts are examined along with its features successfully.

EX No:	MySQL Database Creation, Table Creation, Query.
Date :	

AIM:
To execute the basic queries like database creation, table creation and perform basic queries on tables in MYSQL.

PROCEDURE:

1. Create database.
2. Create the needed Tables
   A. Consider the following schema for a LibraryDatabase:
      1. BOOK (Book_id, Title, Publisher_Name, Pub_Year)
      2. BOOK_AUTHORS (Book_id, Author_Name)
      3. PUBLISHER (Name, Address, Phone)
      4. BOOK_COPIES(Book_id, Branch_id, No-of_Copies)
      5. BOOK_LENDING (Book_id, Branch_id, Card_No, Date_Out, Due_Date)
      6. LIBRARY_BRANCH (Branch_id, Branch_Name, Address)
3. Insert needed number of values (tuples) into the tables
4. Perform the various queries given below:
   1. Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each branch, etc.
   2. Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun2017
   3. Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation.
   4. Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.
   5. Create a view of all books and its number of copies that are currently available in the Library.

QUERIES - SYNTAX:

1. **Database Creation**

CREATE DATABASE LIBRARYDATABASE;

USE LIBRARYDATABASE;

2. **Table Creation**

CREATE TABLE PUBLISHER (NAME VARCHAR (20) PRIMARY KEY, PHONE BIGINT, ADDRESS VARCHAR (20));

CREATE TABLE BOOK (BOOK_ID INTEGER PRIMARY KEY, TITLE VARCHAR (20), PUBLISHER_NAME VARCHAR(20), PUB_YEAR VARCHAR (20), FOREIGN KEY (PUBLISHER_NAME) REFERENCES PUBLISHER (NAME) ON DELETE CASCADE);

CREATE TABLE BOOK_AUTHORS (BOOK_ID INTEGER, AUTHOR_NAME VARCHAR (20),FOREIGN KEY(BOOK_ID) REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, PRIMARY KEY (BOOK_ID, AUTHOR_NAME));

CREATE TABLE LIBRARY_BRANCH (BRANCH_ID INTEGER PRIMARY KEY, BRANCH_NAME VARCHAR (50), ADDRESS VARCHAR (50));
CREATE TABLE BOOK_COPIES (NO_OF_COPIES INTEGER, BOOK_ID INTEGER, BRANCH_ID INTEGER, FOREIGN KEY (BOOK_ID) REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, FOREIGN KEY(BRANCH_ID) REFERENCES LIBRARY_BRANCH (BRANCH_ID) ON DELETE CASCADE, PRIMARY KEY (BOOK_ID, BRANCH_ID));

CREATE TABLE CARD (CARD_NO INTEGER PRIMARY KEY);

CREATE TABLE BOOK_LENDING (DATE_OUT DATE, DUE_DATE DATE, BOOK_ID INTEGER, BRANCH_ID INTEGER, CARD_NO INTEGER, FOREIGN KEY (BOOK_ID) REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, FOREIGN KEY (BRANCH_ID) REFERENCES LIBRARY_BRANCH (BRANCH_ID) ON DELETE CASCADE, FOREIGN KEY (CARD_NO) REFERENCES CARD (CARD_NO) ON DELETE CASCADE, PRIMARY KEY (BOOK_ID, BRANCH_ID, CARD_NO));

3. Insertion of Values to Tables

INSERT INTO PUBLISHER VALUES (MCGRAW-HILL‘, 9989076587,’BANGALORE‘);

INSERT INTO PUBLISHER VALUES (PEARSON‘, 9889076565,’NEWDELHI‘);

INSERT INTO PUBLISHER VALUES (RANDOM HOUSE‘, 7455679345,‘HYDERABAD‘);

INSERT INTO PUBLISHER VALUES (HACHETTE LIVRE‘, 8970862340,‘CHENNAI‘);

INSERT INTO PUBLISHER VALUES (GRUPOPLANETA‘,7756120238,’BANGALORE‘);

INSERT INTO BOOK VALUES (1,‘DBMS‘ ,‘JAN-2017‘, ‘MCGRAW-HILL‘);

INSERT INTO BOOK VALUES (2,‘ADBMS‘ ,‘JUN-2016‘, ‘MCGRAW-HILL‘);

INSERT INTO BOOK VALUES (3,‘CN‘ ,‘SEP-2016‘, ‘PEARSON‘);

INSERT INTO BOOK VALUES (4,‘CG‘ ,‘SEP-2015‘, ‘GRUPO PLANETA‘);

INSERT INTO BOOK VALUES (5,‘OS‘ ,‘MAY-2016‘, ‘PEARSON‘);

INSERT INTO BOOK_AUTHORS VALUES (‘NAVATHE‘, 1);

INSERT INTO BOOK_AUTHORS VALUES (‘NAVATHE‘, 2);

INSERT INTO BOOK_AUTHORS VALUES (‘TANENBAUM‘, 3);

INSERT INTO BOOK_AUTHORS VALUES (‘EDWARD ANGEL‘, 4);

INSERT INTO BOOK_AUTHORS VALUES (‘GALVIN‘, 5);

INSERT INTO LIBRARY_BRANCH VALUES (10,‘RR NAGAR‘,‘BANGALORE‘);

INSERT INTO LIBRARY_BRANCH VALUES (11,‘RNSIT‘,‘BANGALORE‘);

INSERT INTO LIBRARY_BRANCH VALUES (12,‘RAJAJI NAGAR‘, ‘BANGALORE‘);

INSERT INTO LIBRARY_BRANCH VALUES (13,‘NITTE‘,‘MANGALORE‘);

INSERT INTO LIBRARY_BRANCH VALUES (14,‘MANIPAL‘,‘UDUPI‘);

INSERT INTO BOOK_COPIES VALUES (10, 1, 10);

INSERT INTO BOOK_COPIES VALUES (5, 1,11);

INSERT INTO BOOK_COPIES VALUES (2, 2,12);

INSERT INTO BOOK_COPIES VALUES (5, 2,13);

INSERT INTO BOOK_COPIES VALUES (7, 3,14);

INSERT INTO BOOK_COPIES VALUES (1, 5,10);

INSERT INTO BOOK_COPIES VALUES (3, 4,11);

INSERT INTO CARD VALUES (100);

INSERT INTO CARD VALUES (101);

INSERT INTO CARD VALUES (102);

INSERT INTO CARD VALUES (103);

INSERT INTO CARD VALUES (104);

INSERT INTO BOOK_LENDING VALUES (‘17-JAN-07‘,‘17-JUN-01‘, 1, 10, 101);

INSERT INTO BOOK_LENDING VALUES (‘17-JAN-11‘,‘17-MAR-11‘, 3, 14, 101);

INSERT INTO BOOK_LENDING VALUES (‘17-FEB-21‘,‘17-APR-21‘, 2, 13, 101);

INSERT INTO BOOK_LENDING VALUES (‘17-MAR-15‘,‘17-JUL-15‘, 4, 11, 101);

INSERT INTO BOOK_LENDING VALUES (_17-APR-12‘,‘17-MAY-12‘, 1, 11, 104);

4. <u>Basic Queries</u>

1. Query to Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each branch, etc.

SELECT B.BOOK_ID, B.TITLE, B.PUBLISHER_NAME, A.AUTHOR_NAME, C.NO_OF_COPIES, L.BRANCH_ID FROM BOOK B, BOOK_AUTHORS A, BOOK_COPIES C, LIBRARY_BRANCHL WHEREB.BOOK_ID=A.BOOK_ID AND B.BOOK_ID=C.BOOK_ID AND L.BRANCH_ID=C.BRANCH_ID;

2. Query to Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun2017.

SELECT CARD_NO FROM BOOK_LENDING WHERE DATE_OUT BETWEEN ‘01-JAN-2017‘ AND ‘01-JUL-2017‘ GROUP BY CARD_NO HAVING COUNT (*)>3;

3. Query to Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation.

DELETE FROM BOOK WHERE BOOK_ID=3;

4. Query to Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.

CREATE VIEW V_PUBLICATION AS SELECT PUB_YEAR FROM BOOK;

5. Query to Create a view of all books and its number of copies that are currently available in the Library.

CREATE VIEW V_BOOKS AS SELECT B.BOOK_ID, B.TITLE, C.NO_OF_COPIES FROM BOOK B, BOOK_COPIES C, LIBRARY_BRANCH L WHERE B.BOOK_ID=C.BOOK_ID AND C.BRANCH_ID=L.BRANCH_ID;

RESULT :

The queries to create database, create table and query table have been executed successfully.

EX No:	MySQL Replication – Distributed Databases
Date :	

AIM:
To implement Replication in distributed database using MYSQL.

THEORY :
MYSQL - Replication
MySQL supports replication capabilities that allow the databases on one server to be made available on another server. Replication is used for many purposes. For example, by replicating your databases, you have multiple copies available in case a server crashes or goes offline. Clients can use a different server if the one that they normally use becomes unavailable. Replication also can be used to distribute client load. Rather than having a single server to which all clients connect, you can set up multiple servers that each handle a fraction of the client load.

MySQL replication uses a master/slave architecture:
- The server that manages the original databases is the master.
- Any server that manages a copy of the original databases is a slave.
- A given master server can have many slaves, but a slave can have only a single master. (If done with care, it is possible to set up two-way or circular replication, but this study guide does not describe how.)

A replication slave is set up initially by transferring an exact copy of the to-be-replicated databases from the master server to the slave server. Thereafter, each replicated database is kept synchronized to the original database. When the master server makes modifications to its databases, it sends those changes to each slave server, which makes the changes to its copy of the replicated databases.

PROCEDURE:
Setting Up Replication
To set up replication, each slave requires the following:
- A backup copy of the master's databases. This is the replication "baseline" that sets the slave to a known initial state of the master.
- The filename and position within the master's binary log that corresponds to the time of the backup. The values are called the "replication coordinates." They are needed so that the slave can tell the master that it wants all updates made from that point on.
- An account on the master server that the slave can use for connecting to the master and requesting updates. The account must have the global REPLICATION SLAVE privilege. For example, you can set up an account for a slave by issuing these statements on the master server, where slave_user and slave_pass are the username and password for the account, and slave_host is the host from which the slave server will connect:

```
mysql> CREATE USER 'slave_user'@'slave_host' IDENTIFIED BY 'slave_pass';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'slave_user'@'slave_host';
```

Also, you must assign a unique ID value to each server that will participate in your replication setup. ID values are positive integers in the range from 1 to 2^{32}

1. The easiest way to assign these ID values is by placing a server-id option in each server's option file:
[mysqld] server-id=id_value

It's common, though not required, to use an ID of 1 for the master server and values greater than 1 for the slaves. The following procedure describes the general process for setting up replication.

1. Ensure that binary logging is enabled on the master server. If it is not, stop the server, enable logging, and restart the server.
2. On the master server, make a backup of all databases to be replicated. One way to do this is by using mysqldump:

   shell> mysqldump --all-databases --master-data=2 > dump_file

   Assuming that binary logging is enabled, the --master-data=2 option causes the dump file to include a comment containing a CHANGE MASTER statement that indicates the replication coordinates as of the time of the backup. These coordinates can be used later when you tell the slave where to begin replicating in the master's binary log.
3. Copy the dump file to the replication slave host and load it into the MySQL server on that machine:

   shell> mysql < dump_file
4. Tell the slave what master to connect to and the position in the master's binary log at which to begin replicating. To do this, connect to the slave server and issue a CHANGE MASTER statement:

   mysql> CHANGE MASTER TO -> MASTER_HOST = 'master_host_name', -> MASTER_USER = 'slave_user', -> MASTER_PASSWORD = 'slave_pass', -> MASTER_LOG_FILE = 'master_log_file', -> MASTER_LOG_POS = master_log_pos;

   The hostname is the host where the master server is running. The username and password are those for the slave account that you set up on the master. The log file and position are the replication coordinates in the master's binary log. (You can get these from the CHANGE MASTER statement near the beginning of the dump file.)

After you perform the preceding procedure, issue a START SLAVE statement. The slave should connect to the master and begin replicating updates that the master sends to it. The slave also creates a master.info file in its data directory and records the values from the CHANGE MASTER statement in the file. As the slave reads updates from the master, it changes the replication coordinates in the master.info file accordingly. Also, when the slave restarts in the future, it looks in this file to determine which master to use.

By default, the master server logs updates for all databases, and the slave server replicates all updates that it receives from the master. For more fine-grained control, it's possible to tell a master which databases to log updates for, and to tell a slave which of those updates that it receives from the master to apply. You can either name databases to be replicated (in which case those not named are ignored), or you can name databases to ignore (in which case those not named are replicated). The master host options are --binlog-do-db and --binlog-ignore-db. The slave host options are --replicate-do-db and --replicate-ignore-db.

The following example illustrates how this works, using the options that enable replication for specific databases. Suppose that a master server has three databases named a, b, and c. You can elect to replicate only databases a and b when you start the master server by placing these options in an option file read by that server:

[mysqld] binlog-do-db = a binlog-do-db = b

With those options, the master server will log updates only for the named databases to the binary log. Thus, any slave server that connects to the master will receive information only for databases a and b.

Enabling binary logging only for certain databases has an unfortunate side effect: Data recovery operations require both your backup files and your binary logs, so for any database not logged in the binary log, full recovery cannot be performed. For this reason, you might prefer to have the master log changes for all databases to the binary log, and instead filter updates on the slave side.
A slave that takes no filtering action will replicate all events that it receives. If a slave should replicate events only for certain databases, such as databases a and c, you can start it with these lines in an option file:

[mysqld] replicate-do-db = a replicate-do-db = c

RESULT:
The MYSQL replication was executed successfully.

EX No:	Spatial data storage and retrieval in MySQL
Date :	

AIM:

To create a spatial data storage and retrieve data in mysql.

PROCEDURE:

Step 1: Start the MYSQL server.
Step 2: Create a database and set that database.
Step 3: Create table with spatial column and insert the data.
Step 4: Use select statement to retrieve and view the content.

QUERIES:

Creating Spatial Columns:
Use the CREATE TABLE statement to create a table with a spatial column:
CREATE TABLE geom (g GEOMETRY);

Use the ALTER TABLE statement to add or drop a spatial column to or from an existing table
ALTER TABLE geom ADD pt POINT;
ALTER TABLE geom DROP pt;

Populating Spatial Columns
After you have created spatial columns, you can populate them with spatial data. Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format.
The following examples demonstrate how to insert geometry values into a table by converting WKT values to internal geometry format:

Perform the conversion directly in the INSERT statement:

```
INSERT INTO geom VALUES (ST_GeomFromText('POINT(1 1)'));

SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (ST_GeomFromText(@g));
```

Perform the conversion prior to the INSERT:

```
SET @g = ST_GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

To insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (ST_GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (ST_GeomFromText(@g));

SET @g ='GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (ST_GeomFromText(@g));
```

To use ST_GeomFromText() function to create geometry values. We can also use type-specific functions:

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (ST_PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (ST_LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (ST_PolygonFromText(@g));

SET @g ='GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (ST_GeomCollFromText(@g));
```

Inserting a POINT(1 1) value with hex literal syntax:

```
INSERT INTO geom VALUES
(ST_GeomFromWKB(X'010100000000000000000F03F000000000000F03F'));
```

An ODBC application can send a WKB representation, binding it to a placeholder using an argument of BLOB type:

```
INSERT INTO geom VALUES (ST_GeomFromWKB(?))
```

Fetching Spatial Data:

Geometry values stored in a table can be fetched in internal format. You can also convert them to WKT or WKB format. Fetching spatial data in internal format:Fetching geometry values using internal format can be useful in table-to-table transfers:

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

Fetching spatial data in WKT format:The ST_AsText() function converts a geometry from internal format to a WKT string.

```
SELECT ST_AsText(g) FROM geom;
```

Fetching spatial data in WKB format:The ST_AsBinary() function converts a geometry from internal format to a BLOB containing the WKB value.

```
SELECT ST_AsBinary(g) FROM geom;
```

RESULT:

The spatial data storage creation and retrieve of data in mysql has been executed successfully.

EX No: Date :	Temporal data storage and retrieval in MySQL

AIM:
To create a Temporal data storage and retrieval in MySQL.

PROCEDURE:

Step 1: Start the MYSQL server.
Step 2: Create a database and set that database.
Step 3: Create table with temporal column and insert the data.
Step 4: Use select statement to retrieve and view the content.

THEORY :
TEMPORAL DATATYPE
MySQL provides data types for storing different kinds of temporal information. In the following descriptions, the terms YYYY, MM, DD, hh, mm, and ss stand for a year, month, day of month, hour, minute, and second value, respectively.
The following table summarizes the storage requirements and ranges for the date and time data types.

Type	Storage Required	Range
DATE	3 bytes	'1000-01-01' to '9999-12-31'
TIME	3 bytes	'-838:59:59' to '838:59:59'
DATETIME	8 bytes	'1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP	4 bytes	'1970-01-01 00:00:00' to mid-year 2037
YEAR	1 byte	1901 to 2155 (for YEAR(4)), 1970 to 2069 (for YEAR(2))

QUERIES:

To Create table with temporal data
mysql> CREATE TABLE ts_test1 (->ts1 TIMESTAMP, ->ts2 TIMESTAMP, ->fdata CHAR(30) ->);
Query OK, 0 rows affected (0.00 sec)

To describe the table schema
mysql> DESCRIBE ts_test1;
+-------+-----------+------+-----+--------------------+-------+ | Field | Type | Null | Key | Default | Extra | +-------+-----------+------+-----+--------------------+-------+ | ts1 | timestamp | YES | | CURRENT_TIMESTAMP | | | ts2 | timestamp | YES | | 0000-00-00 00:00:00 | | | data | char(30) | YES | | NULL | | +-------+-----------+------+-----+--------------------+-------+
3 rows in set (0.01 sec)

To insert
mysql> INSERT INTO ts_test1 (data) VALUES ('original_value');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM ts_test1;
+---------------------+---------------------+----------------+ | ts1 | ts2 | data | +---------------------+---------------------+----------------+ | 2005-01-04 14:45:51 | 0000-00-00 00:00:00 | original_value | +---------------------+---------------------+----------------+

1 row in set (0.00 sec)

To update

```
mysql> UPDATE ts_test1 SET data='updated_value';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

To retrieve

```
mysql> SELECT * FROM ts_test1;
+---------------------+---------------------+---------------+ | ts1                 | ts2                 | data          | +----
---------------+---------------------+---------------+ | 2005-01-04 14:46:17 | 0000-00-00 00:00:00 |
updated_value | +---------------------+---------------------+---------------+ 1 row in set (0.00 sec)

 mysql> CREATE TABLE ts_test2 (     ->   created_time TIMESTAMP DEFAULT
CURRENT_TIMESTAMP,     ->   data CHAR(30)     -> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO ts_test2 (data) VALUES ('original_value');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM ts_test2;
+---------------------+----------------+ | created_time        | data           | +---------------------+---------------
-+ | 2005-01-04 14:46:39 | original_value | +---------------------+----------------+
1 row in set (0.00 sec)

mysql> UPDATE ts_test2 SET data='updated_value';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM ts_test2;
+---------------------+---------------+ | created_time        | data          | +---------------------+---------------+
| 2005-01-04 14:46:39 | updated_value | +---------------------+---------------+
1 row in set (0.00 sec)

 mysql> CREATE TABLE ts_test3 (     ->   updated_time TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,     ->   data CHAR(30)     -> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO ts_test3 (data) VALUES ('original_value'); Query OK, 1 row affected (0.00
sec)

mysql> SELECT * FROM ts_test3;
+---------------------+----------------+ | updated_time        | data           | +---------------------+---------------
--+ | 0000-00-00 00:00:00 | original_value | +---------------------+----------------+
1 row in set (0.00 sec)

mysql> UPDATE ts_test3 SET data='updated_value';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM ts_test3;
+---------------------+---------------+ | updated_time        | data          | +---------------------+---------------
+ | 2005-01-04 14:47:10 | updated_value | +---------------------+---------------+
1 row in set (0.00 sec)
```

```
mysql> CREATE TABLE ts_test4 (    ->   created TIMESTAMP DEFAULT
CURRENT_TIMESTAMP,    ->   updated TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,    ->   data CHAR(30)    -> );
ERROR 1293 (HY000): Incorrect table definition; there can be only one TIMESTAMP column with
CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause

mysql> CREATE TABLE ts_test5 (    ->   created TIMESTAMP DEFAULT 0,    ->   updated
TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,    ->   data CHAR(30)    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO ts_test5 (created, data)    -> VALUES (NULL, 'original_value');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM ts_test5; +--------------------+--------------------+----------------+ | created
| updated            | data           | +--------------------+--------------------+----------------+ | 2005-01-04
14:47:39 | 0000-00-00 00:00:00 | original_value | +--------------------+--------------------+----------------
--+
1 row in set (0.00 sec)

mysql> UPDATE ts_test5 SET data='updated_value';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM ts_test5;
+--------------------+--------------------+----------------+ | created            | updated            | data           |
+--------------------+--------------------+----------------+ | 2005-01-04 14:47:39 | 2005-01-04 14:47:52 |
updated_value | +--------------------+--------------------+----------------+ 1 row in set (0.00 sec)

mysql> CREATE TABLE ts_null (ts TIMESTAMP NULL);
Query OK, 0 rows affected (0.04 sec)

mysql> DESCRIBE ts_null;
+-------+-----------+------+-----+---------+-------+ | Field | Type      | Null | Key | Default | Extra | +------
-+-----------+------+-----+---------+-------+ | ts    | timestamp | YES  |     | NULL    |       | +-------+-------
----+------+-----+---------+-------+ 1 row in set (0.10 sec)

mysql> SELECT @@global.time_zone, @@session.time_zone;
+--------------------+---------------------+ | @@global.time_zone | @@session.time_zone | +-------------
-------+---------------------+ | SYSTEM             | SYSTEM              | +--------------------+------------------
---+
1 row in set (0.00 sec)

mysql> SET time_zone = '+00:00';
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @@session.time_zone; +---------------------+ | @@session.time_zone | +-------------
--------+ | +00:00              | +---------------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE ts_test (ts TIMESTAMP);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO ts_test (ts) VALUES (NULL);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM ts_test;
+---------------------+ | ts                  | +---------------------+ | 2005-01-04 20:50:18 | +---------------------
+ 1 row in set (0.00 sec)

mysql> SET time_zone = '+02:00';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM ts_test;
+---------------------+ | ts                  | +---------------------+ | 2005-01-04 22:50:18 | +---------------------
+
1 row in set (0.00 sec)

mysql> SET time_zone = '-05:00';
Query OK, 0 rows affected (0.00 sec)
mysql>
SELECT * FROM ts_test; +---------------------+ | ts                  | +---------------------+ | 2005-01-04
15:50:18 | +---------------------+ 1 row in set (0.00 sec)

mysql> SELECT CONVERT_TZ('2005-01-27 13:30:00', '+01:00', '+03:00');
+-----------------------------------------------------+ | CONVERT_TZ('2005-01-27 13:30:00', '+01:00',
'+03:00') | +-----------------------------------------------------+ | 2005-01-27 15:30:00
| +-----------------------------------------------------+
1 row in set (0.00 sec)
```

RESULT:
The Temporal data storage and retrieval in MySQL is executed successfully.

EX No:	Object storage and retrieval
Date :	

AIM:
To create and execute Object data storage and retrieval.

PROCEDURE:

Step 1: Start the Oracle server.
Step 2: Connect to the server through the client.
Step 3: Create a database and set that database.
Step 4: Create table and insert the data.
Step 5: Use select statement to retrieve and view the content.

QUERY:

To create an object type:

```
CREATE TYPE StockItem_objtyp;
CREATE TYPE LineItem_objtyp;
CREATE TYPE PurchaseOrder_objtyp;
CREATE TYPE PhoneList_vartyp AS VARRAY(10) OF VARCHAR2(20);

CREATE TYPE Address_objtyp AS OBJECT (
  Street        VARCHAR2(200),
  City          VARCHAR2(200),
  State         CHAR(2),
  Zip           VARCHAR2(20)
  )

CREATE TYPE Customer_objtyp AS OBJECT (
  CustNo          NUMBER,
  CustName        VARCHAR2(200),
  Address_obj     Address_objtyp,
  PhoneList_var    PhoneList_vartyp,

  ORDER MEMBER FUNCTION
    compareCustOrders(x IN Customer_objtyp) RETURN INTEGER
  )

CREATE TYPE LineItem_objtyp AS OBJECT (
  LineItemNo   NUMBER,
  Stock_ref    REF StockItem_objtyp,
  Quantity     NUMBER,
  Discount     NUMBER
  )
```

To create a table

```
CREATE TABLE Customer_objtab OF Customer_objtyp (CustNo PRIMARY KEY)
   OBJECT ID PRIMARY KEY ;
```

```
CREATE TABLE Stock_objtab OF StockItem_objtyp (StockNo PRIMARY KEY)   OBJECT ID
PRIMARY KEY ;

CREATE TABLE PurchaseOrder_objtab OF PurchaseOrder_objtyp (  /* Line 1 */
  PRIMARY KEY (PONo),                                  /* Line 2 */
  FOREIGN KEY (Cust_ref) REFERENCES Customer_objtab)       /* Line 3 */
  OBJECT ID PRIMARY KEY                                /* Line 4 */
  NESTED TABLE LineItemList_ntab STORE AS PoLine_ntab (     /* Line 5 */
   (PRIMARY KEY(NESTED_TABLE_ID, LineItemNo))           /* Line 6 */
   ORGANIZATION INDEX COMPRESS)                        /* Line 7 */
  RETURN AS LOCATOR                                    /* Line 8 */
```

To alter table

```
ALTER TABLE PoLine_ntab
  ADD (SCOPE FOR (Stock_ref) IS stock_objtab) ;

CREATE OR REPLACE TYPE BODY PurchaseOrder_objtyp AS
  MAP MEMBER FUNCTION getPONo RETURN NUMBER is
    BEGIN
      RETURN PONo;
    END;
  MEMBER FUNCTION sumLineItems RETURN NUMBER IS
    i         INTEGER;
    StockVal   StockItem_objtyp;
    Total      NUMBER := 0;
  BEGIN
    IF (UTL_COLL.IS_LOCATOR(LineItemList_ntab)) -- check for locator
      THEN
        SELECT SUM(L.Quantity * L.Stock_ref.Price) INTO Total
        FROM   TABLE(CAST(LineItemList_ntab AS LineItemList_ntabtyp)) L;
    ELSE
      FOR i in 1..SELF.LineItemList_ntab.COUNT LOOP
        UTL_REF.SELECT_OBJECT(LineItemList_ntab(i).Stock_ref,StockVal);
        Total := Total + SELF.LineItemList_ntab(i).Quantity *
                                          StockVal.Price;
      END LOOP;
    END IF;
  RETURN Total;
  END;
END;

ALTER TABLE PoLine_ntab
  ADD (SCOPE FOR (Stock_ref) IS stock_objtab);
```

To insert

```
INSERT INTO Stock_objtab VALUES(1004, 6750.00, 2) ;
INSERT INTO Stock_objtab VALUES(1011, 4500.23, 2) ;
INSERT INTO Stock_objtab VALUES(1534, 2234.00, 2) ;
INSERT INTO Stock_objtab VALUES(1535, 3456.23, 2) ;

INSERT INTO Customer_objtab
 VALUES (
   1, 'Jean Nance',
   Address_objtyp('2 Avocet Drive', 'Redwood Shores', 'CA', '95054'),
```

```
   PhoneList_vartyp('415-555-1212')
   ) ;

INSERT INTO Customer_objtab
  VALUES (
   2, 'John Nike',
   Address_objtyp('323 College Drive', 'Edison', 'NJ', '08820'),
   PhoneList_vartyp('609-555-1212','201-555-1212')
   ) ;

INSERT INTO PurchaseOrder_objtab
  SELECT  1001, REF(C),
        SYSDATE, '10-MAY-1999',
        LineItemList_ntabtyp(),
        NULL
   FROM   Customer_objtab C
   WHERE  C.CustNo = 1 ;

INSERT INTO PurchaseOrder_objtab
  SELECT  2001, REF(C),
        SYSDATE, '20-MAY-1997',
        LineItemList_ntabtyp(),
        Address_objtyp('55 Madison Ave','Madison','WI','53715')
   FROM   Customer_objtab C
   WHERE  C.CustNo = 2 ;

INSERT INTO TABLE (
  SELECT  P.LineItemList_ntab
   FROM   PurchaseOrder_objtab P
   WHERE  P.PONo = 1001
  )

SELECT  02, REF(S), 10, 10
   FROM   Stock_objtab S
   WHERE  S.StockNo = 1535 ;

INSERT INTO TABLE (
  SELECT  P.LineItemList_ntab
   FROM   PurchaseOrder_objtab P
   WHERE  P.PONo = 2001
  )
  SELECT  10, REF(S), 1, 0
   FROM   Stock_objtab S
   WHERE  S.StockNo = 1004 ;

INSERT INTO TABLE (
  SELECT  P.LineItemList_ntab
   FROM   PurchaseOrder_objtab P
   WHERE  P.PONo = 2001
  )
  VALUES(11, (SELECT REF(S)
    FROM  Stock_objtab S
    WHERE S.StockNo = 1011), 2, 1) ;
```

```
SELECT  p.PONo
 FROM   PurchaseOrder_objtab p
 ORDER BY VALUE(p) ;
```

Customer and Line Item Data for Purchase Order 1001

```
SELECT  DEREF(p.Cust_ref), p.ShipToAddr_obj, p.PONo,
       p.OrderDate, LineItemList_ntab
 FROM   PurchaseOrder_objtab p
 WHERE  p.PONo = 1001 ;
```

Total Value of Each Purchase Order

```
SELECT   p.PONo, p.sumLineItems()
 FROM    PurchaseOrder_objtab p ;
```

Purchase Order and Line Item Data Involving Stock Item 1004

```
SELECT   po.PONo, po.Cust_ref.CustNo,
        CURSOR (
          SELECT  *
           FROM   TABLE (po.LineItemList_ntab) L
           WHERE  L.Stock_ref.StockNo = 1004
          )
 FROM    PurchaseOrder_objtab po ;

SELECT   po.PONo, po.Cust_ref.CustNo, L.*
 FROM    PurchaseOrder_objtab po, TABLE (po.LineItemList_ntab) L
 WHERE   L.Stock_ref.StockNo = 1004 ;

SELECT   po.PONo, po.Cust_ref.CustNo, L.*
 FROM    PurchaseOrder_objtab po, TABLE (po.LineItemList_ntab) (+) L
 WHERE   L.Stock_ref.StockNo = 1004 ;

SELECT   AVG(L.DISCOUNT)
 FROM    PurchaseOrder_objtab po, TABLE (po.LineItemList_ntab) L ;
```

To delete

```
DELETE
 FROM   PurchaseOrder_objtab
 WHERE  PONo = 1001 ;
```

RESULT:

The creation and execution of Object storage and retrieval was executed successfully.

EX No:	XML Databases, XML table creation, XQuery FLWOR expression
Date :	

AIM:
To create and execute XML Databases , XML table creation, XQuery FLWOR expression.

PROCEDURE:

Step 1: Start the Oracle server.
Step 2: Connect to the server through the client.
Step 3: Create a database and set that database.
Step 4: Create table and insert the data.
Step 5: Use select statement (FLWOR) to retrieve and view the content.

QUERIES:

To create table
CREATE TABLE mytable1 (key_column VARCHAR2(10) PRIMARY KEY, xml_column XMLType);

Table created.

CREATE TABLE mytable2 OF XMLType;

Table created.

To insert values:
INSERT INTO mytable2 VALUES (XMLType(bfilename('XMLDIR', 'purchaseOrder.xml'),
nls_charset_id('AL32UTF8')));

To retrieve using XQuery
```
SELECT XMLQuery('for $i in /PurchaseOrder
           where $i/CostCenter eq "A10"
             and $i/User eq "SMCCAIN"
           return <A10po pono="{$i/Reference}"/>'
          PASSING OBJECT_VALUE
          RETURNING CONTENT)
  FROM purchaseorder;
```

XMLQUERY('FOR$IIN/PURCHASEORDERWHERE$I/COSTCENTEREQ"A10"AND$I/USEREQ"SMCCAIN"RET

<A10po pono="SMCCAIN-20021009123336151PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336341PDT"></A10po>
<A10po pono="SMCCAIN-20021009123337173PDT"></A10po>
<A10po pono="SMCCAIN-20021009123335681PDT"></A10po>
<A10po pono="SMCCAIN-20021009123335470PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336972PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336842PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336512PDT"></A10po>
<A10po pono="SMCCAIN-20021009123333894PDT"></A10po>

<A10po pono="SMCCAIN-20021009123337403PDT"></A10po>

XML File:

```
<PurchaseOrder>
 <Reference>SBELL-20021009123336O1PDT</Reference>
 <Actions>
  <Action>
   <User>SVOLLMAN</User>
  </Action>
 </Actions>
. . .
</PurchaseOrder>

<PurchaseOrder>
 <Reference>ABEL-20021127121040897PST</Reference>
 <Actions>
  <Action>
   <User>ZLOTKEY</User>
  </Action>
  <Action>
   <User>KING</User>
  </Action>
 </Actions>
. . .
</PurchaseOrder>
```

RESULT:

The creation and execution of XML Databases , XML table creation, XQuery FLWOR expression has been completed successfully.

MADHA ENGINEERING COLLEGE

(Affiliated to Anna University and Approved by AICTE, New Delhi) Madha Nagar, Kundrathur, Chennai-600069

DEPARTMENT OF Master of Computer Application

MC4112

Python Programming Laboratory

R-2021

LAB MANUAL

CO-PO Mapping

CO	POs					
	PO1	PO2	PO3	PO4	PO5	PO6
1	2	1	3	3	2	2
2	2	1	3	3	3	3
3	3	1	3	3	3	3
4	3	1	3	3	3	3
5	3	1	3	3	3	3
Avg	2.6	1	3	3	2.8	2.8

MC4112 PYTHON PROGRAMMING LABORATORY **L T P C**
0 0 4 2

COURSE OBJECTIVES:

- Develop Python programs with conditionals, loops and functions
- Represent compound data using Python lists, tuples, dictionaries
- Read and write data from/to files in Python
- Implement NumPy, Pandas, Matplotlib libraries
- Implement object oriented concepts

LIST OF EXPERIMENTS:

Note: The examples suggested in each experiment are only indicative. The lab instructor is expected to design other problems on similar lines.

1. Python programming using simple statements and expressions (exchange the values of two variables, circulate the values of n variables, distance between two points).
2. Scientific problems using Conditionals and Iterative loops.
3. Linear search and Binary search
4. Selection sort, Insertion sort
5. Merge sort, Quick Sort
6. Implementing applications using Lists, Tuples.
7. Implementing applications using Sets, Dictionaries.
8. Implementing programs using Functions.
9. Implementing programs using Strings.
10. Implementing programs using written modules and Python Standard Libraries (pandas, numpy, Matplotlib, scipy)
11. Implementing real-time/technical applications using File handling.
12. Implementing real-time/technical applications using Exception handling.
13. Creating and Instantiating classes

HARDWARE/SOFTWARE REQUIREMENTS

1: Processors: Intel Atom® processor Intel®Core™i3 processor
2: Disk space: 1GB.
3: Operating systems: Windows 7,macOS and Linux
4: Python versions:2.7, 3.6, 3.8

Ex.no

Date:

1.EXCHANGE OF VALUE OF TWO VARIABLES

AIM:

To write a python program to exchange the value of two variables.

ALGORITHM:

1.Define a function swap and call the function.
2.Get the first number
3.Get the second number
4.Print the numbers before swapping
5.Assign the first number with temporary variable temp
6.Store the second number to the first number
7.Reassign the second number to the temporary variable
8.Print the Swapped values.

Ex.no

Date:

PROGRAM:

```
P =  int ( input("Please enter value for P: "))
Q = int( input("Please enter value for Q: "))
P, Q = Q, P
print ("The Value of P after swapping: ", P)
print ("The Value of Q after swapping: ", Q)
```

Ex.no

Date:

OUTPUT:

```
Please enter value for P: 3
Please enter value for Q: 8
The Value of P after swapping: 8
The Value of Q after swapping: 3
```

RESULT:

Thus, the Python Program to exchange the value of two variable is executed successfully and the output is verified.

Ex.no

Date:

2.CIRCULATE THE VALUES OF n VARIABLES

AIM:

To write a Python Program to circulate the values of n variables.

ALGORITHM:

First element is removed and appended to the end position and all the values in the list are moved by one position.
This repeats until they reach their original position.
Print statement is given inside the loop to display the element position each time they are moved.

\

Ex.no

Date:

PROGRAM:

```
no_of_terms = int(input("Enter number of values : "))
list1 = []
for val in range(0,no_of_terms,1):
ele = int(input("Enter integer : "))
list1.append(ele)
print("Circulating the elements of list ", list1)
for val in range(0,no_of_terms,1):
ele = list1.pop(0)
list1.append(ele)
print(list1)
```

Ex.no

Date:

OUTPUT:

```
Enter number of values: 5
Enter integer:3
Enter integer:9
Enter integer:5
Enter integer:6
Enter integer:7
['Circulating the elements of list',[3, 9, 5, 6, 7])
[9, 5, 6, 7, 3]
[5, 6, 7, 3, 9]
[6, 7, 3, 9, 5]
[7, 3, 9, 5, 6]
[3, 9, 5, 6, 7]
```

RESULT:

Thus, the Python Program to circulate the values of n variables is executed successfully and the output is verified

Ex.no

Date:

3.FIND THE DISTANCE BETWEEN TWO POINTS

AIM:

To write a python program to find the distance between two variables.

ALGORITHM:

1.Define a function distance with four variables and call the function.
2.Declare the four variables as x1,x2,y1 and y2.
3.To find the distance calculate:
result= ((((x2 – x1) *2) + ((y2-y1)2))*0.5)
4.Then print the result.

Ex.no

Date:

PROGRAM:

```
x1=int(input("enter x1 : "))
x2=int(input("enter x2 : "))
y1=int(input("enter y1 : "))
y2=int(input("enter y2 : "))
result= ((((x2 - x1 )*2) + ((y2-y1)2) )*0.5)
print("distance between",(x1,x2),"and",(y1,y2),"is : ",result)
```

Ex.no

Date:

OUTPUT:

```
Enter x1:5
Enter x2:8
Enter y1:2
Enter y2:9
distance between (5, 8) and (2, 9) is: 7.615773105863909
```

RESULT:

Thus, the Python Program to find the distance between two points is executed successfully and the output is verified

Ex.no

Date:

4.LINEAR SEARCH

AIM:

To write a python program to perform Linear Search.

ALGORITHM:

Step - 1: Start the search from the first element and Check with each element of list x.
Step - 2: If element is found, return the index position of the key.
Step - 3: If element is not found, return element is not present.
We have created a function linear_Search(), which takes three arguments - list1, length of the list, and number to search.
We defined for loop and iterate each element and compare to the key value.
If element is found, return the index else return -1 which means element is not present in the list

Ex.no

Date:

PROGRAM:

```
def linear_Search(list1, n, key):
for i in range(0, n):
if (list1[i] == key):
return i
return -1
list1 = [1 ,3, 5, 4, 7, 9]
key = 7
n = len(list1)
res = linear_Search(list1, n, key)
if(res == -1):
print("Element not found")
else:
print("Element found at index: ", res)
```

Ex.no

Date:

OUTPUT:

```
Enter size of list :- 5
Enter the array of 0 element :- 9
Enter the array of 1 element :- 6
Enter the array of 2 element :- 7
Enter the array of 3 element :- 3
Enter the array of 4 element :- 2
Enter number to search in list :- 7
7 was found at index 2.
Enter number to search in list :- 1
1 was not found.
```

RESULT:

Thus, the Python Program to perform Linear Search is executed successfully and the output is verified.

Ex.no

Date:

5.BINARY SEARCH

AIM:

To write a python program to perform Binary Search.

ALGORITHM:

1. Read the search element
2. Find the middle element in the sorted list
3. Compare the search element with the middle element
i. if both are matching, print element found
ii. else then check if the search element is smaller or larger
than the middle element
4. If the search element is smaller than the middle element, then repeat steps 2 and 3 for the left sublist of the middle element
5. If the search element is larger than the middle element, then repeat steps 2 and 3 for the right sublist of the middle element
6. Repeat the process until the search element if found in the list
7.If the element is not found,loop terminates.

Ex.no

Date:

PROGRAM:

```
def binarySearch(arr, l, r, x):
if r >= l:
mid = l + (r - l) // 2
if arr[mid] == x:
return mid
elif arr[mid] > x:
return binarySearch(arr, l, mid-1, x)
else:
return binarySearch(arr, mid + 1, r, x)
else:
return -1
arr = [2, 3, 4, 10, 40]
x = 3
result = binarySearch(arr, 0, len(arr)-1, x)
if result != -1:
print("Element is present at index % d" % result)
else:
print("Element is not present in array")
```

Ex.no

Date:

OUTPUT:

// If x=3
Element is present at index 1
/// If x=1
Element is not present in array

RESULT:

Thus, the Python Program to perform Binary Search is executed successfully and the output is verified.

Ex.no

Date:

6.QUICK SORT

AIM:

To write a python program to perform Quick Sort.

ALGORITHM:

The QuickSort algorithm operates by picking a pivot element from
the array and splitting the other items into two sub-arrays based on
whether they are bigger or less than the pivot.
1.A pivot element is used to split an array into subarrays.
2.The pivot element should be positioned in such a way that items less than the pivot are maintained on the left side of the pivot and elements bigger than the pivot are kept on the right side of the pivot while splitting the array.
3.The same method is used to split the left and right subarrays.
4.This method is repeated until each subarray has just one
element.
5.The components have already been sorted at this stage. The items are finally merged to produce a sorted array.

Ex.no

Date:

PROGRAM

```
def partition(array,left_index,right_index) :
pivot = array[left_index]
i = left_index
j = right_index
while True :
while True :
i += 1
if i == right_index or array[i] > pivot :
break
while True :
j -= 1
if array[j] <= pivot :
break
if i < j :
temp = array[i]
array[i] = array[j]
array[j] = temp
if i > j :
break
temp = array[j]
array[j] = array[left_index] array[left_index] = temp
return j
def quicksort(array , left_index , right_index) :
if left_index < right_index :
partion_index = partition(array,left_index,right_index)
quicksort(array,left_index,partion_index)
quicksort(array,partion_index+1,right_index)
array = list(map(int,input("Enter space separated elements as input :- ").split(" ")))
quicksort(array,0,len(array))
print("Sorted input array in ascending order is :- ",array)
```

Ex.no

Date:

OUTPUT:

```
Enter space separated elements as input :- 6 5 3 4 8 9 7 2 1
Sorted input array in ascending order is :- [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

RESULT:

Thus, the Python Program to perform Quick Sort is executed successfully and the output is verified.

Ex.no

Date:

7.MERGE SORT

AIM:

To write a python program to perform Merge Sort.

ALGORITHM:

1.Create a function named mergeSort
2.Find the mid of the list
3.Assign l=arr[:a] and r=arr[a:]
4.Initialise b=c=d=0
5.while b<len(l) and c<len(r),perform the following if l[b]<r[c]:
arr[d]=l[b]
b+=1 else
arr[d]=r[c]
c+=1
d+=1
6.while b<len(l),perform the following arr[d]=l[b]
b+=1
d+=1
7.while c<len(r),perform the following arr[d]=r[c]
c+=1
d+=1
8.Pritn the sorted list.

Ex.no

Date:

PROGRAM:

```
def mergeSort(arr):
if len(arr) > 1:
a = len(arr)//2
l = arr[:a]
r = arr[a:]
mergeSort(l)
mergeSort(r)
b = c = d = 0
while b < len(l) and c < len(r):
if l[b] < r[c]:
arr[d] = l[b]
b += 1
else:
arr[d] = r[c]
c += 1
d += 1
while b < len(l):
arr[d] = l[b]
b += 1
d += 1
while c < len(r): arr[d] = r[c]
c += 1
d += 1
def printList(arr):
for i in range(len(arr)):
print(arr[i], end=" ")
print() arr = [0,3,2,1,9,8,7,6,5,4]
mergeSort(arr)
print("Sorted array is: ")
```

Ex.no

Date:

OUTPUT:

Sorted array is:
0 1 2 3 4 5 6 7 8 9

RESULT:

Thus, the Python Program to perform MergeSort is executed successfully and the output is verified

Ex.no

Date:

8. IMPLEMENTING APPLICATIONS USING LISTS

AIM:

To write a Python Program to implementing applications using Lists.

ALGORITHM:

STEP 1: Start Python IDLE
STEP 2: Initialize the Values
STEP 3: Declaring the values as New York, Los Angles, Boston and Denver.
STEP 4: Using Keyword in print statement to get our desired
output.
STEP 5: Again initialize the values in list2.
STEP 6: Declaring the value as 1, 3, 4, 6, 4, 7, 8, 2, 3.
STEP 7: Using Keyword in print statement to get our desired
output.

Ex.no

Date:

PROGRAM:

```
list = [ "New York", "Los Angles", "Boston", "Denver" ]
print(list)
print(list[0])
list2 = [1,3,4,6,4,7,8,2,3]
print(sum(list2))
print(min(list2))
print(max(list2))
print(list2[0])
print(list2[-1])
```

Ex.no

Date:

OUTPUT:

['New York', 'Los Angles', 'Boston', 'Denver']
New York
38
1
8
1
3

RESULT:

Thus, the Python Program to implementing applications using lists is executed successfully and the output is verified

Ex.no

Date:

9. IMPLEMENTING APPLICATIONS USING TUPLES

AIM:

To write a Python Program to implementing applications using Tuples.

ALGORITHM:

STEP 1: Start Python IDLE.
STEP 2: Declaring tuple1 and using tuple1 declaring statement as “Enter the tuple element”.
STEP 3: Using for loop to the elements giving as input.
STEP 4: Implementing applications in tuples one by one till the end of the element given.
STEP 5: Stop the program.

Ex.no

Date:

PROGRAM:

```
tuple1 = tuple(input("Enter the tuple elements ..."))
print(tuple1)
count = 0
for i in tuple1:
print("tuple1[%d] = %s"%(count, i))
count = count+1
```

Ex.no

Date:

OUTPUT:

```
Enter the tuple elements ...987654
('9', '8', '7', '6', '5', '4')
tuple1[0] = 9
tuple1[1] = 8
tuple1[2] = 7
tuple1[3] = 6
tuple1[4] = 5
tuple1[5] = 4
```

RESULT:

Thus, the Python Program to implementing applications using tuples is executed successfully and the output is verified.

Ex.no

Date:

10. IMPLEMENTING APPLICATIONS USING SETS

AIM:

To write a Python Program to implementing applications using sets.

ALGORITHM:

STEP 1: Start Python IDLE.
STEP 2: Initialize the elements using the keyword set by declaring it as “Days”.
STEP 3: Declaring the elements as “Monday, Tuesday,
Wednesday, Thursday, Friday, Saturday, Sunday “.
STEP 4: Print the desired output using the respective conditions.
STEP 5: Stop the program.

Ex.no

Date:

PROGRAM:

```
Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday", "Sunday"])
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
print(i)
```

Ex.no

Date:

OUTPUT:

```
{'Wednesday', 'Thursday', 'Monday', 'Saturday', 'Friday', 'Sunday', 'Tuesday'}
<class 'set'>
looping through the set elements ...
Wednesday
Thursday
Monday
Saturday
Friday
Sunday
Tuesday
```

RESULT:

Thus, the Python Program to implementing applications using sets is executed successfully and the output is verified.

Ex.no

Date:

11. IMPLEMENTING APPLICATIONS USING DICTIONARIES

AIM:

To write a Python Program to implementing applications using Dictionaries.

ALGORITHM:

STEP 1: Start Python IDLE.
STEP 2: Declare the employees detail such as Name: John, Age: 29, Salary: 25000, Company; GOOGLE.
STEP 3: Then use print statement to print all the employee's details that we have given in Step-2.
STEP 4: To print the details of the new employee.
STEP 5: Initialize Employee's Name, Age, Salary, Company so that We can declare the details after the run module.
STEP 6: Conclude the program with print(Employee).
STEP 7: Stop the program.

Ex.no

Date:

PROGRAM:

```
Employee = {"Name": "John", "Age": 29,
"salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
print("Enter the details of the new employee....");
Employee["Name"] = input("Name: ");
Employee["Age"] = int(input("Age: "));
Employee["salary"] = int(input("Salary: "));
Employee["Company"] = input("Company:");
print("printing the new data");
print(Employee)
```

Ex.no

Date:

OUTPUT:

```
<class 'dict'>
printing Employee data ....
{'Name': 'John', 'Age': 29, 'salary': 25000, 'Company': 'GOOGLE'}
Enter the details of the new employee....
Name: GOPAL
Age: 21
Salary: 40000
Company:TCS
printing the new data
{'Name': 'GOPAL', 'Age': 21, 'salary': 40000, 'Company': 'TCS'}
```

RESULT:

Thus, the Python Program to implementing applications using dictionaries is executed successfully and the output is verified.

Ex.no

Date:

12. IMPLEMENTING APPLICATIONS USING FUNCTIONS

AIM:

To write a Python Program to implementing applications
using functions.

ALGORITHM:

STEP 1: Start Python IDLE.
STEP 2: Declare function as tri_recursion.
STEP 3: For recursion,use the formula as result =
k+tri_recursion(k-1)
STEP 4: The value of k will decrease by 1 with each recursive call.
STEP 5: Recursion will stop once the k becomes 0.
STEP 6: Print and Stop the program.

Ex.no

Date:

PROGRAM:

```
def tri_recursion(k):
if(k > 0):
result = k + tri_recursion(k - 1)
print(result)
else:
result = 0
return result
print("\n\nTri-Recursion")
tri_recursion(10)
```

Ex.no

Date:

OUTPUT:

Tri-Recursion
1
3
6
10
15
21
28
36
45
55

RESULT:

Thus, the Python Program to implementing applications using functions is executed successfully and the output is verified.

Ex.no

Date:

13. IMPLEMENTING REAL-TIME/TECHNICAL APPLICATIONS USING EXCEPTION HANDLING

AIM:

To write the Python Program to implementing Real-time/Technical applications using Exception Handling

ALGORITHM:

STEP 1: Start Python IDLE.
STEP 2: Initialize the values.
STEP 3: Use the try statement for arithmetic expression.
STEP 4: Check the if condition and print the result.
STEP 5: if the if condition is not equal then use else block
statement as Exception Handling.
STEP 6: Create files and use open and close methods.
STEP 7: Use split method to split the given sentences.
STEP 8: Stop the program

Ex.no

Date:

PROGRAM:

```
a=int(input("Enter a="))
b=int(input("Enter b="))
try:
c = ((a+b) / (a-b))
if a==b:
raise ZeroDivisionError
except ZeroDivisionError:
print ("a/b result in 0")
else:
print (c)
```

Ex.no

Date:

OUTPUT:

Enter a=4
Enter b=4
a/b result in 0

RESULT:

Thus, the Python Program to implementing real-time/technical applications using exception handlingis executed successfully and the output is verified.

MADHA ENGINEERING COLLEGE

(Affiliated to Anna University and Approved by AICTE, New Delhi) Madha Nagar, Kundrathur, Chennai-600069

DEPARTMENT OF Master of Computer Application

MC4211

Advanced Database Technology

Laboratory

R-2021

LAB MANUAL

4	2	1	2	2	2	2
5	2	1	2	2	2	2
Avg	2	1	2	2	2	2

MC4211 ADVANCED DATABASE TECHNOLOGY LABORATORY L T P C
0 0 4 2

COURSE OBJECTIVES:

- To understand the process of distributing tables across multiple systems
- To understand the process of storing, retrieving spatial and temporal data
- To understand the process of storing, retrieving objects in a database
- To understand the process of storing and retrieving data from a XML Database
- To use the open source database for building a mobile application

LIST OF EXPERIMENTS:

1. NOSQL Exercises
   a. MongoDB – CRUD operations, Indexing, Sharding
   b. Cassandra: Table Operations, CRUD Operations, CQL Types
   c. HIVE: Data types, Database Operations, Partitioning – HiveQL
   d. OrientDB Graph database – OrientDB Features
2. MySQL Database Creation, Table Creation, Query
3. MySQL Replication – Distributed Databases
4. Spatial data storage and retrieval in MySQL
5. Temporal data storage and retrieval in MySQL
6. Object storage and retrieval in MySQL
7. XML Databases , XML table creation, XQuery FLWOR expression
8. Mobile Database Query Processing using open source DB (MongoDB/MySQL etc)

TOTAL: 60 PERIODS

SOFTWARE REQUIREMENTS

1. Java / Python / R / Scala
2. Oracle, MySQL, MongoDB, Casandra, Hive

COURSE OUTCOMES:

On completion of the course, the student will be able to:

CO1: Design and implement advanced databases.
CO2: Use big data frameworks and tools.
CO3: Formulate complex queries using SQL.
CO4: Create an XML document and perform Xquery.
CO5: Query processing in Mobile databases using open source tools.

CO-PO Mapping

CO	**POs**					
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**
1	2	1	2	2	2	2

EX No: Date :	NOSQL EXERCISES MongoDB – CRUD Operations, Indexing, Sharding, Deployment

AIM:
To execute the queries to perform CRUD operations, Indexing, Sharding, Deployment in MongoDB.

PROCEDURE:

Step 1: Start the mongo deamon and run it in behind
Step 2: Start the mongo client
Step 3: Create the database
Step 4: Perform basic queries to perform CRUD (Create, Read, Update, Delete) operations.
Step 5: Perform basic queries for Indexing, Sharding and Deployment.

QUERIES:

//to start mongo deamon
C:/> mongod

//to start mongo client
C:/> mongo

//to list out database names
> show dbs

CRUD Queries:

//to create database
> use db1

//to check in which database I am working
> db

//to drop database in which I am working
> db.dropDatabase()

//To create collection
> db.createCollection('stud')

//to list out collection names
> show collections

//create collection by inserting document
> db.emp.insert({rno:1,name:'Bhavana'})

//Every row/document can be different than other

```
> db.emp.insert({name:'Amit',rno:2})
> db.emp.insert({rno:3, email_id:'a@gmail.com'})
```

// To display data from collection

```
> db.emp.find()
{ "_id" : ObjectId("5d7d3daf315728b4998f522e"), "rno" : 1, "name" : "Bhavana" }
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }
{ "_id" : ObjectId("5d7d3f56315728b4998f5230"), "rno" : 3, "email_id" : "a@gmail.com" }
```

//insert data by providing _id value

```
> db.emp.insert({_id:1,rno:4,name:"Akash"})

> db.emp.find()
{ "_id" : ObjectId("5d7d3daf315728b4998f522e"), "rno" : 1, "name" : "Bhavana" }
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }
{ "_id" : ObjectId("5d7d3f56315728b4998f5230"), "rno" : 3, "email_id" : "a@gmail.com" }
{ "_id" : 1, "rno" : 4, "name" : "Akash" }
```

// trying to insert data with duplicate _id, it will not accept as _id is primary key field

```
> db.emp.insert({_id:1,rno:5,name:"Reena"})
E11000 duplicate key error index: db1.emp.$_id_  dup key: { : 1.0 }
```

//Insert multiple documents at once

```
> db.emp.insert([{rno:7,name:'a'},{rno:8,name:'b'},{rno:8,name:'c'}])

> db.emp.find()
{ "_id" : ObjectId("5d7d3daf315728b4998f522e"), "rno" : 1, "name" : "Bhavana" }
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }
{ "_id" : ObjectId("5d7d3f56315728b4998f5230"), "rno" : 3, "email_id" : "a@gmail.com" }
{ "_id" : 1, "rno" : 4, "name" : "Akash" }
{ "_id" : 2, "rno" : 5, "name" : "Reena" }
{ "_id" : ObjectId("5d7d4244315728b4998f5231"), "rno" : 7, "name" : "a" }
{ "_id" : ObjectId("5d7d4244315728b4998f5232"), "rno" : 8, "name" : "b" }
{ "_id" : ObjectId("5d7d4244315728b4998f5233"), "rno" : 8, "name" : "c" }
```

// to insert multiple values for one key using []

```
> db.emp.insert({rno:10,name:'Ankit',hobbies:['singing','cricket','swimming'],age:21})

> db.emp.find()
{ "_id" : ObjectId("5d7d3daf315728b4998f522e"), "rno" : 1, "name" : "Bhavana" }
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }
{ "_id" : ObjectId("5d7d3f56315728b4998f5230"), "rno" : 3, "email_id" : "a@gmail.com" }
{ "_id" : 1, "rno" : 4, "name" : "Akash" }
{ "_id" : 2, "rno" : 5, "name" : "Reena" }
{ "_id" : ObjectId("5d7d4244315728b4998f5231"), "rno" : 7, "name" : "a" }
```

{ "_id" : ObjectId("5d7d4244315728b4998f5232"), "rno" : 8, "name" : "b" }
{ "_id" : ObjectId("5d7d4244315728b4998f5233"), "rno" : 8, "name" : "c" }
{ "_id" : ObjectId("5d7d433a315728b4998f5234"), "rno" : 10, "name" : "Ankit", "hobbies" : ["singing", "cricket", "swimming"], "age" : 21 }

// Embedded document example

> db.emp.insert({rno:11, Name: {Fname:"Bhavana", Mname:"Amit", Lname:"Khivsara"}})
> db.emp.insert({rno:12, Name: "Janvi", Address:{Flat:501, Building:"Sai Appart", area:"Tidke colony", city: "Nashik", state:"MH", pin:423101}, age:22})

// To insert date use ISODate function

> db.emp.insert({rno:15, name:'Ravina', dob: ISODate("2019-09-14")})

> db.emp.find()
{ "_id" : ObjectId("5d7d3daf315728b4998f522e"), "rno" : 1, "name" : "Bhavana" }
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }
{ "_id" : ObjectId("5d7d3f56315728b4998f5230"), "rno" : 3, "email_id" : "a@gmail.com" }
{ "_id" : 1, "rno" : 4, "name" : "Akash" }
{ "_id" : 2, "rno" : 5, "name" : "Reena" }
{ "_id" : ObjectId("5d7d4244315728b4998f5231"), "rno" : 7, "name" : "a" }
{ "_id" : ObjectId("5d7d4244315728b4998f5232"), "rno" : 8, "name" : "b" }
{ "_id" : ObjectId("5d7d4244315728b4998f5233"), "rno" : 8, "name" : "c" }
{ "_id" : ObjectId("5d7d433a315728b4998f5234"), "rno" : 10, "name" : "Ankit", "hobbies" : ["singing", "cricket", "swimming"], "age" : 21 }
{ "_id" : ObjectId("5d7d4462315728b4998f5235"), "rno" : 11, "Name" : { "Fname" : "Bhavana", "Mname" : "Amit", "Lname" : "Khivsara" } }
{ "_id" : ObjectId("5d7d4574315728b4998f5236"), "rno" : 12, "Name" : "Janvi", "Address" : { "Flat" : 501, "Building" : "Sai Appart", "area" : "Tidke colony", "city" : "Nashik", "state" : "MH", "pin" : 423101 }, "age" : 22 }
{ "_id" : ObjectId("5d7d465d315728b4998f5237"), "rno" : 15, "name" : "Ravina", "dob" : ISODate("2019-09-14T00:00:00Z") }
>

// Multi embedded document with data function

> db.emp.insert({rno:17, name:"Ashika",date:Date(), awards:[{name:"Best c -Designer", year:2010, prize:"winner"},{name:"Wen site competition",year:2012,prize:"Runner-up"},{name:"Fashion show", year:2015,prize:"winner"}], city:"Nashik"})

// ouput using pretty command

> db.emp.find().pretty()
{
	"_id" : ObjectId("5d7d3daf315728b4998f522e"),
	"rno" : 1,
	"name" : "Bhavana"
}
{ "_id" : ObjectId("5d7d3f28315728b4998f522f"), "name" : "Amit", "rno" : 2 }

```
{
        "_id" : ObjectId("5d7d3f56315728b4998f5230"),
        "rno" : 3,
        "email_id" : "a@gmail.com"
}
{ "_id" : 1, "rno" : 4, "name" : "Akash" }
{ "_id" : 2, "rno" : 5, "name" : "Reena" }
{ "_id" : ObjectId("5d7d4244315728b4998f5231"), "rno" : 7, "name" : "a" }
{ "_id" : ObjectId("5d7d4244315728b4998f5232"), "rno" : 8, "name" : "b" }
{ "_id" : ObjectId("5d7d4244315728b4998f5233"), "rno" : 8, "name" : "c" }
{
        "_id" : ObjectId("5d7d433a315728b4998f5234"),
        "rno" : 10,
        "name" : "Ankit",
        "hobbies" : [
                "singing",
                "cricket",
                "swimming"
        ],
        "age" : 21
}
{
        "_id" : ObjectId("5d7d4462315728b4998f5235"),
        "rno" : 11,
        "Name" : {
                "Fname" : "Bhavana",
                "Mname" : "Amit",
                "Lname" : "Khivsara"
        }
}
{
        "_id" : ObjectId("5d7d4574315728b4998f5236"),
        "rno" : 12,
        "Name" : "Janvi",
        "Address" : {
                "Flat" : 501,
                "Building" : "Sai Appart",
                "area" : "Tidke colony",
                "city" : "Nashik",
                "state" : "MH",
                "pin" : 423101
        },
        "age" : 22
}
{
        "_id" : ObjectId("5d7d465d315728b4998f5237"),
```

```
        "rno" : 15,
        "name" : "Ravina",
        "dob" : ISODate("2019-09-14T00:00:00Z")
}
{
        "_id" : ObjectId("5d7d4aa7315728b4998f5238"),
        "rno" : 17,
        "name" : "Ashika",
        "date" : "Sat Sep 14 2019 16:16:39 GMT-0400 (EDT)",
        "awards" : [
                {
                        "name" : "Best C-designer",
                        "year" : 2010,
                        "prize" : "winner"
                },
                {
                        "name" : "Wen site competition",
                        "year" : 2012,
                        "prize" : "Runner-up"
                },
                {
                        "name" : "Fashion show",
                        "year" : 2015,
                        "prize" : "winner"
                }
        ],
        "city" : "Nashik"
}
```

// New collection for Find operation

```
> db.stud.insert([{rno:1, name:'Ashiti'}, {rno:2,name:'Savita'}, {rno:3,name:'Sagar'},
{rno:4,name:'Reena'},{rno:5,name:'Jivan'}])
```

//Simple Find Command

```
> db.stud.find()
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
```

//Find command with Condition

```
> db.stud.find({rno:5})
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
```

//Find command with condition with giving name field only to show

```
> db.stud.find({rno:5},{name:1})
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "name" : "Jivan" }
```

//Find command with condition with giving name field only to show and _id to hide

```
> db.stud.find({rno:5},{name:1,_id:0})
{ "name" : "Jivan" }
```

// Find command to show only names without condition

```
> db.stud.find({},{name:1,_id:0})
{ "name" : "Ashiti" }
{ "name" : "Savita" }
{ "name" : "Sagar" }
{ "name" : "Reena" }
{ "name" : "Jivan" }
```

// To display data whose rno is greater than 2

```
> db.stud.find({rno:{$gt:2}})
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
```

// To display data whose rno is less than equal to 2

```
> db.stud.find({rno:{$lte:2}})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }
```

// To display data whose rno is less than 2

```
> db.stud.find({rno:{$lt:2}})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
```

// To display data whose rno is not equal to 2

```
> db.stud.find({rno:{$ne:2}})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
```

// To display data whose rno is either 1 or 3 or 5 using in operator

```
> db.stud.find({rno:{$in:[1,3,5]}})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
```

// To display data whose rno is either 1 or 3 or 5 or 7 or 9 using in operator

```
> db.stud.find({rno:{$in:[1,3,5,7,9]}})
```

{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }

//Sorting Command -1 is for Descending

> db.stud.find().sort({rno:-1})
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }

//Sorting Command 1 is for Ascending

> db.stud.find().sort({name:1})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }

//Dispay rno & name whose rno is greater than 2. Show output in decending order by rno

> db.stud.find({rno:{$gt:2}},{_id:0}).sort({rno:-1})
{ "rno" : 5, "name" : "Jivan" }
{ "rno" : 4, "name" : "Reena" }
{ "rno" : 3, "name" : "Sagar" }

//Collection with 3 and 5 rollno as duplicate values

> db.stud.find()
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
{ "_id" : ObjectId("5d83b8d9a44331f62bcd836e"), "rno" : 5, "name" : "Radhika" }
{ "_id" : ObjectId("5d83b8eba44331f62bcd836f"), "rno" : 3, "name" : "Manioj" }

//Distinct command to show only unique values for roll no

> db.stud.distinct("rno")
[1, 2, 3, 4, 5]

// Limit use to show only some records from starting- following command shows only first 2 records from collection

> db.stud.find().limit(2)
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }

{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }

// Skip use to show all records after skipping some records- following command shows all records after first 2 records from collection

```
> db.stud.find().skip(2)
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
{ "_id" : ObjectId("5d83b8d9a44331f62bcd836e"), "rno" : 5, "name" : "Radhika" }
{ "_id" : ObjectId("5d83b8eba44331f62bcd836f"), "rno" : 3, "name" : "Manioj" }
```

// Shows documents where name starting with A

```
> db.stud.find({name:/^A/})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
```

// Shows documents where name ending with i

```
> db.stud.find({name:/i$/})
{ "_id" : ObjectId("5d83af5aa44331f62bcd8369"), "rno" : 1, "name" : "Ashiti" }
```

// Shows documents where name having letter a anywhere

```
> db.stud.find({name:/a/})
{ "_id" : ObjectId("5d83af5aa44331f62bcd836a"), "rno" : 2, "name" : "Savita" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836b"), "rno" : 3, "name" : "Sagar" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836c"), "rno" : 4, "name" : "Reena" }
{ "_id" : ObjectId("5d83af5aa44331f62bcd836d"), "rno" : 5, "name" : "Jivan" }
{ "_id" : ObjectId("5d83b8d9a44331f62bcd836e"), "rno" : 5, "name" : "Radhika" }
{ "_id" : ObjectId("5d83b8eba44331f62bcd836f"), "rno" : 3, "name" : "Manioj" }
>
```

//findOne to show only first record

```
> db.stud.findOne()
{
        "_id" : ObjectId("5d83af5aa44331f62bcd8369"),
        "rno" : 1,
        "name" : "Ashiti"
}
```

// count to show number of documents in collection

```
> db.stud.find().count()
7

> db.stud.find({rno:{$gt:2}}).count()
5
```

//Insert one embedded document(for address)

```
> db.stud.insert({rno:8,address:{area:"College Road",city:"Nashik",state:"MH"},name:"Arya"})
```

//To find document having city as Nashik(as city is key of address key specify "address.city"

```
> db.stud.find({"address.city":"Nashik"})
{ "_id" : ObjectId("5d83c04aa44331f62bcd8370"), "rno" : 8, "address" : { "area" : "College Road", "city" : "Nashik", "state" : "MH" }, "name" : "Arya" }
```

//Insert one document with multiple values(eg hobbies)

```
> db.stud.insert({rno:9,hobbies:['singing','dancing','cricket']})
```

//To use find command on multi values attribute(eg hobbies)

```
> db.stud.find({hobbies:'dancing'})
{ "_id" : ObjectId("5d83c165a44331f62bcd8371"), "rno" : 9, "hobbies" : [  "singing",  "dancing",  "cricket" ] }
```

//$unset will remove the column rno from document matching the given condition

```
> db.stud.update({rno:1},{$unset:{rno:1}})
```

//$set to update the value of rno

```
>db.stud.update({rno:2},{$set:{rno:22}})
```

//upsert use to update document if condition found otherwise insert document with updates values.

```
> db.stud.update({rno:50},{$set:{rno:55}},{upsert:true})
```

//multi:true used to update in multiple documents

```
> db.stud.update({rno:5},{$set:{rno:15}},{multiple:true})
```

//It will remove record having rno as 4

```
> db.stud.remove({rno:4})
```

//It will remove only one record having rno as 4

```
> db.stud.remove({rno:4},1)
```

//It will remove all records

```
> db.stud.remove({})
```

Indexing:

//To create index on rno in ascending order(1)- //Single field Index example

```
>db.stud.createIndex({rno:1})
```

//To show the list of Index , v is version, key is on which field you created index
//ns-name space(database name.collection name), name- Name of index given by mongodb

```
>db.stud.getIndexes()
[
        {
```

```
                "v" : 1,
                "key" : {
                        "_id" : 1
                },
                "ns" : "db1.stud",
                "name" : "_id_"
        },
        {
                "v" : 1,
                "key" : {
                        "rno" : 1
                },
                "ns" : "db1.stud",
                "name" : "rno_1"
        }
]
```

//Compound Index Example (-1 is descending & 1 is ascending)

```
>db.stud.createIndex({rno:-1,name:1})

>db.stud.getIndexes()
[
        {
                "v" : 1,
                "key" : {
                        "_id" : 1
                },
                "ns" : "db1.stud",
                "name" : "_id_"
        },
        {
                "v" : 1,
                "key" : {
                        "rno" : 1
                },
                "ns" : "db1.stud",
                "name" : "rno_1"
        },
        {
                "v" : 1,
                "key" : {
                        "rno" : -1,
                        "name" : 1
                },
                "ns" : "db1.stud",
                "name" : "rno_-1_name_1"
        }
]
```

// To drop single index

```
>db.stud.dropIndex({rno:1})
{ "nIndexesWas" : 3, "ok" : 1 }
```

// To drop all indexes at a time

```
>db.stud.dropIndexes()
{
        "nIndexesWas" : 2,
        "msg" : "non-_id indexes dropped for collection",
        "ok" : 1
}
```

RESULT:

The queries to perform CRUD, Indexing, Sharding and Deployment was executed successfully.

EX No:	NOSQL EXERCISES Cassandra: Table Operations, CRUD Operations, CQL Types.
Date :	

AIM:
To execute the queries to perform Table Operations, CRUD operations, CQL Types in Cassandra.

PROCEDURE:

Step 1: Start the Cassandra server and run it in behind.
Step 2: Start the CQL client shell.
Step 3: Create the database
Step 4: Perform basic queries to perform Table operations.
Step 5: Perform basic queries to perform CRUD (Create, Read, Update, Delete) operations.
Step 5: Perform basic queries for CQL Types.

QUERIES:

1. **Create Keyspace:**

```
cqlsh.> CREATE KEYSPACE stud WITH replication = {'class':'SimpleStrategy', 'replication_factor'
: 3};

cqlsh> CREATE KEYSPACE test WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy',
'datacenter1' : 3 } AND DURABLE_WRITES = false;

cqlsh> USE test;
cqlsh:test>
```

2. **Table Operations**

```
cqlsh:test> CREATE TABLE emp( emp_id int PRIMARY KEY, emp_name text, emp_city text,
  emp_sal varint, emp_phone varint);

cqlsh:test> ALTER TABLE emp ADD emp_email text;

cqlsh:test> ALTER TABLE emp DROP emp_email;

cqlsh:test> DROP TABLE emp;

cqlsh:test> TRUNCATE student;
```

3. **CRUD Operations**
   1. **Create data**

```
cqlsh:test> INSERT INTO emp (emp_id, emp_name, emp_city,
  emp_phone, emp_sal) VALUES (1,'ram', 'Hyderabad', 9848022338, 50000);

cqlsh:test> INSERT INTO emp (emp_id, emp_name, emp_city,
  emp_phone, emp_sal) VALUES(2,'robin', 'Hyderabad', 9848022339, 40000);

cqlsh:test> INSERT INTO emp (emp_id, emp_name, emp_city,
  emp_phone, emp_sal) VALUES(3,'rahman', 'Chennai', 9848022330, 45000);
```

2. To read all data from table

```
cqlsh:test> SELECT * FROM emp;

 emp_id |  emp_city | emp_name |  emp_phone | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 | 50000
      2 | Hyderabad |    robin | 9848022339 | 40000
      3 |   Chennai |   rahman | 9848022330 | 45000
 (3   rows)
```

3. To update data in a table

```
cqlsh:test> UPDATE emp SET emp_city='Delhi',emp_sal=50000 WHERE emp_id=2;
cqlsh:test> select * from emp;

 emp_id |  emp_city | emp_name |  emp_phone | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 | 50000
      2 |     Delhi |    robin | 9848022339 | 50000
      3 |   Chennai |   rahman | 9848022330 | 45000
 (3 rows)

cqlsh:test> SELECT emp_name, emp_sal from emp;

 emp_name | emp_sal
----------+---------
      ram | 50000
    robin | 50000
   rajeev | 30000
   rahman | 50000
(4 rows)
```

4. To create index

```
cqlsh:test> CREATE INDEX sal ON emp(emp_sal);
cqlsh:test> SELECT * FROM emp WHERE emp_sal=50000;

 emp_id |  emp_city | emp_name |  emp_phone | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 | 50000
      2 |      null |    robin | 9848022339 | 50000
      3 |   Chennai |   rahman | 9848022330 | 50000
```

5. To drop index

```
cqlsh:test> drop index sal;
```

6. To Deleting Data from a Table

```
cqlsh:tutorialspoint> DELETE emp_sal FROM emp WHERE emp_id=3;
cqlsh:test> select * from emp;

 emp_id |  emp_city | emp_name |  emp_phone | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 | 50000
      2 |     Delhi |    robin | 9848022339 | 50000
      3 |   Chennai |   rahman | 9848022330 | null
(3 rows)
```

To Delete a complete row in a column

```
cqlsh:test> DELETE FROM emp WHERE emp_id=3;
cqlsh:test> select * from emp;

 emp_id |  emp_city | emp_name |  emp_phone | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 | 50000
      2 |     Delhi |    robin | 9848022339 | 50000
(2 rows)
```

4. CQL Types

CQL provides a rich set of built-in data types, including collection types. Along with these data types, users can also create their own custom data types. The following table provides a list of built-in data types available in CQL.

Data Type	Constants	Description
Ascii	Strings	Represents ASCII character string
bigint	Bigint	Represents 64-bit signed long
blob	Blobs	Represents arbitrary bytes
Boolean	Booleans	Represents true or false
counter	Integers	Represents counter column
decimal	integers, floats	Represents variable-precision decimal
double	Integers	Represents 64-bit IEEE-754 floating point
float	integers, floats	Represents 32-bit IEEE-754 floating point
inet	Strings	Represents an IP address, IPv4 or IPv6
int	Integers	Represents 32-bit signed int
text	Strings	Represents UTF8 encoded string
timestamp	integers, strings	Represents a timestamp
timeuuid	Uuids	Represents type 1 UUID
uuid	uuids	Represents type 1 or type 4 UUID
varchar	Strings	Represents uTF8 encoded string
varint	Integers	Represents arbitrary-precision integer

Collection Types

Cassandra Query Language also provides a collection data types. The following table provides a list of Collections available in CQL.

Collection	Description
List	A list is a collection of one or more ordered elements.
Map	A map is a collection of key-value pairs.
Set	A set is a collection of one or more elements.

User-defined datatypes

Cqlsh provides users a facility of creating their own data types. Given below are the commands used while dealing with user defined data types.

- CREATE TYPE − Creates a user-defined data type.
- ALTER TYPE − Modifies a user-defined data type.
- DROP TYPE − Drops a user-defined data type.
- DESCRIBE TYPE − Describes a user-defined data type.
- DESCRIBE TYPES − Describes user-defined data types.

1. **Create a user-defined type named address.**

```
CREATE TYPE mykeyspace.address (
  street text,
  city text,
  zip_code int,
  phones set<text> );
```

2. **Create a user-defined type for the name of a user.**

```
CREATE TYPE mykeyspace.fullname (
  firstname text,
  lastname text);
```

3. **Create a table for storing user data in columns of type fullname and address.**
   Use the frozen keyword in the definition of the user-defined type column.

```
CREATE TABLE mykeyspace.users (
  id uuid PRIMARY KEY,
  name frozen <fullname>,
  direct_reports set<frozen <fullname>>,     // a collection set
  addresses map<text, frozen <address>>      // a collection map);
```

4. **Insert a user's name into the fullname column.**

```
INSERT INTO mykeyspace.users (id, name) VALUES (62c36092-82a1-3a00-93d1-
   46196ee77204, {firstname: 'Marie-Claude', lastname: 'Josset'});
```

5. **Insert an address labeled home into the table.**

```
UPDATE mykeyspace.users SET addresses = addresses + {'home': { street: '191 Rue St.
Charles', city: 'Paris', zip_code: 75015, phones: {'33 6 78 90 12 34'}}} WHERE
id=62c36092-82a1-3a00-93d1-46196ee77204;
```

6. **Retrieve the full name of a user.**

```
SELECT name FROM mykeyspace.users WHERE id=62c36092-82a1-3a00-93d1-46196ee77204;
name
------------------------------------------------
 {firstname: 'Marie-Claude', lastname: 'Josset'}
```

7. **Using dot notation, you can retrieve a component of the user-defined type column**.

```
 SELECT name.lastname FROM mykeyspace.users WHERE id=62c36092-82a1-3a00-93d1-46196ee77204;
 name.lastname
---------------
   Josset
```

8. **To create index**

```
CREATE INDEX on mykeyspace.users (name);
SELECT id FROM mykeyspace.users WHERE name = {firstname: 'Marie-Claude', lastname: 'Josset'};

 id
--------------------------------------
 62c36092-82a1-3a00-93d1-46196ee77204
```

9. To update a complete UDT

```
UPDATE mykeyspace.users SET direct_reports = { ( 'Naoko', 'Murai'), ( 'Sompom', 'Peh') } WHERE id=62c36092-82a1-3a00-93d1-46196ee77204;

INSERT INTO mykeyspace.users (id, direct_reports) VALUES ( 7db1a490-5878-11e2-bcfd-0800200c9a66, { ('Jeiranan', 'Thongnopneua') } );

SELECT direct_reports FROM mykeyspace.users;

 direct_reports
-------------------------------------------------------------------------------
                        {{firstname: 'Jeiranan', lastname: 'Thongnopneua'}}
 {{firstname: 'Naoko', lastname: 'Murai'}, {firstname: 'Sompom', lastname: 'Peh'}}
```

RESULT:

The queries to create keyspace, create table and CQL types have been executed successfully.

EX No: Date :	NOSQL EXERCISES HIVE: Data types, Database Operations, Partitioning – HiveQL

AIM:
To create a database for executing database operations , partitioning and execute hive queries in HIVE.

PROCEDURE:
Step 1: Start the hadoop and run it in behind.
Step 2: Start the derby server and let it run in background.
Step 3: Start yarn/ MapReduce
Step 4: Start network server (use 0.0.0.0 as host address)
Step 5: Start hive. Create the database .Perform basic queries to perform database operations.
Step 5: Perform basic queries to perform partitioning. Perform basic queries for HiveQL .

THEORY and QUERY:

1. **Data types**

All the data types in Hive are classified into four types, given as follows:

- Column Types
- Literals
- Null Values
- Complex Types

Column Types
Column type are used as column data types of Hive. They are as follows:

Integral Types
Integer type data can be specified using integral data types, INT. When the data range exceeds the range of INT, you need to use BIGINT and if the data range is smaller than the INT, you use SMALLINT. TINYINT is smaller than SMALLINT.

The following table depicts various INT data types:

Type	Postfix	Example
TINYINT	Y	10Y
SMALLINT	S	10S
INT	-	10
BIGINT	L	10L

String Types
String type data types can be specified using single quotes (' ') or double quotes (" "). It contains two data types: VARCHAR and CHAR. Hive follows C-types escape characters.

The following table depicts various CHAR data types:

Data Type	Length
VARCHAR	1 to 65355
CHAR	255

Timestamp
It supports traditional UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format “YYYY-MM-DD HH:MM:SS.fffffffff” and format “yyyy-mm-dd hh:mm:ss.fffffffff”.

Dates
DATE values are described in year/month/day format in the form {{YYYY-MM-DD}}.

Decimals
The DECIMAL type in Hive is as same as Big Decimal format of Java. It is used for representing immutable arbitrary precision. The syntax and example is as follows:

DECIMAL (precision, scale)
Decimal (10,0)

Union Types
Union is a collection of heterogeneous data types. You can create an instance using create union. The syntax and example is as follows:

UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>
{0:1}
{1:2.0}
{2:["three","four"]}
{3:{"a":5,"b":"five"}}
{2:["six","seven"]}
{3:{"a":8,"b":"eight"}}
{0:9}
{1:10.0}

Literals
The following literals are used in Hive:

Floating Point Types
Floating point types are nothing but numbers with decimal points. Generally, this type of data is composed of DOUBLE data type.

Decimal Type
Decimal type data is nothing but floating point value with higher range than DOUBLE data type. The range of decimal type is approximately -10-308 to 10308.
Null Value
Missing values are represented by the special value NULL.

Complex Types
The Hive complex data types are as follows:

Arrays
Arrays in Hive are used the same way they are used in Java.
Syntax: ARRAY<data_type>

Maps
Maps in Hive are similar to Java Maps.
Syntax: MAP<primitive_type, data_type>

Structs
Structs in Hive is similar to using complex data with comment.
Syntax: STRUCT<col_name : data_type [COMMENT col_comment], ...>

2. Database Operations and partitions

To create Database:

```
hive> CREATE DATABASE financials;

hive> CREATE DATABASE IF NOT EXISTS financials;
```

To list the Database:

```
hive> SHOW DATABASES;
default
financials

hive> CREATE DATABASE human_resources;

hive> SHOW DATABASES LIKE 'h.*';
human_resources
```

To create database by specifying its location to store

```
hive> CREATE DATABASE financials
   > LOCATION '/my/preferred/directory';
```

To create database with a comment

```
hive> DESCRIBE DATABASE financials;
financials   Holds all financial tables
  hdfs://master-server/user/hive/warehouse/financials.db
```

To create database with additional properties

```
hive> CREATE DATABASE financials
   > WITH DBPROPERTIES ('creator' = 'Mark Moneybags', 'date' = '2021-01-02');
```

To describe the database design

```
hive> DESCRIBE DATABASE financials;
financials   hdfs://master-server/user/hive/warehouse/financials.db

hive> DESCRIBE DATABASE EXTENDED financials;
financials   hdfs://master-server/user/hive/warehouse/financials.db
 {date=2021-01-02, creator=Mark Moneybags);
```

To set a database

```
hive> USE financials;
```

To delete a database

```
hive> DROP DATABASE IF EXISTS financials;
```

To drop a table inside the database before deleting the database

```
hive> DROP DATABASE IF EXISTS financials CASCADE;
```

To alter the database properties

```
hive> ALTER DATABASE financials SET DBPROPERTIES ('edited-by' = 'Joe Dba');
```

To create table

```
CREATE TABLE IF NOT EXISTS mydb.employees (
  name         STRING COMMENT 'Employee name',
  salary       FLOAT  COMMENT 'Employee salary',
```

```
  subordinates ARRAY<STRING> COMMENT 'Names of subordinates',
  deductions   MAP<STRING, FLOAT>
          COMMENT 'Keys are deductions names, values are percentages',
  address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
          COMMENT 'Home address')
COMMENT 'Description of the table'
TBLPROPERTIES ('creator'='me', 'created_at'='2021-01-02 10:00:00', ...)
LOCATION '/user/hive/warehouse/mydb.db/employees';
```

To copy a schema

```
CREATE TABLE IF NOT EXISTS mydb.employees2
LIKE mydb.employees;
```

To list out the tables

```
hive> USE mydb;

hive> SHOW TABLES;
employees
table1
table2

hive> USE default;

hive> SHOW TABLES IN mydb;
employees
table1
table2
```

To describe the table schema

```
hive> DESCRIBE EXTENDED mydb.employees;
name    string  Employee name
salary  float   Employee salary
subordinates    array<string>   Names of subordinates
deductions      map<string,float> Keys are deductions names, values are percentages
address struct<street:string,city:string,state:string,zip:int>  Home address

Detailed Table Information Table(tableName:employees, dbName:mydb, owner:me,
...
location:hdfs://master-server/user/hive/warehouse/mydb.db/employees,
parameters:{creator=me, created_at='2021-01-02 10:00:00',
          last_modified_user=me, last_modified_time=1337544510,
          comment:Description of the table, ...}, ...)
```

To describe the schema of a particular column

```
hive> DESCRIBE mydb.employees.salary;
salary  float   Employee salary
```

To partition the data first by country and then by state:

```
CREATE TABLE employees (
  name         STRING,
  salary       FLOAT,
  subordinates ARRAY<STRING>,
  deductions   MAP<STRING, FLOAT>,
  address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
```

)
PARTITIONED BY (country STRING, state STRING);

To show the partitions
hive> SHOW PARTITIONS employees;
...
Country=CA/state=AB
country=CA/state=BC
...
country=US/state=AL
country=US/state=AK
...

To describe the partitioned extended table
hive> DESCRIBE EXTENDED employees;
name string,
salary float,
...
address struct<...>,
country string,
state string

Detailed Table Information...
partitionKeys:[FieldSchema(name:country, type:string, comment:null),
FieldSchema(name:state, type:string, comment:null)],
...

To delete a table
DROP TABLE IF EXISTS employees;

RESULT:
The database its operations , partitioning and hive queries have been executed successfully in HIVE.

EX No: Date :	NOSQL EXERCISES OrientDB Graph database – OrientDB Features

AIM:
To study about the OrientDB Graph database and its features.

THEORY:
Introduction
OrientDB is an Open Source NoSQL Database Management System, which contains the features of traditional DBMS along with the new features of both Document and Graph DBMS. It is written in Java and is amazingly fast. It can store 220,000 records per second on commodity hardware. OrientDB, is one of the best open-source, multi-model, next generation NoSQL product.

OrientDB is an Open Source NoSQL Database Management System. NoSQL Database provides a mechanism for storing and retrieving NO-relation or NON-relational data that refers to data other than tabular data such as document data or graph data. NoSQL databases are increasingly used in Big Data and real-time web applications. NoSQL systems are also sometimes called "Not Only SQL" to emphasize that they may support SQL-like query languages.

OrientDB also belongs to the NoSQL family. OrientDB is a second generation Distributed Graph Database with the flexibility of Documents in one product with an open source of Apache 2 license.

Features	**MongoDB**	**OrientDB**
Relationships	Uses the RDBMS JOINS to create relationship between entities. It has high runtime cost and does not scale when database scale increases.	Embeds and connects documents like relational database. It uses direct, super-fast links taken from graph database world.
Fetch Plan	Costly JOIN operations.	Easily returns complete graph with interconnected documents.
Transactions	Doesn't support ACID transactions, but it supports atomic operations.	Supports ACID transactions as well as atomic operations.
Query language	Has its own language based on JSON.	Query language is built on SQL.
Indexes	Uses the B-Tree algorithm for all indexes.	Supports three different indexing algorithms so that the user can achieve best performance.
Storage engine	Uses memory mapping technique.	Uses the storage engine name LOCAL and PLOCAL.

OrientDB is the first Multi-Model open source NoSQL DBMS that brings together the power of graphs and flexibility of documents into a scalable high-performance operational database.

The main feature of OrientDB is to support multi-model objects, i.e. it supports different models like Document, Graph, Key/Value and Real Object. It contains a separate API to support all these four models.

Document Model

The terminology Document model belongs to NoSQL database. It means the data is stored in the Documents and the group of Documents are called as Collection. Technically, document means a set of key/value pairs or also referred to as fields or properties.

OrientDB uses the concepts such as classes, clusters, and link for storing, grouping, and analyzing the documents.

The following table illustrates the comparison between relational model, document model, and OrientDB document model

Relational Model	**Document Model**	**OrientDB Document Model**
Table	Collection	Class or Cluster
Row	Document	Document
Column	Key/value pair	Document field
Relationship	Not available	Link

Graph Model

A graph data structure is a data model that can store data in the form of Vertices (Nodes) interconnected by Edges (Arcs). The idea of OrientDB graph database came from property graph. The vertex and edge are the main artifacts of the Graph model. They contain the properties, which can make these appear similar to documents.

The following table shows a comparison between graph model, relational data model, and OrientDB graph model.

Relational Model	**Graph Model**	**OrientDB Graph Model**
Table	Vertex and Edge Class	Class that extends "V" (for Vertex) and "E" (for Edges)
Row	Vertex	Vertex
Column	Vertex and Edge property	Vertex and Edge property
Relationship	Edge	Edge

The Key/Value Model

The Key/Value model means that data can be stored in the form of key/value pair where the values can be of simple and complex types. It can support documents and graph elements as values.

The following table illustrates the comparison between relational model, key/value model, and OrientDB key/value model.

Relational Model	Key/Value Model	OrientDB Key/Value Model
Table	Bucket	Class or Cluster
Row	Key/Value pair	Document
Column	Not available	Document field or Vertex/Edge property
Relationship	Not available	Link

The Object Model
This model has been inherited by Object Oriented programming and supports **Inheritance** between types (sub-types extends the super-types), **Polymorphism** when you refer to a base class and **Direct binding** from/to Objects used in programming languages.

The following table illustrates the comparison between relational model, Object model, and OrientDB Object model.

Relational Model	Object Model	OrientDB Object Model
Table	Class	Class or Cluster
Row	Object	Document or Vertex
Column	Object property	Document field or Vertex/Edge property
Relationship	Pointer	Link

Following are some of the important terminologies in OrientDB.
Record
The smallest unit that you can load from and store in the database. Records can be stored in four types.

- Document
- Record Bytes
- Vertex
- Edge

Record ID
When OrientDB generates a record, the database server automatically assigns a unit identifier to the record, called RecordID (RID). The RID looks like #<cluster>:<position>. <cluster> means cluster identification number and the <position> means absolute position of the record in the cluster.

Documents
The Document is the most flexible record type available in OrientDB. Documents are softly typed and are defined by schema classes with defined constraint, but you can also insert the document without any schema, i.e. it supports schema-less mode too.
Documents can be easily handled by export and import in JSON format. For example, take a look at the following JSON sample document. It defines the document details.

```
{
  "id"       : "1201",
  "name"     : "Jay",
  "job"      : "Developer",
```

```
  "creations" : [
    {
      "name"    : "Amiga",
      "company" : "Commodore Inc."
    },

    {
      "name"    : "Amiga 500",
      "company" : "Commodore Inc."
    }
  ]
}
```

RecordBytes
Record Type is the same as BLOB type in RDBMS. OrientDB can load and store document Record type along with binary data.

Vertex
OrientDB database is not only a Document database but also a Graph database. The new concepts such as Vertex and Edge are used to store the data in the form of graph. In graph databases, the most basic unit of data is node, which in OrientDB is called a vertex. The Vertex stores information for the database.

Edge
There is a separate record type called the Edge that connects one vertex to another. Edges are bidirectional and can only connect two vertices. There are two types of edges in OrientDB, one is regular and another one lightweight.

Class
The class is a type of data model and the concept drawn from the Object-oriented programming paradigm. Based on the traditional document database model, data is stored in the form of collection, while in the Relational database model data is stored in tables. OrientDB follows the Document API along with OPPS paradigm. As a concept, the class in OrientDB has the closest relationship with the table in relational databases, but (unlike tables) classes can be schema-less, schema-full or mixed. Classes can inherit from other classes, creating trees of classes. Each class has its own cluster or clusters, (created by default, if none are defined).

Cluster
Cluster is an important concept which is used to store records, documents, or vertices. In simple words, Cluster is a place where a group of records are stored. By default, OrientDB will create one cluster per class. All the records of a class are stored in the same cluster having the same name as the class. You can create up to 32,767(2^15-1) clusters in a database.
The CREATE class is a command used to create a cluster with specific name. Once the cluster is created you can use the cluster to save records by specifying the name during the creation of any data model.

Relationships
OrientDB supports two kinds of relationships: referenced and embedded. Referenced relationships means it stores direct link to the target objects of the relationships. Embedded relationships means it stores the relationship within the record that embeds it. This relationship is stronger than the reference relationship.

Database

The database is an interface to access the real storage. IT understands high-level concepts such as queries, schemas, metadata, indices, and so on. OrientDB also provides multiple database types. For more information on these types, see Database Types.

RESULT:
The OrientDB Graph database concepts are examined along with its features successfully.

EX No:	MySQL Database Creation, Table Creation, Query.
Date :	

AIM:
To execute the basic queries like database creation, table creation and perform basic queries on tables in MYSQL.

PROCEDURE:

1. Create database.
2. Create the needed Tables
   A. Consider the following schema for a LibraryDatabase:
      1. BOOK (Book_id, Title, Publisher_Name, Pub_Year)
      2. BOOK_AUTHORS (Book_id, Author_Name)
      3. PUBLISHER (Name, Address, Phone)
      4. BOOK_COPIES(Book_id, Branch_id, No-of_Copies)
      5. BOOK_LENDING (Book_id, Branch_id, Card_No, Date_Out, Due_Date)
      6. LIBRARY_BRANCH (Branch_id, Branch_Name, Address)
3. Insert needed number of values (tuples) into the tables
4. Perform the various queries given below:
   1. Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each branch, etc.
   2. Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun2017
   3. Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation.
   4. Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.
   5. Create a view of all books and its number of copies that are currently available in the Library.

QUERIES - SYNTAX:

1. **Database Creation**

CREATE DATABASE LIBRARYDATABASE;

USE LIBRARYDATABASE;

2. **Table Creation**

CREATE TABLE PUBLISHER (NAME VARCHAR (20) PRIMARY KEY, PHONE BIGINT, ADDRESS VARCHAR (20));

CREATE TABLE BOOK (BOOK_ID INTEGER PRIMARY KEY, TITLE VARCHAR (20), PUBLISHER_NAME VARCHAR(20), PUB_YEAR VARCHAR (20), FOREIGN KEY (PUBLISHER_NAME) REFERENCES PUBLISHER (NAME) ON DELETE CASCADE);

CREATE TABLE BOOK_AUTHORS (BOOK_ID INTEGER, AUTHOR_NAME VARCHAR (20),FOREIGN KEY(BOOK_ID) REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, PRIMARY KEY (BOOK_ID, AUTHOR_NAME));

CREATE TABLE LIBRARY_BRANCH (BRANCH_ID INTEGER PRIMARY KEY, BRANCH_NAME VARCHAR (50), ADDRESS VARCHAR (50));
CREATE TABLE BOOK_COPIES (NO_OF_COPIES INTEGER, BOOK_ID INTEGER, BRANCH_ID INTEGER, FOREIGN KEY (BOOK_ID) REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, FOREIGN KEY(BRANCH_ID) REFERENCES LIBRARY_BRANCH (BRANCH_ID) ON DELETE CASCADE, PRIMARY KEY (BOOK_ID, BRANCH_ID));

CREATE TABLE CARD (CARD_NO INTEGER PRIMARY KEY);

CREATE TABLE BOOK_LENDING (DATE_OUT DATE, DUE_DATE DATE, BOOK_ID INTEGER, BRANCH_ID INTEGER, CARD_NO INTEGER, FOREIGN KEY (BOOK_ID) REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, FOREIGN KEY (BRANCH_ID) REFERENCES LIBRARY_BRANCH (BRANCH_ID) ON DELETE CASCADE, FOREIGN KEY (CARD_NO) REFERENCES CARD (CARD_NO) ON DELETE CASCADE, PRIMARY KEY (BOOK_ID, BRANCH_ID, CARD_NO));

3. Insertion of Values to Tables

INSERT INTO PUBLISHER VALUES (MCGRAW-HILL‘, 9989076587,’BANGALORE‘);

INSERT INTO PUBLISHER VALUES (PEARSON‘, 9889076565,’NEWDELHI‘);

INSERT INTO PUBLISHER VALUES (RANDOM HOUSE‘, 7455679345,‘HYDERABAD‘);

INSERT INTO PUBLISHER VALUES (HACHETTE LIVRE‘, 8970862340,‘CHENNAI‘);

INSERT INTO PUBLISHER VALUES (GRUPOPLANETA‘,7756120238,’BANGALORE‘);

INSERT INTO BOOK VALUES (1,‘DBMS‘ ,‘JAN-2017‘, ‘MCGRAW-HILL‘);

INSERT INTO BOOK VALUES (2,‘ADBMS‘ ,‘JUN-2016‘, ‘MCGRAW-HILL‘);

INSERT INTO BOOK VALUES (3,‘CN‘ ,‘SEP-2016‘, ‘PEARSON‘);

INSERT INTO BOOK VALUES (4,‘CG‘ ,‘SEP-2015‘, ‘GRUPO PLANETA‘);

INSERT INTO BOOK VALUES (5,‘OS‘ ,‘MAY-2016‘, ‘PEARSON‘);

INSERT INTO BOOK_AUTHORS VALUES (‘NAVATHE‘, 1);

INSERT INTO BOOK_AUTHORS VALUES (‘NAVATHE‘, 2);

INSERT INTO BOOK_AUTHORS VALUES (‘TANENBAUM‘, 3);

INSERT INTO BOOK_AUTHORS VALUES (‘EDWARD ANGEL‘, 4);

INSERT INTO BOOK_AUTHORS VALUES (‘GALVIN‘, 5);

INSERT INTO LIBRARY_BRANCH VALUES (10,‘RR NAGAR‘,‘BANGALORE‘);

INSERT INTO LIBRARY_BRANCH VALUES (11,‘RNSIT‘,‘BANGALORE‘);

INSERT INTO LIBRARY_BRANCH VALUES (12,‘RAJAJI NAGAR‘, ‘BANGALORE‘);

INSERT INTO LIBRARY_BRANCH VALUES (13,‘NITTE‘,‘MANGALORE‘);

INSERT INTO LIBRARY_BRANCH VALUES (14,‘MANIPAL‘,‘UDUPI‘);

INSERT INTO BOOK_COPIES VALUES (10, 1, 10);

INSERT INTO BOOK_COPIES VALUES (5, 1,11);

INSERT INTO BOOK_COPIES VALUES (2, 2,12);

INSERT INTO BOOK_COPIES VALUES (5, 2,13);

INSERT INTO BOOK_COPIES VALUES (7, 3,14);

INSERT INTO BOOK_COPIES VALUES (1, 5,10);

INSERT INTO BOOK_COPIES VALUES (3, 4,11);

INSERT INTO CARD VALUES (100);

INSERT INTO CARD VALUES (101);

INSERT INTO CARD VALUES (102);

INSERT INTO CARD VALUES (103);

INSERT INTO CARD VALUES (104);

INSERT INTO BOOK_LENDING VALUES (‘17-JAN-07‘,‘17-JUN-01‘, 1, 10, 101);

INSERT INTO BOOK_LENDING VALUES (‘17-JAN-11‘,‘17-MAR-11‘, 3, 14, 101);

INSERT INTO BOOK_LENDING VALUES (‘17-FEB-21‘,‘17-APR-21‘, 2, 13, 101);

INSERT INTO BOOK_LENDING VALUES (‘17-MAR-15‘,‘17-JUL-15‘, 4, 11, 101);

INSERT INTO BOOK_LENDING VALUES (_17-APR-12‘,‘17-MAY-12‘, 1, 11, 104);

4. <u>Basic Queries</u>

1. Query to Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each branch, etc.

SELECT B.BOOK_ID, B.TITLE, B.PUBLISHER_NAME, A.AUTHOR_NAME, C.NO_OF_COPIES, L.BRANCH_ID FROM BOOK B, BOOK_AUTHORS A, BOOK_COPIES C, LIBRARY_BRANCHL WHEREB.BOOK_ID=A.BOOK_ID AND B.BOOK_ID=C.BOOK_ID AND L.BRANCH_ID=C.BRANCH_ID;

2. Query to Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun2017.

SELECT CARD_NO FROM BOOK_LENDING WHERE DATE_OUT BETWEEN ‘01-JAN-2017‘ AND ‘01-JUL-2017‘ GROUP BY CARD_NO HAVING COUNT (*)>3;

3. Query to Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation.

DELETE FROM BOOK WHERE BOOK_ID=3;

4. Query to Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.

CREATE VIEW V_PUBLICATION AS SELECT PUB_YEAR FROM BOOK;

5. Query to Create a view of all books and its number of copies that are currently available in the Library.

CREATE VIEW V_BOOKS AS SELECT B.BOOK_ID, B.TITLE, C.NO_OF_COPIES FROM BOOK B, BOOK_COPIES C, LIBRARY_BRANCH L WHERE B.BOOK_ID=C.BOOK_ID AND C.BRANCH_ID=L.BRANCH_ID;

RESULT :

The queries to create database, create table and query table have been executed successfully.

EX No:	MySQL Replication – Distributed Databases
Date :	

AIM:
To implement Replication in distributed database using MYSQL.

THEORY :
MYSQL - Replication
MySQL supports replication capabilities that allow the databases on one server to be made available on another server. Replication is used for many purposes. For example, by replicating your databases, you have multiple copies available in case a server crashes or goes offline. Clients can use a different server if the one that they normally use becomes unavailable. Replication also can be used to distribute client load. Rather than having a single server to which all clients connect, you can set up multiple servers that each handle a fraction of the client load.

MySQL replication uses a master/slave architecture:
- The server that manages the original databases is the master.
- Any server that manages a copy of the original databases is a slave.
- A given master server can have many slaves, but a slave can have only a single master. (If done with care, it is possible to set up two-way or circular replication, but this study guide does not describe how.)

A replication slave is set up initially by transferring an exact copy of the to-be-replicated databases from the master server to the slave server. Thereafter, each replicated database is kept synchronized to the original database. When the master server makes modifications to its databases, it sends those changes to each slave server, which makes the changes to its copy of the replicated databases.

PROCEDURE:
Setting Up Replication
To set up replication, each slave requires the following:
- A backup copy of the master's databases. This is the replication "baseline" that sets the slave to a known initial state of the master.
- The filename and position within the master's binary log that corresponds to the time of the backup. The values are called the "replication coordinates." They are needed so that the slave can tell the master that it wants all updates made from that point on.
- An account on the master server that the slave can use for connecting to the master and requesting updates. The account must have the global REPLICATION SLAVE privilege. For example, you can set up an account for a slave by issuing these statements on the master server, where slave_user and slave_pass are the username and password for the account, and slave_host is the host from which the slave server will connect:

```
mysql> CREATE USER 'slave_user'@'slave_host' IDENTIFIED BY 'slave_pass';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'slave_user'@'slave_host';
```

Also, you must assign a unique ID value to each server that will participate in your replication setup. ID values are positive integers in the range from 1 to 2^{32}

1. The easiest way to assign these ID values is by placing a server-id option in each server's option file:
[mysqld] server-id=id_value

It's common, though not required, to use an ID of 1 for the master server and values greater than 1 for the slaves. The following procedure describes the general process for setting up replication.

1. Ensure that binary logging is enabled on the master server. If it is not, stop the server, enable logging, and restart the server.
2. On the master server, make a backup of all databases to be replicated. One way to do this is by using mysqldump:

   shell> mysqldump --all-databases --master-data=2 > dump_file

   Assuming that binary logging is enabled, the --master-data=2 option causes the dump file to include a comment containing a CHANGE MASTER statement that indicates the replication coordinates as of the time of the backup. These coordinates can be used later when you tell the slave where to begin replicating in the master's binary log.
3. Copy the dump file to the replication slave host and load it into the MySQL server on that machine:

   shell> mysql < dump_file
4. Tell the slave what master to connect to and the position in the master's binary log at which to begin replicating. To do this, connect to the slave server and issue a CHANGE MASTER statement:

   mysql> CHANGE MASTER TO -> MASTER_HOST = 'master_host_name', -> MASTER_USER = 'slave_user', -> MASTER_PASSWORD = 'slave_pass', -> MASTER_LOG_FILE = 'master_log_file', -> MASTER_LOG_POS = master_log_pos;

   The hostname is the host where the master server is running. The username and password are those for the slave account that you set up on the master. The log file and position are the replication coordinates in the master's binary log. (You can get these from the CHANGE MASTER statement near the beginning of the dump file.)

After you perform the preceding procedure, issue a START SLAVE statement. The slave should connect to the master and begin replicating updates that the master sends to it. The slave also creates a master.info file in its data directory and records the values from the CHANGE MASTER statement in the file. As the slave reads updates from the master, it changes the replication coordinates in the master.info file accordingly. Also, when the slave restarts in the future, it looks in this file to determine which master to use.

By default, the master server logs updates for all databases, and the slave server replicates all updates that it receives from the master. For more fine-grained control, it's possible to tell a master which databases to log updates for, and to tell a slave which of those updates that it receives from the master to apply. You can either name databases to be replicated (in which case those not named are ignored), or you can name databases to ignore (in which case those not named are replicated). The master host options are --binlog-do-db and --binlog-ignore-db. The slave host options are --replicate-do-db and --replicate-ignore-db.

The following example illustrates how this works, using the options that enable replication for specific databases. Suppose that a master server has three databases named a, b, and c. You can elect to replicate only databases a and b when you start the master server by placing these options in an option file read by that server:

[mysqld] binlog-do-db = a binlog-do-db = b

With those options, the master server will log updates only for the named databases to the binary log. Thus, any slave server that connects to the master will receive information only for databases a and b.

Enabling binary logging only for certain databases has an unfortunate side effect: Data recovery operations require both your backup files and your binary logs, so for any database not logged in the binary log, full recovery cannot be performed. For this reason, you might prefer to have the master log changes for all databases to the binary log, and instead filter updates on the slave side.
A slave that takes no filtering action will replicate all events that it receives. If a slave should replicate events only for certain databases, such as databases a and c, you can start it with these lines in an option file:

[mysqld] replicate-do-db = a replicate-do-db = c

RESULT:
The MYSQL replication was executed successfully.

EX No:	Spatial data storage and retrieval in MySQL
Date :	

AIM:

To create a spatial data storage and retrieve data in mysql.

PROCEDURE:

Step 1: Start the MYSQL server.
Step 2: Create a database and set that database.
Step 3: Create table with spatial column and insert the data.
Step 4: Use select statement to retrieve and view the content.

QUERIES:

Creating Spatial Columns:
Use the CREATE TABLE statement to create a table with a spatial column:
CREATE TABLE geom (g GEOMETRY);

Use the ALTER TABLE statement to add or drop a spatial column to or from an existing table
ALTER TABLE geom ADD pt POINT;
ALTER TABLE geom DROP pt;

Populating Spatial Columns
After you have created spatial columns, you can populate them with spatial data. Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format.
The following examples demonstrate how to insert geometry values into a table by converting WKT values to internal geometry format:

Perform the conversion directly in the INSERT statement:

```
INSERT INTO geom VALUES (ST_GeomFromText('POINT(1 1)'));

SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (ST_GeomFromText(@g));
```

Perform the conversion prior to the INSERT:

```
SET @g = ST_GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

To insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (ST_GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (ST_GeomFromText(@g));

SET @g ='GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (ST_GeomFromText(@g));
```

To use ST_GeomFromText() function to create geometry values. We can also use type-specific functions:

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (ST_PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (ST_LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (ST_PolygonFromText(@g));

SET @g ='GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (ST_GeomCollFromText(@g));
```

Inserting a POINT(1 1) value with hex literal syntax:

```
INSERT INTO geom VALUES
(ST_GeomFromWKB(X'010100000000000000000F03F000000000000F03F'));
```

An ODBC application can send a WKB representation, binding it to a placeholder using an argument of BLOB type:

```
INSERT INTO geom VALUES (ST_GeomFromWKB(?))
```

Fetching Spatial Data:

Geometry values stored in a table can be fetched in internal format. You can also convert them to WKT or WKB format. Fetching spatial data in internal format:Fetching geometry values using internal format can be useful in table-to-table transfers:

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

Fetching spatial data in WKT format:The ST_AsText() function converts a geometry from internal format to a WKT string.

```
SELECT ST_AsText(g) FROM geom;
```

Fetching spatial data in WKB format:The ST_AsBinary() function converts a geometry from internal format to a BLOB containing the WKB value.

```
SELECT ST_AsBinary(g) FROM geom;
```

RESULT:

The spatial data storage creation and retrieve of data in mysql has been executed successfully.

EX No: Date :	Temporal data storage and retrieval in MySQL

AIM:
To create a Temporal data storage and retrieval in MySQL.

PROCEDURE:

Step 1: Start the MYSQL server.
Step 2: Create a database and set that database.
Step 3: Create table with temporal column and insert the data.
Step 4: Use select statement to retrieve and view the content.

THEORY :
TEMPORAL DATATYPE
MySQL provides data types for storing different kinds of temporal information. In the following descriptions, the terms YYYY, MM, DD, hh, mm, and ss stand for a year, month, day of month, hour, minute, and second value, respectively.
The following table summarizes the storage requirements and ranges for the date and time data types.

Type	Storage Required	Range
DATE	3 bytes	'1000-01-01' to '9999-12-31'
TIME	3 bytes	'-838:59:59' to '838:59:59'
DATETIME	8 bytes	'1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP	4 bytes	'1970-01-01 00:00:00' to mid-year 2037
YEAR	1 byte	1901 to 2155 (for YEAR(4)), 1970 to 2069 (for YEAR(2))

QUERIES:

To Create table with temporal data
mysql> CREATE TABLE ts_test1 (->ts1 TIMESTAMP, ->ts2 TIMESTAMP, ->fdata CHAR(30) ->);
Query OK, 0 rows affected (0.00 sec)

To describe the table schema
mysql> DESCRIBE ts_test1;
+-------+-----------+------+-----+--------------------+-------+ | Field | Type | Null | Key | Default | Extra | +-------+-----------+------+-----+--------------------+-------+ | ts1 | timestamp | YES | | CURRENT_TIMESTAMP | | | ts2 | timestamp | YES | | 0000-00-00 00:00:00 | | | data | char(30) | YES | | NULL | | +-------+-----------+------+-----+--------------------+-------+
3 rows in set (0.01 sec)

To insert
mysql> INSERT INTO ts_test1 (data) VALUES ('original_value');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM ts_test1;
+---------------------+---------------------+----------------+ | ts1 | ts2 | data | +---------------------+---------------------+----------------+ | 2005-01-04 14:45:51 | 0000-00-00 00:00:00 | original_value | +---------------------+---------------------+----------------+

1 row in set (0.00 sec)

To update

```
mysql> UPDATE ts_test1 SET data='updated_value';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

To retrieve

```
mysql> SELECT * FROM ts_test1;
+---------------------+---------------------+---------------+ | ts1                 | ts2                 | data          | +----
---------------+---------------------+---------------+ | 2005-01-04 14:46:17 | 0000-00-00 00:00:00 |
updated_value | +---------------------+---------------------+---------------+ 1 row in set (0.00 sec)

 mysql> CREATE TABLE ts_test2 (     ->   created_time TIMESTAMP DEFAULT
CURRENT_TIMESTAMP,     ->   data CHAR(30)     -> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO ts_test2 (data) VALUES ('original_value');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM ts_test2;
+---------------------+----------------+ | created_time        | data           | +---------------------+---------------
-+ | 2005-01-04 14:46:39 | original_value | +---------------------+----------------+
1 row in set (0.00 sec)

mysql> UPDATE ts_test2 SET data='updated_value';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM ts_test2;
+---------------------+---------------+ | created_time        | data          | +---------------------+---------------+
| 2005-01-04 14:46:39 | updated_value | +---------------------+---------------+
1 row in set (0.00 sec)

 mysql> CREATE TABLE ts_test3 (     ->   updated_time TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,     ->   data CHAR(30)     -> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO ts_test3 (data) VALUES ('original_value'); Query OK, 1 row affected (0.00
sec)

mysql> SELECT * FROM ts_test3;
+---------------------+----------------+ | updated_time        | data           | +---------------------+---------------
--+ | 0000-00-00 00:00:00 | original_value | +---------------------+----------------+
1 row in set (0.00 sec)

mysql> UPDATE ts_test3 SET data='updated_value';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM ts_test3;
+---------------------+---------------+ | updated_time        | data          | +---------------------+---------------
+ | 2005-01-04 14:47:10 | updated_value | +---------------------+---------------+
1 row in set (0.00 sec)
```

```
mysql> CREATE TABLE ts_test4 (    ->   created TIMESTAMP DEFAULT
CURRENT_TIMESTAMP,    ->   updated TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,    ->   data CHAR(30)    -> );
ERROR 1293 (HY000): Incorrect table definition; there can be only one TIMESTAMP column with
CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause

mysql> CREATE TABLE ts_test5 (    ->   created TIMESTAMP DEFAULT 0,    ->   updated
TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,    ->   data CHAR(30)    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO ts_test5 (created, data)    -> VALUES (NULL, 'original_value');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM ts_test5; +--------------------+--------------------+----------------+ | created
| updated            | data           | +--------------------+--------------------+----------------+ | 2005-01-04
14:47:39 | 0000-00-00 00:00:00 | original_value | +--------------------+--------------------+----------------
--+
1 row in set (0.00 sec)

mysql> UPDATE ts_test5 SET data='updated_value';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM ts_test5;
+--------------------+--------------------+----------------+ | created            | updated            | data           |
+--------------------+--------------------+----------------+ | 2005-01-04 14:47:39 | 2005-01-04 14:47:52 |
updated_value | +--------------------+--------------------+----------------+ 1 row in set (0.00 sec)

mysql> CREATE TABLE ts_null (ts TIMESTAMP NULL);
Query OK, 0 rows affected (0.04 sec)

mysql> DESCRIBE ts_null;
+-------+-----------+------+-----+---------+-------+ | Field | Type      | Null | Key | Default | Extra | +------
-+-----------+------+-----+---------+-------+ | ts    | timestamp | YES  |     | NULL    |       | +-------+-------
----+------+-----+---------+-------+ 1 row in set (0.10 sec)

mysql> SELECT @@global.time_zone, @@session.time_zone;
+--------------------+---------------------+ | @@global.time_zone | @@session.time_zone | +-------------
-------+---------------------+ | SYSTEM             | SYSTEM              | +--------------------+------------------
---+
1 row in set (0.00 sec)

mysql> SET time_zone = '+00:00';
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @@session.time_zone; +---------------------+ | @@session.time_zone | +-------------
--------+ | +00:00              | +---------------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE ts_test (ts TIMESTAMP);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO ts_test (ts) VALUES (NULL);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM ts_test;
+---------------------+ | ts                  | +---------------------+ | 2005-01-04 20:50:18 | +---------------------
+ 1 row in set (0.00 sec)

mysql> SET time_zone = '+02:00';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM ts_test;
+---------------------+ | ts                  | +---------------------+ | 2005-01-04 22:50:18 | +---------------------
+
1 row in set (0.00 sec)

mysql> SET time_zone = '-05:00';
Query OK, 0 rows affected (0.00 sec)
mysql>
SELECT * FROM ts_test; +---------------------+ | ts                  | +---------------------+ | 2005-01-04
15:50:18 | +---------------------+ 1 row in set (0.00 sec)

mysql> SELECT CONVERT_TZ('2005-01-27 13:30:00', '+01:00', '+03:00');
+-----------------------------------------------------+ | CONVERT_TZ('2005-01-27 13:30:00', '+01:00',
'+03:00') | +-----------------------------------------------------+ | 2005-01-27 15:30:00
| +-----------------------------------------------------+
1 row in set (0.00 sec)
```

RESULT:
The Temporal data storage and retrieval in MySQL is executed successfully.

EX No:	Object storage and retrieval
Date :	

AIM:
To create and execute Object data storage and retrieval.

PROCEDURE:

Step 1: Start the Oracle server.
Step 2: Connect to the server through the client.
Step 3: Create a database and set that database.
Step 4: Create table and insert the data.
Step 5: Use select statement to retrieve and view the content.

QUERY:

To create an object type:

```
CREATE TYPE StockItem_objtyp;
CREATE TYPE LineItem_objtyp;
CREATE TYPE PurchaseOrder_objtyp;
CREATE TYPE PhoneList_vartyp AS VARRAY(10) OF VARCHAR2(20);

CREATE TYPE Address_objtyp AS OBJECT (
  Street        VARCHAR2(200),
  City          VARCHAR2(200),
  State         CHAR(2),
  Zip           VARCHAR2(20)
  )

CREATE TYPE Customer_objtyp AS OBJECT (
  CustNo          NUMBER,
  CustName        VARCHAR2(200),
  Address_obj     Address_objtyp,
  PhoneList_var    PhoneList_vartyp,

  ORDER MEMBER FUNCTION
    compareCustOrders(x IN Customer_objtyp) RETURN INTEGER
  )

CREATE TYPE LineItem_objtyp AS OBJECT (
  LineItemNo   NUMBER,
  Stock_ref    REF StockItem_objtyp,
  Quantity     NUMBER,
  Discount     NUMBER
  )
```

To create a table

```
CREATE TABLE Customer_objtab OF Customer_objtyp (CustNo PRIMARY KEY)
   OBJECT ID PRIMARY KEY ;
```

```
CREATE TABLE Stock_objtab OF StockItem_objtyp (StockNo PRIMARY KEY)   OBJECT ID
PRIMARY KEY ;

CREATE TABLE PurchaseOrder_objtab OF PurchaseOrder_objtyp (  /* Line 1 */
  PRIMARY KEY (PONo),                                  /* Line 2 */
  FOREIGN KEY (Cust_ref) REFERENCES Customer_objtab)       /* Line 3 */
  OBJECT ID PRIMARY KEY                                /* Line 4 */
  NESTED TABLE LineItemList_ntab STORE AS PoLine_ntab (     /* Line 5 */
   (PRIMARY KEY(NESTED_TABLE_ID, LineItemNo))           /* Line 6 */
   ORGANIZATION INDEX COMPRESS)                        /* Line 7 */
  RETURN AS LOCATOR                                    /* Line 8 */
```

To alter table

```
ALTER TABLE PoLine_ntab
  ADD (SCOPE FOR (Stock_ref) IS stock_objtab) ;

CREATE OR REPLACE TYPE BODY PurchaseOrder_objtyp AS
  MAP MEMBER FUNCTION getPONo RETURN NUMBER is
    BEGIN
      RETURN PONo;
    END;
  MEMBER FUNCTION sumLineItems RETURN NUMBER IS
    i         INTEGER;
    StockVal   StockItem_objtyp;
    Total      NUMBER := 0;
  BEGIN
    IF (UTL_COLL.IS_LOCATOR(LineItemList_ntab)) -- check for locator
      THEN
        SELECT SUM(L.Quantity * L.Stock_ref.Price) INTO Total
        FROM   TABLE(CAST(LineItemList_ntab AS LineItemList_ntabtyp)) L;
    ELSE
      FOR i in 1..SELF.LineItemList_ntab.COUNT LOOP
        UTL_REF.SELECT_OBJECT(LineItemList_ntab(i).Stock_ref,StockVal);
        Total := Total + SELF.LineItemList_ntab(i).Quantity *
                                          StockVal.Price;
      END LOOP;
    END IF;
  RETURN Total;
  END;
END;

ALTER TABLE PoLine_ntab
  ADD (SCOPE FOR (Stock_ref) IS stock_objtab);
```

To insert

```
INSERT INTO Stock_objtab VALUES(1004, 6750.00, 2) ;
INSERT INTO Stock_objtab VALUES(1011, 4500.23, 2) ;
INSERT INTO Stock_objtab VALUES(1534, 2234.00, 2) ;
INSERT INTO Stock_objtab VALUES(1535, 3456.23, 2) ;

INSERT INTO Customer_objtab
 VALUES (
   1, 'Jean Nance',
   Address_objtyp('2 Avocet Drive', 'Redwood Shores', 'CA', '95054'),
```

```
   PhoneList_vartyp('415-555-1212')
   ) ;

INSERT INTO Customer_objtab
  VALUES (
   2, 'John Nike',
   Address_objtyp('323 College Drive', 'Edison', 'NJ', '08820'),
   PhoneList_vartyp('609-555-1212','201-555-1212')
   ) ;

INSERT INTO PurchaseOrder_objtab
  SELECT  1001, REF(C),
        SYSDATE, '10-MAY-1999',
        LineItemList_ntabtyp(),
        NULL
   FROM   Customer_objtab C
   WHERE  C.CustNo = 1 ;

INSERT INTO PurchaseOrder_objtab
  SELECT  2001, REF(C),
        SYSDATE, '20-MAY-1997',
        LineItemList_ntabtyp(),
        Address_objtyp('55 Madison Ave','Madison','WI','53715')
   FROM   Customer_objtab C
   WHERE  C.CustNo = 2 ;

INSERT INTO TABLE (
  SELECT  P.LineItemList_ntab
   FROM   PurchaseOrder_objtab P
   WHERE  P.PONo = 1001
  )

SELECT  02, REF(S), 10, 10
   FROM   Stock_objtab S
   WHERE  S.StockNo = 1535 ;

INSERT INTO TABLE (
  SELECT  P.LineItemList_ntab
   FROM   PurchaseOrder_objtab P
   WHERE  P.PONo = 2001
  )
  SELECT  10, REF(S), 1, 0
   FROM   Stock_objtab S
   WHERE  S.StockNo = 1004 ;

INSERT INTO TABLE (
  SELECT  P.LineItemList_ntab
   FROM   PurchaseOrder_objtab P
   WHERE  P.PONo = 2001
  )
  VALUES(11, (SELECT REF(S)
    FROM  Stock_objtab S
    WHERE S.StockNo = 1011), 2, 1) ;
```

```
SELECT  p.PONo
 FROM   PurchaseOrder_objtab p
 ORDER BY VALUE(p) ;
```

Customer and Line Item Data for Purchase Order 1001

```
SELECT  DEREF(p.Cust_ref), p.ShipToAddr_obj, p.PONo,
       p.OrderDate, LineItemList_ntab
 FROM   PurchaseOrder_objtab p
 WHERE  p.PONo = 1001 ;
```

Total Value of Each Purchase Order

```
SELECT   p.PONo, p.sumLineItems()
 FROM    PurchaseOrder_objtab p ;
```

Purchase Order and Line Item Data Involving Stock Item 1004

```
SELECT   po.PONo, po.Cust_ref.CustNo,
        CURSOR (
          SELECT  *
           FROM   TABLE (po.LineItemList_ntab) L
           WHERE  L.Stock_ref.StockNo = 1004
          )
 FROM    PurchaseOrder_objtab po ;

SELECT   po.PONo, po.Cust_ref.CustNo, L.*
 FROM    PurchaseOrder_objtab po, TABLE (po.LineItemList_ntab) L
 WHERE   L.Stock_ref.StockNo = 1004 ;

SELECT   po.PONo, po.Cust_ref.CustNo, L.*
 FROM    PurchaseOrder_objtab po, TABLE (po.LineItemList_ntab) (+) L
 WHERE   L.Stock_ref.StockNo = 1004 ;

SELECT   AVG(L.DISCOUNT)
 FROM    PurchaseOrder_objtab po, TABLE (po.LineItemList_ntab) L ;
```

To delete

```
DELETE
 FROM   PurchaseOrder_objtab
 WHERE  PONo = 1001 ;
```

RESULT:

The creation and execution of Object storage and retrieval was executed successfully.

EX No:	XML Databases, XML table creation, XQuery FLWOR expression
Date :	

AIM:
To create and execute XML Databases , XML table creation, XQuery FLWOR expression.

PROCEDURE:

Step 1: Start the Oracle server.
Step 2: Connect to the server through the client.
Step 3: Create a database and set that database.
Step 4: Create table and insert the data.
Step 5: Use select statement (FLWOR) to retrieve and view the content.

QUERIES:

To create table
CREATE TABLE mytable1 (key_column VARCHAR2(10) PRIMARY KEY, xml_column XMLType);

Table created.

CREATE TABLE mytable2 OF XMLType;

Table created.

To insert values:
INSERT INTO mytable2 VALUES (XMLType(bfilename('XMLDIR', 'purchaseOrder.xml'),
nls_charset_id('AL32UTF8')));

To retrieve using XQuery
```
SELECT XMLQuery('for $i in /PurchaseOrder
           where $i/CostCenter eq "A10"
             and $i/User eq "SMCCAIN"
           return <A10po pono="{$i/Reference}"/>'
          PASSING OBJECT_VALUE
          RETURNING CONTENT)
  FROM purchaseorder;
```

XMLQUERY('FOR$IIN/PURCHASEORDERWHERE$I/COSTCENTEREQ"A10"AND$I/USEREQ"SMCCAIN"RET

<A10po pono="SMCCAIN-20021009123336151PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336341PDT"></A10po>
<A10po pono="SMCCAIN-20021009123337173PDT"></A10po>
<A10po pono="SMCCAIN-20021009123335681PDT"></A10po>
<A10po pono="SMCCAIN-20021009123335470PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336972PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336842PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336512PDT"></A10po>
<A10po pono="SMCCAIN-20021009123333894PDT"></A10po>

<A10po pono="SMCCAIN-20021009123337403PDT"></A10po>

XML File:

```
<PurchaseOrder>
 <Reference>SBELL-20021009123336O1PDT</Reference>
 <Actions>
  <Action>
   <User>SVOLLMAN</User>
  </Action>
 </Actions>
. . .
</PurchaseOrder>

<PurchaseOrder>
 <Reference>ABEL-20021127121040897PST</Reference>
 <Actions>
  <Action>
   <User>ZLOTKEY</User>
  </Action>
  <Action>
   <User>KING</User>
  </Action>
 </Actions>
. . .
</PurchaseOrder>
```

RESULT:

The creation and execution of XML Databases , XML table creation, XQuery FLWOR expression has been completed successfully.

MADHA ENGINEERING COLLEGE

(Affiliated to Anna University and Approved by AICTE, New Delhi) Madha Nagar, Kundrathur, Chennai-600069

DEPARTMENT OF Master of Computer Application

MC4212

Full Stack Web Development

Laboratory

R-2021

LAB MANUAL

2	2	1	2	2	2	2
3	2	1	2	2	2	2
4	2	1	2	2	2	2
5	2	1	2	2	2	2
Avg	2	1	2	2	2	2

MC4212 FULL STACK WEB DEVELOPMENT LABORATORY L T P C
0 0 4 2

COURSE OBJECTIVES:

- To implement the client side of the web application using javascript.
- To understand Javascript on the desktop using NodeJS.
- To develop a web application using NodeJS and Express.
- To implement a SPA using React.
- To develop a full stack single page application using React, NodeJS, and a Database (MongoDB or SQL).

LIST OF EXPERIMENTS:

1. Create a form and validate the contents of the form using JavaScript.
2. Get data using Fetch API from an open-source endpoint and display the contents in the form of a card.
3. Create a NodeJS server that serves static HTML and CSS files to the user without using Express.
4. Create a NodeJS server using Express that stores data from a form as a JSON file and displays it in another page. The redirect page should be prepared using Handlebars.
5. Create a NodeJS server using Express that creates, reads, updates and deletes students' details and stores them in MongoDB database. The information about the user should be obtained from a HTML form.
6. Create a NodeJS server that creates, reads, updates and deletes event details and stores them in a MySQL database. The information about the user should be obtained from a HTML form.
7. Create a counter using ReactJS
8. Create a Todo application using ReactJS. Store the data to a JSON file using a simple NodeJS server and retrieve the information from the same during page reloads.
9. Create a simple Sign up and Login mechanism and authenticate the user using cookies. The user information can be stored in either MongoDB or MySQL and the server should be built using NodeJS and Express Framework.
10. Create and deploy a virtual machine using a virtual box that can be accessed from the host computer using SSH.
11. Create a docker container that will deploy a NodeJS ping server using the NodeJS image.

TOTAL: 60 PERIODS

SOFTWARE REQUIREMENTS

1. NodeJS/Express JS, ReactJS, Docker, any IDE like NOTEPAD++/visual studio code/sublime text etc.,
2. MySQL, MongoDB

Ex.No.1
Date :

ADD CSS3 PROPERTIES OF STYLES TO INLIN, INTERNAL CSS3 PROPERTIES TO DOCUMENT

AIM: Add Styles to your Resume using CSS 3 Properties, Add External, Internal and Inline CSS styles to know the priority & Add CSS3 Animation to your profile.

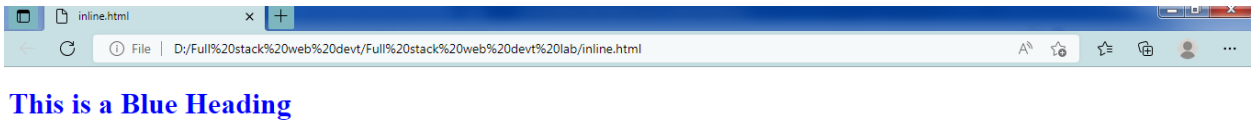
Inline CSS

- An inline CSS is used to apply a unique style to a single HTML element.
- An inline CSS uses the style attribute of an HTML element.
- To use inline styles you use the style attribute in the relevant tag. The style attribute can contain any CSS property.
- Inline CSS has the highest priority out of the three ways you can use CSS: external, internal, and inline.
- Specify the desired CSS properties with the style HTML attribute.

a) Inline Style Sheets

```
<!DOCTYPE html>
<html>
<body>

<h1 style="color:blue;">This is a Blue Heading</h1>

</body>
</html>
```

OUTPUT

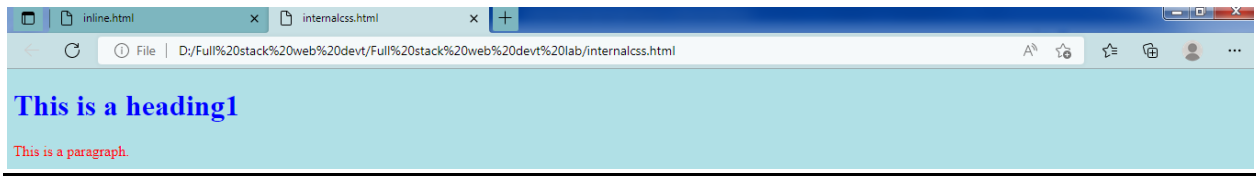


---

b) Internal CSS

An internal CSS is used to define a style for a single HTML page.
An internal CSS is defined in the <head> section of an HTML page, within a <style> element:

```
<!DOCTYPE html>
<html>
<head>
<style>
    body {background-color: powderblue;}
    h1   {color: blue;}
    p    {color: red;
          align:center;}
</style>
</head>

<body>

<h1>This is a heading1</h1>
<p>This is a paragraph.</p>

</body>
</html>
```



Result : The above program was executed successfully, hence the output verified.

Ex.No: 2
Date :

FORM VALIDATION USING JAVASCRIPT

AIM: To create Form validation using JavaScript.

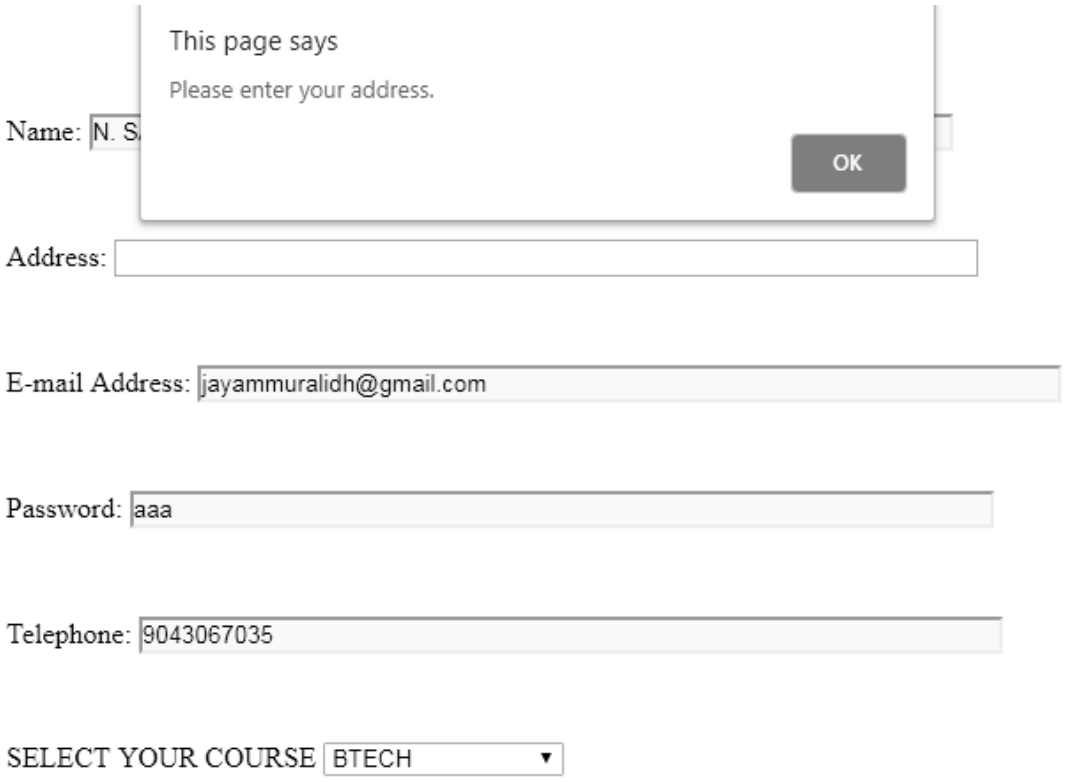
Program

```
<html>
<head>
<script>

function VALIDATEDETAIL()
{
   var name = document.forms["RegForm"]["Name"];
   var email = document.forms["RegForm"]["EMail"];
   var phone = document.forms["RegForm"]["Telephone"];
   var what =  document.forms["RegForm"]["Subject"];
   var password = document.forms["RegForm"]["Password"];
   var address = document.forms["RegForm"]["Address"];

   if (name.value == "")
   {
      window.alert("Please enter your name.");
      name.focus();
      return false;
   }

   if (address.value == "")
   {
      window.alert("Please enter your address.");
      name.focus();
      return false;
   }

   if (email.value == "")
   {
      window.alert("Please enter a valid e-mail address.");
      email.focus();
      return false;
   }
```

```
   if (email.value.indexOf("@", 0) < 0)
   {
      window.alert("Please enter a valid e-mail address.");
      email.focus();
      return false;
   }

   if (email.value.indexOf(".", 0) < 0)
   {
      window.alert("Please enter a valid e-mail address.");
      email.focus();
      return false;
   }

   if (phone.value == "")
   {
      window.alert("Please enter your telephone number.");
      phone.focus();
      return false;
   }

   if (password.value == "")
   {
      window.alert("Please enter your password");
      password.focus();
      return flase;
   }

   if (what.selectedIndex < 1)
   {
      alert("Please enter your course.");
      what.focus();
      return false;
   }

   return true;
}</script>
  <style>
VALIDATEDETAIL {
   font-weight: bold ;
   float: left;
   width: 100px;
   text-align: left;
   margin-right: 10px;
   font-size:14px;
}
```

```
div {
   box-sizing: border-box;
   width: 100%;
   border: 100px solid black;
   float: left;
   align-content: center;
   align-items: center;
}

form {
   margin: 0 auto;
   width: 600px;
}</style></head>

<body>
<h1 style="text-align: center"> REGISTRATION FORM </h1>
<form name="RegForm" action="submit.php" onsubmit="return VALIDATEDETAIL()"
method="post">

   <p>Name: <input type="text" size=65 name="Name"> </p><br>
   <p> Address: <input type="text" size=65 name="Address">  </p><br>
   <p>E-mail Address: <input type="text" size=65 name="EMail">  </p><br>
    <p>Password: <input type="text" size=65 name="Password"> </p><br>
   <p>Telephone: <input type="text" size=65 name="Telephone"> </p><br>

   <p>SELECT YOUR COURSE
      <select type="text" value="" name="Subject">
         <option>BTECH</option>
         <option>BBA</option>
         <option>BCA</option>
         <option>B.COM</option>
         <option>VALIDATEDETAIL</option>
      </select></p><br><br>
   <p>Comments:  <textarea cols="55" name="Comment">  </textarea></p>
   <p><input type="submit" value="send" name="Submit">
      <input type="reset" value="Reset" name="Reset">
   </p>
</form>
</body>
</html>
```

OUTPUT



Result : The above program was executed successfully, hence the output verified.

Ex. No:3
Date :

Get data using Fetch API from an open-source endpoint and display the contents in the form of a card.

AIM: To fetch data and display the contents in the form of a card.

script.js

```
// api url
const api_url =
     "https://employeedetails.sriventech.ac.in/my/api/path";

// Defining async function
async  function   getapi(url) {

    // Storing response
    const response = await fetch(url);

    var data = await response.json();
    console.log(data);
    if (response) {
        hideloader();
    }
    show(data);
}
// Calling that async function
getapi(api_url);

// Function to hide the loader
function hideloader() {
    document.getElementById('loading').style.display = 'none';
}
// Function to define innerHTML for HTML table
function show(data) {
    let tab =
        `<tr>
          <th>Name</th>
          <th>Office</th>
          <th>Position</th>
          <th>Salary</th>
        </tr>`;
      }
```

```
    // Loop to access all rows
    for (let r of data.list) {
        tab += `<tr>
    <td>${r.name} </td>
    <td>${r.office}</td>
    <td>${r.position}</td>
    <td>${r.salary}</td>
</tr>`;
    }
    // Setting innerHTML as tab variable
    document.getElementById("employees").innerHTML = tab;
}
```

employee.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <script src="script.js"></script>

        <meta charset="UTF-8" />
        <meta name="viewport"
              content="width=device-width, initial-scale=1.0" />
        <title>Document</title>
    </head>
    <body>
        <!-- Here a loader is created which
             loads till response comes -->
        <div class="d-flex justify-content-center">
            <div class="spinner-border"
                 role="status" id="loading">
                <span class="sr-only">Loading...</span>
            </div>
        </div>
        <h1>Registered Employees</h1>
        <!-- table for showing data -->
        <table id="employees"></table>
    </body>
</html>
```

Output

Result : The above program was executed successfully, hence the output verified.

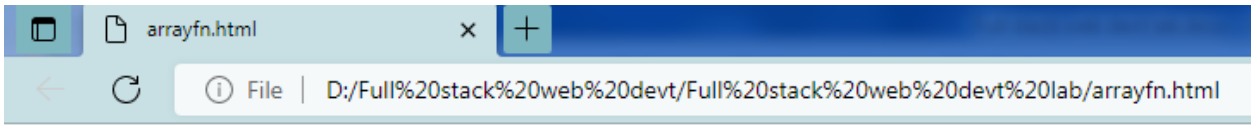
Ex. No:4
Date :

JAVASCRIPT ARRAY FUNCTIONS

AIM: To use array function in Javascript.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>
<p  id="demo"></p>
<p  id="demo2"></p>
<p  id="demo3"></p>
<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
<!-- Converting Arrays to Strings -->
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo2").innerHTML = fruits.toString();
</script>
<!-- Joins all array elements -->
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo3").innerHTML = fruits.join(" * ");
</script>

</body>
</html>
```

OUTPUT



JavaScript Arrays

Saab,Volvo,BMW

Banana,Orange,Apple,Mango

Banana * Orange * Apple * Mango

Result : The above program was executed successfully, hence the output verified.

Ex. No:5
Date :

Create a NodeJS server using Express that stores data from a form as a JSON file

Aim : Create a NodeJS server using Express that stores data from a form as a JSON file and displays it in another page. The redirect page should be prepared using Handlebars.

Steps :

1.Create a folder called fetch API in VS Code and create a new file JavaScript file
Now, write the Code get free API and store it in Const api_url get the used id from Api using prompt method.

2. Convert into parseInt. Using asyn function get the API and point the needed data from the API to Card. New show function get the Emp details in the table format and send it to html file.

3.In HTML file create a Card using css file and Show the data fetched from the API. To the table. Using "Id - demo" link the css file, Script file to the html and open it in Live Server.

Server.js

```
const express = require ('express');
const expbs = require('express-handlebars'); const app = express();
app.use(express.urlencoded());

app.engine('handlebars', expbs.engine ({defaultLayout: false,})); app.set('view engine',
'handlebars');

//routing
app.get('/', function(request, response, next){ response.render('index', { layout: false });
});
app.post('/', function(request, response, next){ response.send(request.body);
});

app.listen(2000);
```

```
=>(main.handlebar)
<html>
<head>
<title>
{{{ title }}}
</title>
</head>
<body>
{{{ body }}}
</body>
</html>
```

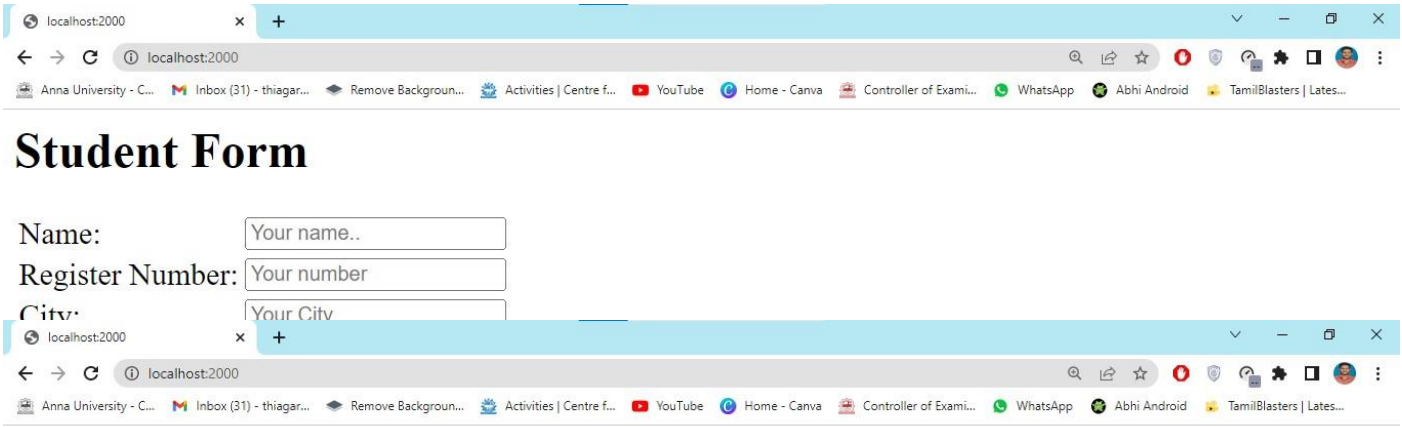
main.html

```
<html>
<head>
<title>
{{{ title }}}
</title>
</head>
<body>
{{{ body }}}
</body>
</html>
```

index.html

```
<h1>Student Form</h1>
<form action='/' method="post">
<Table style="font-size:20px;">
<tr>
<td><label for="name">Name:</label></td>
<td><input type="text" id="fname" name="name" placeholder="Your name.."></td>
</tr>
<tr>
<td><label for="reg">Register Number:</label></td>
<td><input type="text" id="lname" name="lastname" placeholder="Your number"></td>
</tr>
<tr>
<td><label for="city">City:</label></td>
<td><input id="subject" name="subject" placeholder="Your City" ></input></td>
</tr>
<tr>
<td><input type="submit" value="Submit"></td>
</tr>
</Table>
</form>
```

Output:



{"name":"Admin","lastname":"110821622000","subject":"Avadi"}

Result : The above program was executed successfully, hence the output verified.

Ex. No:6
Date :

Create a NodeJS server using Express that creates, insert, updates and deletes customers details and stores them in MongoDB database. The information about the user should be obtained from a HTML form.

create_database.js

```
var MongoClient = require('mongodb').MongoClient;
//Create a database named "mydb":
var url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  console.log("Database created!");
  db.close();
});
```

Output

```
C:\Users\My Name>node demo_create_mongo_db.js
Database created!
```

Creating a Collection

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  //Create a collection name "customers":
  dbo.createCollection("customers", function(err, res) {
    if (err) throw err;
    console.log("Collection created!");
    db.close();
  });
});
```

Output

```
C:\Users\My Name>node demo_mongodb_createcollection.js
Collection created!
```

Insert data into Collection

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = { name: "Company Inc", address: "Highway 37" };
  dbo.collection("customers").insertOne(myobj, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
    db.close();
  });
});
```

Output

```
C:\Users\My Name>node demo_mongodb_insert.js
1 document inserted
```

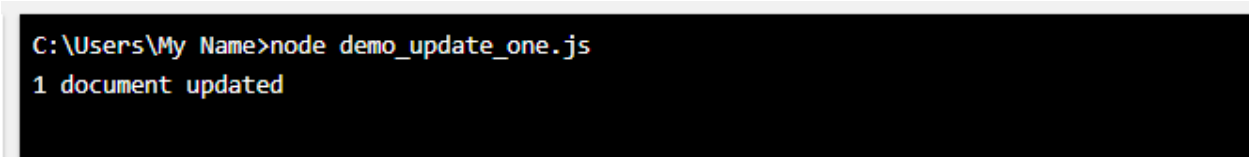
Update document

Update the document with the address "Valley 345" to name="Mickey" and address="Canyon 123":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myquery = { address: "Valley 345" };
  var newvalues = { $set: { name: "Michael", address: "Canyon 123" } };
```

```
  dbo.collection("customers").updateOne(myquery, newvalues, function(err, res) {
    if (err) throw err;
    console.log("1 document updated");
    db.close();
  });
});
```

Output



Result : The above program was executed successfully, hence the output verified.

Ex. No:7
Date :

Create a NodeJS server that creates, reads, updates and deletes event details and stores them in a MySQL database. The information about the user should be obtained from a HTML form.

Procedure:

1. **Create database in mysql**
2. **Write connection.php**
3. **Write php coding for reading data from database**

```
<?php
include_once('connection.php');
$query="select * from emp where age>25";
$result=mysql_query($query);
?>

<!DOCTYPE html>
<html>
   <head>
      <title> Fetch Data From Database </title>
   </head>
<body>

   <table align="center" border="1px" style="width:600px; line-height:40px;">
      <tr>
         <th colspan="4"><h2>Employee Record</h2></th>
      </tr>
      <t>
         <th>Empno</th>
         <th> Name </th>
         <th> Age </th>

      </t>
   <?php
      while($rows=mysql_fetch_assoc($result))
      {
   ?>
         <tr>
            <td><?php echo $rows['empno']; ?></td>
            <td><?php echo $rows['empname']; ?></td>
            <td><?php echo $rows['age']; ?></td>

         </tr>
   <?php
      }
   ?>
   </table>
   </body>
</html>
```

Output

```
+------------+---------------+----------------+----------------+
Empno | Empname          | Age |
+------------+---------------+----------------+----------------+
|          1 |  John Poul       | 20     |
|          2 |  Abdul S         | 25     |
|          3 |  Sanjay          | 24     |
+------------+---------------+----------------+----------------+
```

Result : The above program was executed successfully, hence the output verified.

Ex. No:8
Date :

Create a counter using ReactJS

Aim : To create a counter using ReactJS

```
<!DOCTYPE html>
<html lang="en">

<body style="text-align:center">
    <h1>Fullstack</h1>
    <p>COUNTS</p>
    <div id="counter">
        <!-- counts -->
    </div>

    <script>
        let counts=setInterval(updated);
        let upto=0;
        function updated(){
            var count= document.getElementById("counter");
            count.innerHTML=++upto;
            if(upto===1000)
            {
                clearInterval(counts);
            }
        }
    </script>
</body>
</html>
```

OUTPUT

Result : The above program was executed successfully, hence the output verified.

Ex. No:9
Date :

Create and Deploy a virtual machine using a virtual box that can be accessed from the hostcomputer using SSH

Aim : To Create and Deploy a virtual machine using a virtual box that can be accessed from the hostcomputer using SSH

Procedure:

Prepare your computer for virtualization. Install Hypervisor (virtualization tool).Import a virtual machine. Start the virtual machine. Using the virtual machine. Shutdown the virtualmachine

VIRTUALIZATION – the underlying technology that allows a virtual operating system tobe run as an application on your computer's operating system.

HYPERVISOR – the virtualization application (such as VirtualBox or VMware) running onyour host computer that allows it to run a guest virtual operating system.

HOST – the computer on which you are running the hypervisor application.

GUEST – a virtual operating system running within the hypervisor on your host computer.The virtual operating system term is synonymous with other terms such as Virtual \ Machine, VM and instance.

Program:

Step 1: Prepare your computer for Virtualization:

Enable Processor Virtualization: Ensure Virtualization is enabled on your computer.See the Virtualization Error (VT-d/VT-x or AMD-V) for troubleshooting support.

Review File Sync Services for tools like OneDrive, Nextcloud, DropBox Sync, iCloud, etc. If you are using a data synchronization service, make sure it DOES NOT (or at least not frequently)synchronize the folder in which your hypervisor imports and installs the Virtual Machines.

File sync services can cause a dramatic fall-off in performance for your entire system as these services try to synchronize these massive files that are getting updated constantly while youare using the Virtual Machines.

Sufficient Disk Space: Virtual Machines require a significant amount of Disk space (10 GBor more each is typical). Ensure you have sufficient space on your computer.

Admin Privileges: Installing a hypervisor on a host in most cases requires admin privileges.

Step 2: Install Hypervisor (Virtualization Tool):

Installing a hypervisor on your host is usually quite simple. In most cases, the install program will ask only a couple of questions, such as where to install the hypervisor software.

Step 3: Import a Virtual Machine:

The first step is to download the Virtual Machine for your course from our Course VirtualMachines page. This will download an .ova file. The .ova file is actually a compressed (zipped) tarball of a Virtual Machine exported from Virtual Box.

Once the Virtual Machine has been imported, it will normally show up in the guest listwithin your hypervisor tool.

Step 4: Start the Virtual Machine:

To start up a Virtual Machine guest in most hypervisors, you simply click on the desiredguest and click the Start button (often double-clicking the guest icon will work as well).

Step 5: Using the Virtual Machine:

Sharing files between the guest and host: To learn about different ways of sharing files,check out this guide.
Run a command with sudo (root) privileges: Open a terminal and type any command withsudo in front to run that command as root.

Example: sudo apt-get install vim – will install the vim text editor package on an Ubuntu LinuxVirtual Machine.

Find the IP address of your guest: Open a terminal and type ifconfig | more – The | more (pronounced “pipe more”) will “pipe” the output of the ifconfig command to the more command,which will show the results one page at a time, so it doesn’t scroll by before you see it all.

If you have a Host-Only Network IP address, you will see an IP of 192.168.56.101 (or somethingsimilar). Check the Trouble-Shooting section below for more information about the Host-Only Network.

Step 6: Shut down the Virtual Machine:

When you are done using a guest Virtual Machine, regardless of hypervisor, you need toshut it down properly. This can be done in three ways:

1. Press the shutdown button found on the desktop, taskbar, or task menu of the guestoperating system.
2. Open a terminal and type the command: sudo shutdown -h now
3. In the guest window, click Machine (menu) -> ACPI Shut down – This will simulate thepower button being pressed.

RESULT:

The above program is executed successfully. Hence output verified.