PDEA's College Of Engineering
Manjari, Pune

LAB MANUAL

Subject: Advanced Database Management System

Lab Practice I [314448]

Department of Information Technology

PROGRAM EDUCATIONAL OBJECTIVES (PEOs):

1. **Database Design Proficiency:** Demonstrate proficiency in designing complex databases, including the ability to create and optimize database schemas using concepts like Star and Snowflake schemas.

2. **MongoDB Expertise:** Master MongoDB, including the ability to create and manage databases, collections, and documents, and perform various operations like insertion, deletion, and updates.

3. **Querying Skills:** Execute a wide range of queries in MongoDB, showcasing skills in querying for specific values, using query criteria, handling conditional queries, working with different data types, and utilizing advanced query options like cursors.

4. **Aggregation and Map-Reduce:** Implement the MongoDB aggregation framework and map-reduce operations, demonstrating the ability to process and analyze data efficiently.

5. **Indexing Knowledge:** Create and manage various types of indexes in MongoDB, highlighting the advantages of indexing for query performance optimization.

6. **Application Development:** Develop a mini-project application that integrates a graphical user interface (GUI) with a MongoDB backend. Implement basic CRUD operations and incorporate data entry validations.

7. **Teamwork and Project Management:** Collaborate effectively in teams of 3 to 4 people to build the mini-project. Plan and manage project development, ensuring timely completion and adherence to requirements.

8. **Documentation and Reporting:** Prepare a comprehensive project report that includes project title, abstract, hardware and software requirements, source code, graphical user interface screenshots, and a conclusion summarizing the project's outcomes.

PROGRAM OUTCOMES (Pos)

1. **Database Creation and Management:** Create and manage databases using MongoDB, including batch insertion of documents and implementing validation rules for data integrity.

2. **Document Manipulation:** Perform document operations, such as inserting and saving documents, removing documents, and updating documents using various methods including document replacement, modifiers, and upserts.

3. **Query Proficiency:** Execute at least 10 queries in MongoDB, showcasing competence in finding specific values, utilizing query criteria with conditionals, performing OR queries, using $not and conditional semantics, handling type-specific queries (null, regular expression, array querying), and utilizing $where queries.

4. **Cursor Handling:** Demonstrate the ability to work with cursors, including limiting, skipping, sorting, and using advanced query options in MongoDB.

5. **Map-Reduce and Aggregation:** Implement Map-Reduce and the MongoDB aggregation

framework for data processing and analysis.

6. **Indexing Knowledge:** Create, manage, and explain different types of indexes in MongoDB, highlighting their advantages for query optimization.

7. **Schema Design:** Design a conceptual model using both Star and Snowflake schema techniques for a given database scenario.

8. **Application Development:** Develop a mini-project application that integrates a graphical user interface (GUI) with the MongoDB backend, implementing basic CRUD (Create, Read, Update, Delete) operations and ensuring proper data entry validations.

9. **Team Collaboration:** Collaborate effectively in teams of 3 to 4 members to develop the mini-project, demonstrating teamwork and project management skills.

10. **Documentation and Reporting:** Prepare a comprehensive project report including the project title, abstract, hardware and software requirements for both backend and frontend, source code, screenshots of the graphical user interface, and a conclusion summarizing the project's outcomes.

Program Specific Outcomes (PSO)

1**Database Creation and Document Manipulation (MongoDB):**

- PSO1: Create a MongoDB database with a suitable example.
- PSO2: Implement batch insertion of documents and define insert validation rules for data integrity.
- PSO3: Remove documents from the database.
- PSO4: Update documents using various techniques, including document replacement, modifiers, upserts, and returning updated documents.
-

2. **MongoDB Query Proficiency:**

    - PSO5: Execute a minimum of 10 queries on MongoDB, illustrating the following query types:
        - PSO5.1: Find and findOne queries for retrieving specific values.
        - PSO5.2: Query criteria, including query conditionals, OR queries, $not, and conditional semantics.
        - PSO5.3: Type-specific queries, such as null, regular expression, and querying arrays.
        - PSO5.4: Utilize $where queries for complex conditions.
        - PSO5.5: Work with cursors, applying operations like limit, skip, sort, and advanced query options.
        -

3. **Map-Reduce, Aggregation, and Indexing (MongoDB):**

    - PSO6: Implement Map-Reduce and demonstrate the use of the MongoDB aggregation framework for data processing and analysis.
    - PSO7: Create and drop different types of indexes in MongoDB and explain their advantages for query optimization.
    -

4. **Database Schema Design:**

- PSO8: Design a conceptual model using both Star and Snowflake schema techniques for a selected database case study.

5. **Mini Project Development:**

- PSO9: Form teams of 3 to 4 members for the mini project.
- PSO10: Develop a GUI-based application with proper data entry validations.
- PSO11: Establish a database connection with the front end.
- PSO12: Implement basic CRUD (Create, Read, Update, Delete) operations in the application.
- PSO13: Prepare and submit a comprehensive project report, including the project title, abstract, hardware and software requirements for both backend and frontend, source code, graphical user interface screenshots, and a conclusion summarizing the project's outcomes.

TABLE OF CONTENTS

Assignment: 1

AIM: Create a database with suitable example using MongoDB and implement

1. Inserting and saving document (batch insert, insert validation)
2. Removing document
3. Updating document (document replacement, using modifiers, up inserts, updating multiple documents, returning updated documents)
4. Execute at least 10 queries on any suitable MongoDB database that demonstrates following:
    - Find and find One (specific values)
    - Query criteria (Query conditionals, OR queries, $not, Conditional semantics)
    - Type-specific queries (Null, Regular expression, Querying arrays) where queries
    - Cursors (Limit, skip, sort, advanced query options)

PROBLEM STATEMENT /DEFINITION

Create a database with suitable example using MongoDB and implement

- Inserting and saving document (batch insert, insert validation)
- Removing document
- Updating document (document replacement, using modifiers, upserts, updating multiple documents, returning updated documents)

OBJECTIVE:

- To understand Mongodb basic commands
- To implement the concept of document oriented databases. To understand Mongodb retrieval commands

THEORY:

SQL VsMongoDB

SQL Concepts	MongoDB Concepts
Database	**Database**
Table	**Collection**

Row	**Document 0r BSON Document**
Column	**Field**
Index	**Index**
Table Join	**Embedded Documents & Linking**
Primary Key	**Primary Key**
Specify Any Unique Column Or Column Combination As Primary Key.	**In Mongodb, The Primary Key Is Automatically Set To The _Id Field.**
Aggregation (E.G. Group By)	**Aggregation Pipeline**

1. Create a collection in mongodb

db.createCollection(“Teacher_info")

2. Create a capped collection in mongodb

>db.createCollection("audit", {capped:true, size:20480})

{ "ok" : 1 }

3. Insert a document into collection

db.Teacher_info.insert({ Teacher_id: “Pic001", Teacher_Name: “Ravi",Dept_Name: “IT”, Sal:30000, status: "A" })

db.Teacher_info.insert({ Teacher_id: “Pic002", Teacher_Name: “Ravi",Dept_Name: “IT”, Sal:20000, status: "A" })

db.Teacher_info.insert({ Teacher_id: “Pic003", Teacher_Name: “Akshay",Dept_Name: “Comp”, Sal:25000, status: “N" })

4. Update a document into collectiondb. Teacher_info.update({ sal: { $gt: 25000 } }, { $set: { Dept_name: “ETC" } }, {multi: true })

db. Teacher_info.update({ status: "A" } , { $inc: { sal: 10000 } }, { multi: true })

5. Remove a document from collection

```
db.Teacher_info.remove({Teacher_id:
"pic001"}); db. Teacher_info.remove({})
```

6. Alter a field into a mongodb document

```
db.Teacher_info.update( { }, { $set: { join_date: new Date() } }, { multi:
true} )
```

7. To drop a particular collection

```
db.Teacher_info.drop()
```

Retrieval From Database:-

When we retrieve a document from mongodb collection it always add a _id field in the every document which conatin unique _id field.

ObjectId(<hexadecimal>)

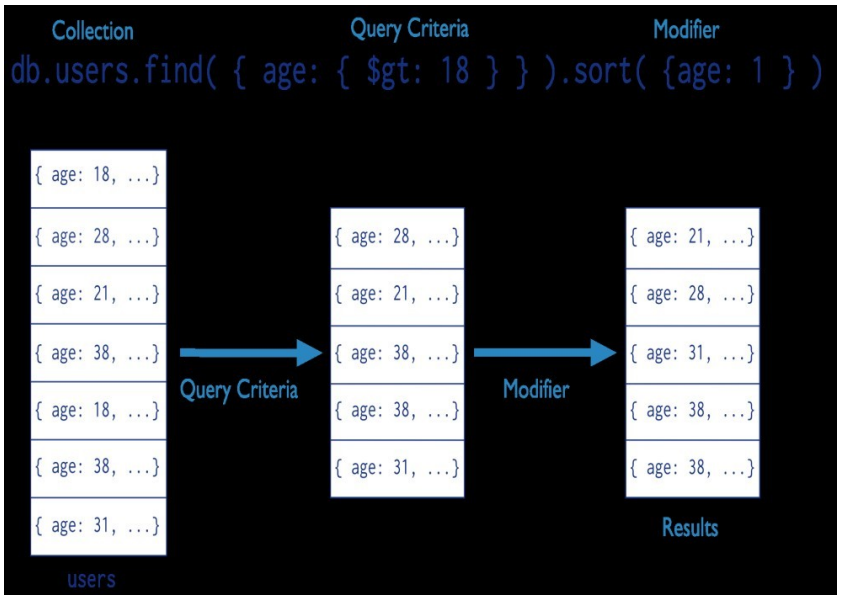
Returns a new ObjectId value. The 12-byte ObjectId value consists of:

4-byte value representing the seconds since the Unix epoch,

3-byte machine identifier,

2-byte process id, and

3-byte counter,starting with a random value.



1. **Retrieve a collection in mongodb using Find command**

db.Teacher.find()

```
{ "_id" : 101, "Name" : "Dev",
"Address" : [  {  "City" : "Pune",  "Pin" : 444043 } ],
"Department" : [ { "Dept_id" : 111, "Dept_name" : "IT" } ],
 "Salary" : 78000 }

{ "_id" : 135, "Name" : "Jennifer", "Address" : [  {  "City" :
"Mumbai", "Pin" : 444111 } ], "Department" : [ { "Dept_id" : 112,
"Dept_name" : "COMP" } ], "Salary" : 65000 }
{ "_id" : 126, "Name" : "Gaurav", "Address" : [  {  "City" : "Nashik",
"Pin" : 444198 } ], "Department" : [  {  "Dept_id" : 112,  "Dept_name"
: "COMP" } ], "Salary" : 90000 }
{ "_id" : 175, "Name" : "Shree", "Address" : [  {  "City" : "Nagpur",
"Pin" : 444158 } ], "Department" : [  {  "Dept_id" : 113,  "Dept_name"
: "ENTC" } ], "Salary" : 42000 }
{ "_id" : 587, "Name" : "Raman", "Address" : [  {  "City" : "Banglore",
"Pin" : 445754 } ], "Department" : [  {  "Dept_id" : 113,  "Dept_name"
: "ENTC" } ], "Salary" : 79000 }
{ "_id" : 674, "Name" : "Mandar", "Address" : [  {  "City" : "Jalgaon",
"Pin" : 465487 } ], "Department" : [  {  "Dept_id" : 111,  "Dept_name"
: "IT" } ], "Salary" : 88000 }
{ "_id" : 573, "Name" : "Manish", "Address" : [  {  "City" : "Washim",
"Pin" : 547353 } ], "Department" : [  {  "Dept_id" : 112,  "Dept_name"
: "COMP" } ], "Salary" : 65000 }
```

2. Retrieve a document from collection in mongodb using Find command using condition

>db.Teacher_info.find({sal: 25000})

3. Retrieve a document from collection in mongodb using Find command using or operator

>db.Teacher_info.find({ $or: [{ status: "A" } , { sal:50000 }] })

4. Retrieve a document from collection in mongodb using Find command using greater than , less than, greater than and equal to ,less than and equal to operator

>db. Teacher_info.find({ sal: **{ $gt:** 40000 } })

>db.media.find({ Released : **{$gt :** 2000} }, { **"Cast" : 0** })

{ "_id" : ObjectId("4c4369a3c603000000007ed3"), "Type" : "DVD", "Title" :"Toy Story 3", "Released" : 2010 }

>db.media.find ({ Released : **{$gte** : 1999 } }, { "Cast" : 0 })

{ "_id" : ObjectId("4c43694bc603000000007ed1"), "Type" : "DVD", "Title" :"Matrix, The", "Released" : 1999 }

{ "_id" : ObjectId("4c4369a3c603000000007ed3"), "Type" : "DVD", "Title" :"Toy Story 3", "Released" : 2010 }

>db.media.find ({ Released : **{$lt :** 1999 } }, { "Cast" : 0 })

{ "_id" : ObjectId("4c436969c603000000007ed2"), "Type" : "DVD", "Title" : "Blade Runner", "Released" : 1982 }

>db.media.find({Released : **{$lte:** 1999}}, { "Cast" : 0 })

{ "_id" : ObjectId("4c43694bc603000000007ed1"), "Type" : "DVD", "Title" :"Matrix, The", "Released" : 1999 }

{ "_id" : ObjectId("4c436969c603000000007ed2"), "Type" : "DVD", "Title" :"Blade Runner", "Released" : 1982 }

>db.media.find({Released : **{$gte: 1990, $lt : 2010**}}, { "Cast" : 0 })

{ "_id" : ObjectId("4c43694bc603000000007ed1"), "Type" : "DVD", "Title" :"Matrix, The", "Released" : 1999 }

Retrieval a value from document which contain array field

Exact Match on an Array

db.inventory.find({ tags: ['fruit', 'food', 'citrus'] })

Match an Array Element

db.inventory.find({ tags: 'fruit' })

Match a Specific Element of an Array

db.inventory.find({ 'tags.**0**' : 'fruit' })

6. **MongoDB provides a db.collection.findOne() method as a special case of find() that returns a single document.**

7. **Exclude One Field from a Result Set**

>db.records.Find({ "user_id": { $lt: 42} }, { history: 0})

8. Return Two fields and the _id Field

>db.records.find({ "user_id": { $lt: 42} }, { "name": 1, "email": 1})

9. Return Two Fields and Exclude _id

>db.records.find({ "user_id": { $lt: 42} }, { "_id": 0, "name": 1 , "email": 1 })

10. Retrieve a collection in mongodb using Find command and pretty appearance

>db.<collection>.find().pretty()

db.users.find(collection

{age:{$gt:18}}, query criteria

{name :1,address:1} projection

Retrieve a document in ascending or descending order using 1 for ascending and -1 for descendingfrom collection in mongodb

>db. Teacher_info.find({ status: "A" }).sort({sal: -1 })

>db.audit.find().sort({ $natural: -1 }).limit (10)

>db.Employee.find().sort({_id:-1})

```
{ "_id" : 106, "Name" : "RAJ", "Address" : [ { "City" : "NASIK", "Pin" :
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" :
"ACCOUNTING"
} ], "Salary" : 50000 }
{ "_id" : 105, "Name" : "ASHOK", "Address" : [ { "City" : "NASIK", "Pin"
:
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" :
"ACCOUNTING"
} ], "Salary" : 40000 }
{ "_id" : 104, "Name" : "JOY", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" }
], "Salary" : 20000 }
{ "_id" : 103, "Name" : "RAM", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" }
], "Salary" : 10000 }
{ "_id" : 102, "Name" : "AKASH", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
```

```
"Salary" : 80000 }
{ "_id" : 101, "Name" : "Dev", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
"Salary" : 78000 }
```

>db.Employee.find().sort({_id:1})

```
{ "_id" : 101, "Name" : "Dev", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
"Salary" : 78000 }
{ "_id" : 102, "Name" : "AKASH", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
"Salary" : 80000 }
{ "_id" : 103, "Name" : "RAM", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" }
], "Salary" : 10000 }
{ "_id" : 104, "Name" : "JOY", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" }
], "Salary" : 20000 }
{ "_id" : 105, "Name" : "ASHOK", "Address" : [ { "City" : "NASIK", "Pin"
:
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" :
"ACCOUNTING"
} ], "Salary" : 40000 }
{ "_id" : 106, "Name" : "RAJ", "Address" : [ { "City" : "NASIK", "Pin" :
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" :
"ACCOUNTING"
} ], "Salary" : 50000 }
>db.Employee.find().sort({$natural:-1}).limit(2)
{ "_id" : 106, "Name" : "RAJ", "Address" : [ { "City" : "NASIK", "Pin" :
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" :
"ACCOUNTING"
} ], "Salary" : 50000 }
{ "_id" : 105, "Name" : "ASHOK", "Address" : [ { "City" : "NASIK", "Pin"
:
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" :
"ACCOUNTING"
} ], "Salary" : 40000 }
```

>db.Employee.find().sort({$natural:1}).limit(2)

```
{ "_id" : 101, "Name" : "Dev", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
"Salary" : 78000 }
{ "_id" : 102, "Name" : "AKASH", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
"Salary" : 80000 }
>db.Employee.find({Salary:{$in:[10000,30000]}})
{ "_id" : 103, "Name" : "RAM", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" }
], "Salary" : 10000 }
>db.Employee.update({"Name":"RAM"},{ $set :{Address:{City: "Nasik"}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>db.Employee.find({"Name":"RAM"})
{ "_id" : 103, "Name" : "RAM", "Address" : { "City" : "Nasik" },
"Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" } ], "Salary" :
10000 }
```

```
>db.Employee.update({"Name":"RAM"},{$inc :{"Salary": 10000 } })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>db.Employee.find({"Name":"RAM"})
{ "_id" : 103, "Name" : "RAM", "Address" : { "City" : "Nasik" },
"Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" } ], "Salary" :
20000 }
```

Retrieve documentwith a particular from collection in mongodb

>db.Employee.find().limit(2).pretty()

```
{
      "_id" : 101,
      "Name" :
      "Dev",
      "Address" : [
            {
                  "City" :
                  "Pune", "Pin" :
                  444043
            }
      ],
      "Department" : [
            {
                  "Dept_id" : 111,
                  "Dept_name" :
                  "HR"
            }
      ],
      "Salary" : 78000
}
{
      "_id" : 102,
      "Name" :
      "AKASH",
      "Address" : [
            {
                  "City" :
                  "Pune", "Pin" :
                  444043
            }
      ],
      "Department" : [
            {
                  "Dept_id" : 111,
                  "Dept_name" :
                  "HR"
            }
      ],
      "Salary" : 80000
}
```

Retrieve document skipping some documents from collection in mongodb

>db.Employee.find().skip(3).pretty()

```
{
      "_id" : 104,
      "Name" :
      "JOY",
      "Address" : [
            {
                  "City" :
                  "Pune", "Pin" :
                  444043
            }
      ],
      "Department" : [
            {
                  "Dept_id" : 112,
                  "Dept_name" :
                  "SALES"
            }
      ],
      "Salary" : 20000
}
{
      "_id" : 105,
      "Name" :
      "ASHOK",
      "Address" : [
            {
                  "City" :
                  "NASIK", "Pin" :
                  411002
            }
      ],
      "Department" : [
            {
                  "Dept_id" : 113,
                  "Dept_name" : "ACCOUNTING"
            }
      ],
      "Salary" : 40000
}
{
      "_id" : 106,
      "Name" :
      "RAJ",
      "Address" : [
            {
                  "City" :
                  "NASIK", "Pin" :
                  411002
            }
      ],
      "Department" : [
            {
                  "Dept_id" : 113,
```

```
                "Dept_name" : "ACCOUNTING"
            }
        ],
        "Salary" : 50000
}
```

REFERENCE BOOK:

Kristina Chodorow, MongoDB The definitive guide, O’Reilly Publications, ISBN:978-93-5110-269-4,2nd Edition.

CONCLUSION:

Understand to implement data from mongodb database with the help of statement and operators.

Assignment: 2

AIM: Implement Map reduces operation with suitable example on above MongoDB database

- Aggregation framework
- Create and drop different types of indexes and explain () to show the advantage of the indexes.

PROBLEM STATEMENT /DEFINITION

Implement Map reduces operation with suitable example on above MongoDB database

- Aggregation framework
- Create and drop different types of indexes and explain () to show the advantage of the indexes.

OBJECTIVE:

- To understand the concept of Mapreduce in mongodb.
- To understand the concept of Aggregation in mongodb.
- To implement the concept of document oriented databases.

THEORY:

- Implements the MapReduce model for processing large data sets.
  Can choose from one of several output options (inline, new collection, merge, replace, reduce)
- MapReduce functions are written in JavaScript.
- Supports non-sharded and sharded input collections.
- Can be used for incremental aggregation over large collections.
- MongoDB 2.2 implements much better support for sharded map reduce output.
- New feature in the Mongodb2.2.0 production release (August, 2012).
- Designed with specific goals of **improving performance** and **usability.**
- Returns result set inline.
- Supports **non-sharded and sharded**input collections.

- Uses a **"pipeline"** approach where objects are transformed as they pass through a series of pipeline operators such as matching, projecting, sorting, and grouping.
- **Pipeline operators need not produce one output document for every input document:** Operators may also generate new documents or filter out documents.
- Map/Reduce involves two steps:
- first, map the data from the collection specified;
- second, reduce the results.

>db.createCollection("Order")

- { "ok" : 1 }

>db.order.insert({cust_id:"A123",amount:500,status:"A"})

- WriteResult({ "nInserted" : 1 })

>db.order.insert({cust_id:"A123",amount:250,status:"A"})

- WriteResult({ "nInserted" : 1 })

>db.order.insert({cust_id:"B212",amount:200,status:"A"})

- WriteResult({ "nInserted" : 1 })

>db.order.insert({cust_id:"A123",amount:300,status:"d"})

- WriteResult({ "nInserted" : 1 })

• Map Function

- var mapFunction1 = function()
- { emit(this.cust_id, this.amount);};

• Reduce Function

- var reduceFunction1 = function(key, values)
- {return Array.sum(values); };

db.order.mapReduce

(mapFunction1, reduceFunction1, {query: {status: "A" },

out: "order_totals"});

"result" : "order_totals",

"timeMillis" : 28,

"counts" : {

"input" :3,

"emit" :3,

"reduce" : 1,

"output" : 2

},

"ok" : 1,}

>db.order.mapReduce(

Map Function -> function() { emit(this.cust_id, this.amount);},

Reduce Function -> function(key, values) { return Array.sum (values)},

Query à {query: { status:"A"},

Output collection à out: "order_ totals"})

{

"result" : "order_totals",

"timeMillis" : 27,

"counts" : {

"input" : 3,

"emit" : 3,

"reduce" : 1,

"output" : 2

},

"ok" : 1,

}

To display result of mapReduce function use collection created in OUT.

Db.<collection name>.find();

db.order_totals.find();

{ "_id" : "A123", "value" : 750 }

{ "_id" : "B212", "value" : 200 }

Implementation of Aggregation:-

```
> use  Teacher
switched to db Teacher
>db.Teacher.find()
{ "_id" : 101, "Name" : "Dev", "Address" : [ {  "City" : "Pune",  "Pin" : 444043 } ],
"Department" : [ {  "Dept_id" : 111,  "Dept_name" : "IT" } ], "Salary" : 78000 }
{ "_id" : 135, "Name" : "Jennifer", "Address" : [ {  "City" : "Mumbai",  "Pin" : 444111 } ],
"Department" : [ {  "Dept_id" : 112,  "Dept_name" : "COMP" } ], "Salary" : 65000 }
{ "_id" : 126, "Name" : "Gaurav", "Address" : [ {  "City" : "Nashik",  "Pin" : 444198 } ],
"Department" : [ {  "Dept_id" : 112,  "Dept_name" : "COMP" } ], "Salary" : 90000 }
{ "_id" : 175, "Name" : "Shree", "Address" : [ {  "City" : "Nagpur",  "Pin" : 444158 } ],
"Department" : [ {  "Dept_id" : 113,  "Dept_name" : "ENTC" } ], "Salary" : 42000 }
{ "_id" : 587, "Name" : "Raman", "Address" : [ {  "City" : "Banglore",  "Pin" : 445754 } ],
"Department" : [ {  "Dept_id" : 113,  "Dept_name" : "ENTC" } ], "Salary" : 79000 }
{ "_id" : 674, "Name" : "Mandar", "Address" : [ {  "City" : "Jalgaon",  "Pin" : 465487 } ],
"Department" : [ {  "Dept_id" : 111,  "Dept_name" : "IT" } ], "Salary" : 88000 }
{ "_id" : 573, "Name" : "Manish", "Address" : [ {  "City" : "Washim",  "Pin" : 547353 } ],
"Department" : [ {  "Dept_id" : 112,  "Dept_name" : "COMP" } ], "Salary" : 65000 }

>db.Teacher.aggregate([
... {$group:{_id:"$Department",totalsalary:{$sum:"$Salary"}}}
... ])
{
        "result" : [
                {
                        "_id" : [
                                {
                                        "Dept_id" : 113,
                                        "Dept_name" : "ENTC"
                                }
                        ],
                        "totalsalary" : 121000
                },
                {
                        "_id" : [
                                {
                                        "Dept_id" : 112,
                                        "Dept_name" : "COMP"
                                }],
                        "totalsalary" : 220000
                },
                {
                        "_id" : [
                                {
```

```
                                        "Dept_id" : 111,
                                        "Dept_name" : "IT"
                                }
                        ],
                        "totalsalary" : 166000
                }
        ],
        "ok" : 1
}
```

>db.Teacher.aggregate([
{$group:{_id:"$Department",totalsalary:{$sum:"$Salary"}}},{$group:{_id:"$_id.Department",AvgSal:{$sum:"$totalsalary"}}}])

```
{ "result" : [ { "_id" : [ ], "AvgSal" : 507000 } ], "ok" : 1 }
```

>db.Teacher.aggregate([
{$group:{_id:"$Department",totalsalary:{$sum:"$Salary"}}},{$match:{totalsalary:{$gte:200000}}}])

```
{
        "result" : [
                {
                        "_id" : [
                                {
                                        "Dept_id" : 112,
                                        "Dept_name" : "COMP"
                                }
                        ],
                        "totalsalary" : 220000
                }
        ],
        "ok" : 1
}
```

>db.Teacher.aggregate([{$group:{_id:"$Department",totalsalary:{$sum:"$Salary"}}}, {$sort:{totalsalary:1}}])

```
{
        "result" : [
                {
                        "_id" : [
                                {
                                        "Dept_id" : 113,
                                        "Dept_name" : "ENTC"
                                }

                        ],
                        "totalsalary" : 121000
                },
                {
                        "_id" : [
                                {
                                        "Dept_id" : 111,
                                        "Dept_name" : "IT"
                                }
```

```
                        ],
                        "totalsalary" : 166000
                },
                {
                        "_id" : [
                                {
                                        "Dept_id" : 112,
                                        "Dept_name" : "COMP"
                                }
                        ],
                        "totalsalary" : 220000
                }
        ],
        "ok" : 1
}
```


>db.Teacher.aggregate([{$group:{_id:"$Department",totalsalary:{$sum:"$Salary"}}}, { $group: { _id:"$_id.Department", big: { $last: "$_id.Dept_name" }, bigsalary: { $last:"$totalsalary"}, small: { $first:"$_id.Dept_name"}, smallsalary: { $first:"$totalsalary"} }}])


```
{
        "result" : [
                {
                        "_id" : [ ],
                        "big" : [
                                "IT"
                        ],
                        "bigsalary" : 166000,
                        "small" : [
                                "ENTC"
                        ],
                        "smallsalary" : 121000
                }
        ],
        "ok" : 1
}
```


REFERENCE BOOK:

Kristina Chodorow, MongoDB The definitive guide, O'Reilly Publications, ISBN:978-93-5110-269-4,2nd Edition.

CONCLUSION:

Understand to mapreduce operation in mongodb

Assignment 3

Aim : Case Study: Design conceptual model using Star and Snowflake schema for any one database.

Problem statement: Design conceptual model using Star and Snowflake schema for any one database.

OBJECTIVE:

1. To understand concepts of multidimensional data.
2. To understand the the relational implementation of the multidimensional data model is typically a star schema, or a snowflake schema.

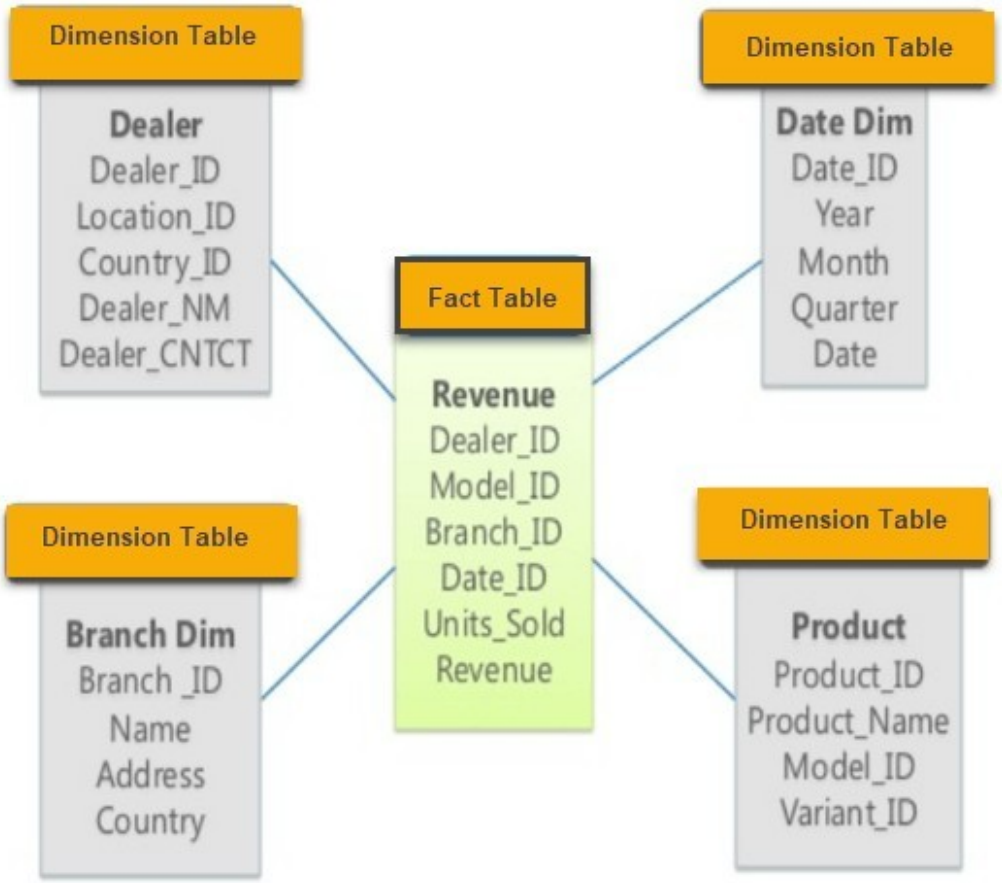
Theory:

The data warehouses are considered modern ancient techniques, since the early days for the relational databases, the idea of the keeping a historical data for reference when it needed has been originated, and the idea was primitive to create archives for the historical data to save these data, despite of the usage of a special techniques for the recovery of these data from the different storage modes. This research applied of structured databases for a trading company operating across the continents, has a set of branches each one has its own stores and showrooms, and the company branch's group of sections with specific activities, such as stores management, showrooms management, accounting management, contracts and other departments. It also assumes that the company center exported software to manage databases for all branches to ensure the safety performance, standardization of processors and prevent the possible errors and bottlenecks problems. Also the research provides this methods the best requirements have been used for the applied of the data warehouse (DW), the information that managed by such an applied must be with high accuracy. It must be emphasized to ensure compatibility information and hedge its security, in schemes domain, been applied to a comparison between the two schemes (Star and Snowflake Schemas) with the concepts of multidimensional database. It turns out that Star Schema is better than Snowflake Schema in (Query complexity, Query performance, Foreign Key Joins),And finally it has been concluded that Star Schema center fact and change, while Snowflake Schema center fact and not change.

Example:

1. **Star Schema**

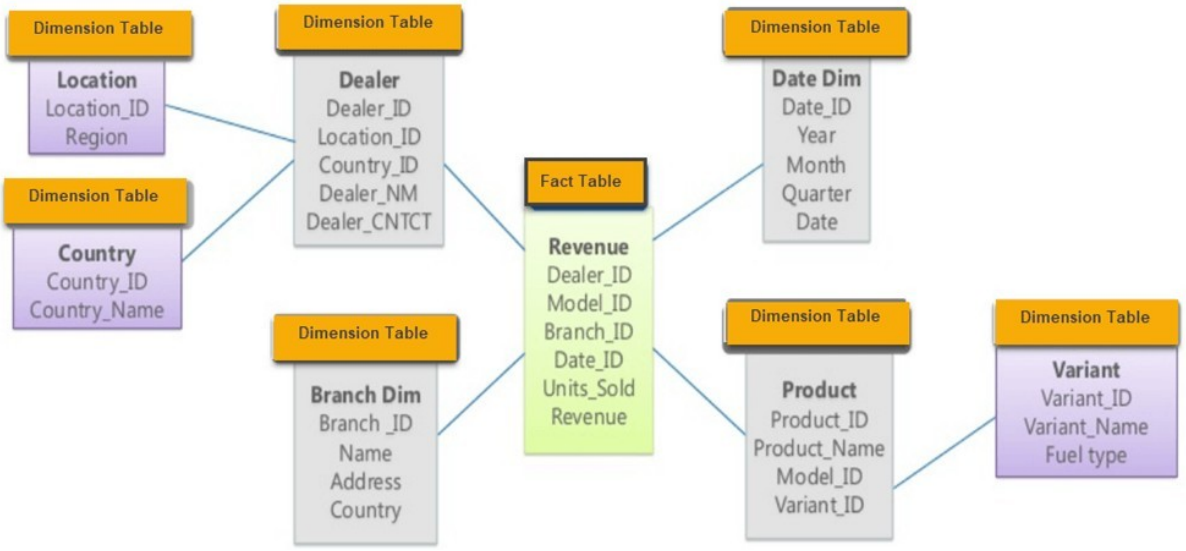
Star Schema in data warehouse, in which the center of the star can have one fact table and a number of associated dimension tables. It is known as star schema as its structure resembles a star. The Star Schema data model is the simplest type of Data Warehouse schema. It is also known as Star Join Schema and is optimized for querying large data sets.



2. **Snowflake Schema?**

Snowflake Schema in data warehouse is a logical arrangement of tables in a multidimensional database such that the ER diagram resembles a snowflake shape. A Snowflake Schema is an extension of a Star Schema, and it adds additional dimensions. The dimension tables are normalized which splits data into additional tables.

In the following Snowflake Schema example, Country is further normalized into an individual table.



- **Multidimensional schema is especially designed to model data warehouse systems**

- The star schema is the simplest type of Data Warehouse schema. It is known as star schema as its structure resembles a star.

- Comparing Snowflake vs Star schema, a Snowflake Schema is an extension of a Star Schema, and it adds additional dimensions. It is called snowflake because its diagram resembles a Snowflake.
- In a star schema, only single join defines the relationship between the fact table and any dimension tables.
- Star schema contains a fact table surrounded by dimension tables.
- Snowflake schema is surrounded by dimension table which are in turn surrounded by dimension table
- A snowflake schema requires many joins to fetch the data.
- Comparing Star vs Snowflake schema, Start schema has simple DB design, while Snowflake schema has very complex DB design.

REFERENCE BOOK:

Jiawei Han, Micheline Kamber, Jian Pei “Data Mining: concepts and techniques”, 2nd Edition, Publisher: Elsevier/Morgan Kaufmann.

CONCLUSION:

Understand to concept of multidimensional data.

PART C:

MINI PROJECT

Mini Project

AIM: Build the mini project based on the relevant applicable concepts of Machine Learning / DAA / ADBMS by forming teams of around 3 to 4 students.

A] Sample ML mini–project Format:

PROBLEM STATEMENT /DEFINITION

Design and Implement any Data science Application using Python. Obtain Data, Scrub data (Data cleaning), Explore data, Prepare and validate data model and Interpret data (Data Visualization). Visualize data using any visualization tool like Matplotlib, ggplot, Tableau etc. Prepare Project Report.

OBJECTIVE:

1. To explore the Data science project life cycle.
2. To identify need of project and define problem statement.
3. To extract and process data.
4. To interpret and analyze results using data visualization.

Mini Project Report Format:

Abstract

Acknowledgement

List of Tables & Figures

Contents

1. Introduction

    1.1 Purpose, Problem statement

    1.2 Scope, Objective

    1.3 Definition, Acronym, and Abbreviations

    1.4 References

2. Literature Survey

    2.1 Introduction

    2.2 Detail Literature survey

    2.4 Findings of Literature survey

3. System Architecture and Design

3.1 Detail Architecture

3.2 Dataset Description

3.3 Detail Phases

3.4Algorithms

4. Experimentation and Results

4.1 Phase-wise results

4.2 Explanation with example

4.3 Comparison of result with standard

4.4Accuracy

4.5 Visualization

4.6 Tools used

5. Conclusion and Future scope

5.1 Conclusion

5.2 Future scope

References

Annexure:

A. GUIs / Screen Snapshot of the System Developed

B. Implementation /code

B] Sample ADBMS mini–project Format:

PROBLEM STATEMENT /DEFINITION

Build the mini project based on the requirement document and design prepared as a part of Database Management Lab in second year.

Form teams of around 3 to 4 people.

A. Develop the application: Build a suitable GUI by using forms and placing the controls on it for any application. Proper data entry validations are expected.
B. Add the database connection with front end. Implement the basic CRUD operations.
C. Prepare and submit report to include Title of the Project, Abstract, List the hardware and software requirements at the backend and at the front end, Source Code, Graphical User Interface, Conclusion.

OBJECTIVE:

3. To understand applications of document-oriented database by implementing mini project.
4. To learn effective UI designs.
5. To learn to design & implement database system for specific domain.
6. To learn to design system architectural & flow diagram.

.

Mini Project Report Format:

Abstract

Acknowledgement

List of Tables & Figures

Contents

1. Introduction

    1.1 Purpose

    1.2 Scope

    1.3 Definition, Acronym, and Abbreviations

    1.4 References

    1.5 Developers' Responsibilities: An Overview

2. General Description

    2.1 Product Function Perspective

    2.2 User Characteristics.

    2.4 General Constraints

    2.5 Assumptions and Dependencies

3. Specific Requirements

    3.1 Inputs and Outputs

    3.2 Functional Requirements