# Practical 1

**Creating database employee  
Create collections emp\_personal\_details with emp\_id, emp\_name, emp\_address, emp\_DOB, emp\_age, emp\_mobilenumber**

// Switch to the 'employee' database. If it doesn't exist, it will be created.  
use employee  
  
// Optional: Explicitly create the collections. They will be created automatically on the first insert.  
db.createCollection('emp\_personal\_details')  
db.createCollection('emp\_professional\_details')

# Practical 2

**Create another collection emp\_professional\_details with emp\_id, emp\_name, designation , salary , incentive, working hours**

// Create the emp\_professional\_details collection  
db.createCollection('emp\_professional\_details')  
  
// This collection will contain:  
// emp\_id, emp\_name, designation, salary, incentive, working\_hours

# Practical 3

**Insert 10 records in collection emp\_personal\_details and emp\_professional\_details  
2. Show all the employees having designation manager  
3. Show all the employees having salary 6000**

// Insert 10 records into the emp\_personal\_details collection.  
db.emp\_personal\_details.insertMany([  
 { emp\_id: 1, emp\_name: "John Doe", emp\_address: "123 Main St", emp\_DOB: "1990-05-15", emp\_age: 35, emp\_mobilenumber: "123-456-7890" },  
 { emp\_id: 2, emp\_name: "Jane Smith", emp\_address: "456 Oak Ave", emp\_DOB: "1988-11-20", emp\_age: 37, emp\_mobilenumber: "987-654-3210" },  
 { emp\_id: 3, emp\_name: "Peter Jones", emp\_address: "789 Pine Ln", emp\_DOB: "1975-02-28", emp\_age: 50, emp\_mobilenumber: "555-123-4567" },  
 { emp\_id: 4, emp\_name: "Mary Williams", emp\_address: "101 Maple Rd", emp\_DOB: "1960-07-10", emp\_age: 65, emp\_mobilenumber: "222-333-4444" },  
 { emp\_id: 5, emp\_name: "David Brown", emp\_address: "202 Birch Blvd", emp\_DOB: "1995-09-01", emp\_age: 30, emp\_mobilenumber: "111-222-3333" },  
 { emp\_id: 6, emp\_name: "Laura Miller", emp\_address: "303 Cedar Dr", emp\_DOB: "1982-04-05", emp\_age: 43, emp\_mobilenumber: "444-555-6666" },  
 { emp\_id: 7, emp\_name: "James Wilson", emp\_address: "404 Elm Ct", emp\_DOB: "1958-12-12", emp\_age: 67, emp\_mobilenumber: "777-888-9999" },  
 { emp\_id: 8, emp\_name: "Sarah Davis", emp\_address: "505 Ash Way", emp\_DOB: "1993-06-25", emp\_age: 32, emp\_mobilenumber: "999-888-7777" },  
 { emp\_id: 9, emp\_name: "Robert Taylor", emp\_address: "606 Spruce Ave", emp\_DOB: "1980-08-30", emp\_age: 45, emp\_mobilenumber: "666-777-8888" },  
 { emp\_id: 10, emp\_name: "Linda Thomas", emp\_address: "707 Pine Pkwy", emp\_DOB: "1970-03-18", emp\_age: 55, emp\_mobilenumber: "888-999-0000" }  
]);  
  
// Insert 10 records into the emp\_professional\_details collection.  
db.emp\_professional\_details.insertMany([  
 { emp\_id: 1, emp\_name: "John Doe", designation: "Software Engineer", salary: 75000, incentive: 2000, working\_hours: 45 },  
 { emp\_id: 2, emp\_name: "Jane Smith", designation: "Manager", salary: 90000, incentive: 3000, working\_hours: 50 },  
 { emp\_id: 3, emp\_name: "Peter Jones", designation: "Accountant", salary: 60000, incentive: 1500, working\_hours: 40 },  
 { emp\_id: 4, emp\_name: "Mary Williams", designation: "Clerk", salary: 45000, incentive: 500, working\_hours: 35 },  
 { emp\_id: 5, emp\_name: "David Brown", designation: "Software Engineer", salary: 85000, incentive: 2500, working\_hours: 55 },  
 { emp\_id: 6, emp\_name: "Laura Miller", designation: "Manager", salary: 95000, incentive: 3500, working\_hours: 52 },  
 { emp\_id: 7, emp\_name: "James Wilson", designation: "Clerk", salary: 48000, incentive: 600, working\_hours: 38 },  
 { emp\_id: 8, emp\_name: "Sarah Davis", designation: "Accountant", salary: 62000, incentive: 1800, working\_hours: 42 },  
 { emp\_id: 9, emp\_name: "Robert Taylor", designation: "Software Engineer", salary: 80000, incentive: 2200, working\_hours: 48 },  
 { emp\_id: 10, emp\_name: "Linda Thomas", designation: "Manager", salary: 100000, incentive: 4000, working\_hours: 60 }  
]);  
  
// Show all the employees having designation manager.  
db.emp\_professional\_details.find({ designation: "Manager" });  
  
// Show all the employees having salary 6000.  
db.emp\_professional\_details.find({ salary: 6000 });

# Practical 4

**Update the collection emp\_personal\_details , add field status and set it to retired where age is greater than 60.  
2. Update collection emp\_professional\_details, give incentive 5000 to employees whose working hours is greater than 45 per week  
3. Add 1000 to the salary employee whose designation is accountant**

// Update the 'emp\_personal\_details' collection.  
// Add the 'status' field and set it to 'retired' for employees whose age is greater than 60.  
db.emp\_personal\_details.updateMany(  
 { emp\_age: { $gt: 60 } },  
 { $set: { status: "retired" } }  
);  
  
// Update the 'emp\_professional\_details' collection.  
// Give an incentive of 5000 to employees whose working hours is greater than 45 per week.  
db.emp\_professional\_details.updateMany(  
 { working\_hours: { $gt: 45 } },  
 { $set: { incentive: 5000 } }  
);  
  
// Add 1000 to the salary of employees whose designation is 'accountant'.  
db.emp\_professional\_details.updateMany(  
 { designation: "Accountant" },  
 { $inc: { salary: 1000 } }  
);

# Practical 5

**Create index on emp\_id in collection emp\_professional\_details  
2. Create multiple index on emp\_id,emp\_name in collection emp\_professonal details**

// Create a single index on 'emp\_id' in the 'emp\_professional\_details' collection.  
db.emp\_professional\_details.createIndex({ emp\_id: 1 });  
  
// Create a compound index on 'emp\_id' and 'emp\_name'.  
db.emp\_professional\_details.createIndex({ emp\_id: 1, emp\_name: 1 });

# Practical 6

**1. Find sum of salaries of employees having designation clerk.  
2. Filter the employees having the designation software engineer and find the minimum salary.**

// Find the sum of salaries for all employees with the designation 'clerk'.  
db.emp\_professional\_details.aggregate([  
 { $match: { designation: "Clerk" } },  
 { $group: { \_id: null, total\_salary: { $sum: "$salary" } } }  
]);  
  
// Filter for employees with the designation 'software engineer' and find the minimum salary.  
db.emp\_professional\_details.aggregate([  
 { $match: { designation: "Software Engineer" } },  
 { $group: { \_id: null, min\_salary: { $min: "$salary" } } }  
]);

# Practical 7

**Use unwind command and show the employees whose mobile number is stored in array  
2. Use skip command to skip first 3 records and display rest of records  
3. Use limit command to show only first four records of collection**

// First, update a document to have an array of mobile numbers to demonstrate '$unwind'.  
db.emp\_personal\_details.updateOne(  
 { emp\_id: 1 },  
 { $set: { emp\_mobilenumber: ["123-456-7890", "098-765-4321"] } }  
);  
  
// Use the '$unwind' command to show documents with mobile numbers stored in an array.  
// This deconstructs the array field, outputting one document for each element.  
db.emp\_personal\_details.aggregate([  
 { $unwind: "$emp\_mobilenumber" }  
]);  
  
// Use the '$skip' command to skip the first 3 records and display the rest.  
db.emp\_professional\_details.find().skip(3);  
  
// Use the '$limit' command to show only the first four records.  
db.emp\_professional\_details.find().limit(4);

# Practical 8

**Create replica set of employee database and insert records in primary node and display the same records in secondary nodes**

// Start MongoDB with replica set support  
// Command line: mongod --port 27017 --dbpath /data/rs0-0 --replSet rs0  
  
// Initialize the replica set  
rs.initiate()  
  
// Add secondary members  
rs.add("hostname:port")  
  
// Connect to the primary node and insert a record.  
db.emp\_personal\_details.insertOne({   
 emp\_id: 11,   
 emp\_name: "Alice Johnson",   
 emp\_address: "123 New St",   
 emp\_DOB: "1992-01-15",   
 emp\_age: 33,   
 emp\_mobilenumber: "123-000-1111"   
});  
  
// Connect to a secondary node (read-only by default) and display the same records.  
// You must set the secondary to allow reads.  
rs.secondaryOk();  
db.emp\_personal\_details.find();

# Practical 9

**Create a MongoDB collection named restaurants to store the following information about restaurants:  
Building number  
Street name  
Zip code  
Coordinates (longitude and latitude)  
Borough  
Cuisine type  
Grades (each grade includes: date, grade (A/B/C), and score)**

// Switch to test database and create restaurants collection  
use test;  
  
// Create a collection named 'restaurants'.  
db.createCollection("restaurants");  
  
// Sample document structure for restaurants collection  
/\*  
{  
 "address": {  
 "building": "1007",  
 "coord": [-73.856077, 40.848447],  
 "street": "Morris Park Ave",  
 "zipcode": "10462"  
 },  
 "borough": "Bronx",  
 "cuisine": "Bakery",  
 "grades": [  
 {  
 "date": ISODate("2014-03-03T00:00:00Z"),  
 "grade": "A",  
 "score": 2  
 }  
 ],  
 "name": "Morris Park Bake Shop",  
 "restaurant\_id": "30075445"  
}  
\*/

# Practical 10

**1.Write a MongoDB query to display all the documents in the collection restaurants  
2. Write a MongoDB query to display the fields,restaurant\_id, name, borough and cuisine for all the documents in the collection restaurant**

// Display all documents in the restaurants collection  
db.restaurants.find();  
  
// Display specific fields: restaurant\_id, name, borough and cuisine  
db.restaurants.find({}, { restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 });

# Practical 11

**1.Write a MongoDB query to display the fields restaurant\_id, name, borough and cuisine, but exclude the field \_id for all the documents in the collection restaurant  
2. Write a MongoDB query to display all the restaurant which is in the borough Bronx**

// Display specific fields excluding \_id  
db.restaurants.find({}, { restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 });  
  
// Display all restaurants in the borough Bronx  
db.restaurants.find({ borough: "Bronx" });

# Practical 12

**1. Write a MongoDB query to display the first 5 restaurants which are in the borough Bronx.  
2. Write a MongoDB query to display the next 5 restaurants after skipping first 5 which are in the borough Bronx**

// Find the first 5 restaurants in the Bronx  
db.restaurants.find({ borough: "Bronx" }).limit(5);  
  
// Skip the first 5 and display the next 5  
db.restaurants.find({ borough: "Bronx" }).skip(5).limit(5);

# Practical 13

**1.Write a MongoDB query to find the restaurants who achieved a score more than 90  
2.Write a MongoDB query to find the restaurantsthat achieved a score, more than 80 but less than 100**

// Find restaurants with a score greater than 90  
db.restaurants.find({ "grades.score": { $gt: 90 } });  
  
// Find restaurants with a score between 80 and 100  
db.restaurants.find({ "grades.score": { $gt: 80, $lt: 100 } });

# Practical 14

**Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American ' and achieved a grade point 'A' not belonging to the boroughBrooklyn. The document must be displayed according to the cuisine in descending order**

// Find restaurants with complex conditions and sort by cuisine in descending order  
db.restaurants.find({  
 cuisine: { $ne: 'American ' },  
 "grades.grade": 'A',  
 borough: { $ne: 'Brooklyn' }  
}).sort({ cuisine: -1 });

# Practical 15

**Write a MongoDB query to find the restaurant Id,name, borough and cuisine for those restaurants which contain 'Wil' as first three letters for its name**

// Find restaurant names that start with 'Wil'  
db.restaurants.find(  
 { name: /^Wil/ },  
 { restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 }  
);

# Practical 16

**Write a MongoDB query to find the restaurant Id,name, borough and cuisine for those restaurants which contain 'ces' as the last three letters for its name.**

// Find restaurant names that end with 'ces'  
db.restaurants.find(  
 { name: /ces$/ },  
 { restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 }  
);

# Practical 17

**Write a MongoDB query to find the restaurant Id,name, borough and cuisine for those restaurants which contain 'Reg' as three letters somewhere in its name**

// Find restaurant names that contain 'Reg' anywhere in the name  
db.restaurants.find(  
 { name: /Reg/ },  
 { restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 }  
);

# Practical 18

**Write a MongoDB query to find the restaurants which belong to the borough Bronx and prepared either American or Chinese dish**

// Find restaurants in Bronx with American or Chinese cuisine  
db.restaurants.find({  
 borough: "Bronx",  
 cuisine: { $in: ["American", "Chinese"] }  
});

# Practical 19

**Write a MongoDB query to find the restaurant Id,name, borough and cuisine for those restaurants which belong to the borough Staten Island or Queens or Bronx or Brooklyn.**

// Find restaurants in specific boroughs using $in operator  
db.restaurants.find(  
 { borough: { $in: ["Staten Island", "Queens", "Bronx", "Brooklyn"] } },  
 { restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 }  
);

# Practical 20

**Write a MongoDB query to find the restaurant Id,name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island Or Queens or Bronxor Brooklyn.**

// Find restaurants not in specific boroughs using $nin operator  
db.restaurants.find(  
 { borough: { $nin: ["Staten Island", "Queens", "Bronx", "Brooklyn"] } },  
 { restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 }  
);

# Practical 21

**Write a MongoDB query to find the restaurant Id,name, borough and cuisine for those restaurants which achieved a score which is not more than 10**

// Find restaurants where score is not more than 10  
db.restaurants.find(  
 { "grades.score": { $not: { $gt: 10 } } },  
 { restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 }  
);

# Practical 22

**Write a MongoDB query to find the restaurant Id,name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinese' or restaurant's name begins with letter 'Wil'**

// Use $or operator to combine different conditions  
db.restaurants.find(  
 { $or: [{ cuisine: { $nin: ["American", "Chinese"] } }, { name: /^Wil/ }] },  
 { restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 }  
);

# Practical 23

**Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns**

// Sort restaurants by name in descending order  
db.restaurants.find().sort({ name: -1 });

# Practical 24

**Write a MongoDB query to arranged the name of the cuisine in ascending order and for that same cuisine borough should be in descending order.**

// Sort by cuisine ascending and borough descending for same cuisine  
db.restaurants.find().sort({ cuisine: 1, borough: -1 });

# Practical 25

**Write a MongoDB query to know whether all the addresses contains the street or no**

// Check if address.street field exists  
db.restaurants.find({ "address.street": { $exists: true } });

# Practical 26

**Write a MongoDBquery which will select all documents in the restaurants collection where the coord field value is Double**

// Find documents where address.coord field has Double type  
db.restaurants.find({ "address.coord": { $type: "double" } });

# Practical 27

**Write a MongoDBquery which will select the restaurant Id, name and grades for those restaurants which returns 0 as a remainder after dividing thescore by 7.**

// Use aggregation to find restaurants with scores divisible by 7  
db.restaurants.aggregate([  
 { $unwind: "$grades" },  
 {  
 $match: {  
 "grades.score": { $exists: true },  
 $expr: { $eq: [{ $mod: ["$grades.score", 7] }, 0] }  
 }  
 },  
 {  
 $project: {  
 restaurant\_id: 1,  
 name: 1,  
 grades: 1,  
 \_id: 0  
 }  
 }  
]);

# Practical 28

**Write a MongoDB query to find the restaurant name, borough, longitude and attitude and cuisine for those restaurants which contains 'mon' as three letters somewhere in its name**

// Find restaurant names containing 'mon' and project specific fields  
db.restaurants.find(  
 { name: /mon/ },  
 { name: 1, borough: 1, "address.coord": 1, cuisine: 1, \_id: 0 }  
);

# Practical 29

**Write a MongoDB query to use sum, avg,min max expression**

// Example using aggregation expressions for statistical analysis  
// Find the sum, average, min, and max salary for each designation  
db.emp\_professional\_details.aggregate([  
 {  
 $group: {  
 \_id: "$designation",  
 total\_salary: { $sum: "$salary" },  
 avg\_salary: { $avg: "$salary" },  
 min\_salary: { $min: "$salary" },  
 max\_salary: { $max: "$salary" }  
 }  
 }  
]);  
  
// Example with restaurants - find statistics for scores  
db.restaurants.aggregate([  
 { $unwind: "$grades" },  
 {  
 $group: {  
 \_id: "$cuisine",  
 total\_score: { $sum: "$grades.score" },  
 avg\_score: { $avg: "$grades.score" },  
 min\_score: { $min: "$grades.score" },  
 max\_score: { $max: "$grades.score" }  
 }  
 }  
]);

# Practical 30

**1.create backup of collections emp\_personal\_details and emp\_professional\_Details  
2.Delete some record and then restore it from backup  
3.Export the collection in csv and json format**

// Create backup using mongodump (command line)  
mongodump --db employee --collection emp\_personal\_details --out ./employee\_backup  
mongodump --db employee --collection emp\_professional\_details --out ./employee\_backup  
  
// Delete some records from the collection  
db.emp\_personal\_details.deleteOne({ emp\_id: 1 });  
db.emp\_professional\_details.deleteMany({ designation: "Clerk" });  
  
// Restore the collection from the backup (command line)  
mongorestore --db employee --collection emp\_personal\_details ./employee\_backup/employee/emp\_personal\_details.bson  
mongorestore --db employee --collection emp\_professional\_details ./employee\_backup/employee/emp\_professional\_details.bson  
  
// Export to JSON format (command line)  
mongoexport --db employee --collection emp\_professional\_details --out emp\_professional\_details.json  
  
// Export to CSV format with specific fields (command line)  
mongoexport --db employee --collection emp\_professional\_details --type=csv --fields emp\_id,emp\_name,designation,salary --out emp\_professional\_details.csv  
  
// Alternative: Export using MongoDB Compass or MongoDB Atlas UI