Sameer Khatwani

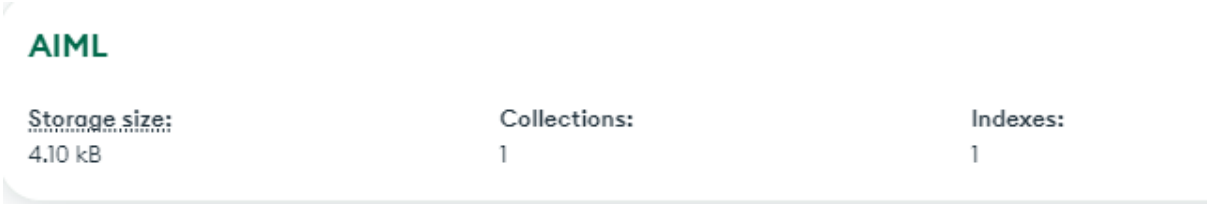
AIML-B1 FS3

22070126099

Problem statement:

Create a database from .json file and execute NO SQL Queries.
1. Create Database: Create a MongoDB database named AIML.

Soln.



2. Create Collection: Create a collection named employee within the AIML database.

Soln.

In MongoDB Compass:

1. Click "Create Database"
2. Database Name: **AIML**
3. Collection Name: **employee**
4. Click Create

3. Import Employee Records: Import employee records from a JSON file into the employee collection. Save the following data in a json file and use it for import in Mongodb.

```
[
    {
        "Id": 1,
        "Name": "Charlie Moore",
        "Age": 44,
        "Gender": "Male",
        "Project_id": 4,
        "Hrs_worked": 12
    },
    {
        "Id": 2,
        "Name": "Frank Smith",
        "Age": 36,
        "Gender": "Female",
        "Project_id": 1,
        "Hrs_worked": 12
    },
    {
        "Id": 3,
        "Name": "Hannah Brown",
```

```
        "Age": 33,
        "Gender": "Female",
        "Project_id": 2,
        "Hrs_worked": 36
    },
    {
        "Id": 4,
        "Name": "Hannah Davis",
        "Age": 31,
        "Gender": "Female",
        "Project_id": 2,
        "Hrs_worked": 22
    },
    {
        "Id": 5,
        "Name": "Bob Davis",
        "Age": 41,
        "Gender": "Female",
        "Project_id": 4,
        "Hrs_worked": 16
    },
    {
        "Id": 6,
        "Name": "Hannah Davis",
        "Age": 57,
        "Gender": "Male",
        "Project_id": 2,
        "Hrs_worked": 28
    },
    {
        "Id": 7,
        "Name": "Frank Brown",
        "Age": 49,
        "Gender": "Male",
        "Project_id": 4,
        "Hrs_worked": 30
    },
    {
        "Id": 8,
        "Name": "David Williams",
        "Age": 60,
        "Gender": "Male",
        "Project_id": 2,
        "Hrs_worked": 34
    },
    {
        "Id": 9,
        "Name": "Charlie Johnson",
        "Age": 49,
        "Gender": "Male",
        "Project_id": 3,
        "Hrs_worked": 32
    },
    {
```

```
        "Id": 10,
        "Name": "Charlie Johnson",
        "Age": 60,
        "Gender": "Male",
        "Project_id": 4,
        "Hrs_worked": 36
    },
    {
        "Id": 11,
        "Name": "Grace Wilson",
        "Age": 42,
        "Gender": "Female",
        "Project_id": 2,
        "Hrs_worked": 15
    },
    {
        "Id": 12,
        "Name": "Isaac Smith",
        "Age": 32,
        "Gender": "Male",
        "Project_id": 2,
        "Hrs_worked": 23
    },
    {
        "Id": 13,
        "Name": "Emma Jones",
        "Age": 52,
        "Gender": "Male",
        "Project_id": 1,
        "Hrs_worked": 33
    },
    {
        "Id": 14,
        "Name": "Isaac Wilson",
        "Age": 21,
        "Gender": "Female",
        "Project_id": 3,
        "Hrs_worked": 32
    },
    {
        "Id": 15,
        "Name": "Hannah Davis",
        "Age": 43,
        "Gender": "Male",
        "Project_id": 4,
        "Hrs_worked": 32
    },
    {
        "Id": 16,
        "Name": "Alice Moore",
        "Age": 29,
        "Gender": "Male",
        "Project_id": 1,
        "Hrs_worked": 39
```

```
    },
    {
        "Id": 17,
        "Name": "David Miller",
        "Age": 25,
        "Gender": "Female",
        "Project_id": 4,
        "Hrs_worked": 32
    },
    {
        "Id": 18,
        "Name": "Isaac Wilson",
        "Age": 32,
        "Gender": "Female",
        "Project_id": 2,
        "Hrs_worked": 31
    },
    {
        "Id": 19,
        "Name": "Charlie Williams",
        "Age": 59,
        "Gender": "Female",
        "Project_id": 1,
        "Hrs_worked": 31
    },
    {
        "Id": 20,
        "Name": "Frank Miller",
        "Age": 55,
        "Gender": "Female",
        "Project_id": 4,
        "Hrs_worked": 24
    }
]
```

4. insertOne: Inserts a single document into the collection.

```
{ "Id": 21, "Name": "John Doe", "Project_id": 2, "Hrs_worked": 35 }
```

Soln.

```
> db.employee.insertOne({
    "Id": 21,
    "Name": "John Doe",
    "Project_id": 2,
    "Hrs_worked": 35
  })
< {
    acknowledged: true,
    insertedId: ObjectId('67f163ee53f6b261d007b4b1')
  }
```

5. insertMany: Inserts multiple documents into the collection.

```
{ "Id": 22, "Name": "Jane Smith", "Project_id": 1, "Hrs_worked": 28 },
{ "Id": 23, "Name": "Alice Johnson", "Project_id": 3, "Hrs_worked": 42 }
```

Soln.

```
> db.employee.insertMany([
    {"Id": 22, "Name": "Jane Smith", "Project_id": 1, "Hrs_worked": 28},
    {"Id": 23, "Name": "Alice Johnson", "Project_id": 3, "Hrs_worked": 42}
  ])
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('67f1642353f6b261d007b4b2'),
      '1': ObjectId('67f1642353f6b261d007b4b3')
    }
  }
```

6. updateOne: Updates a single document that matches the filter.

```
{ "Id": 21 }, { $set: { "Hrs_worked": 40 } }
```

Soln.

```
> db.employee.updateOne(
    { "Id": 21 },
    { $set: { "Hrs_worked": 40 } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

7. updateMany: Updates multiple documents that match the filter.

```
{ "Hrs_worked": { $gt: 30 } }, { $set: { "Overtime": true } }
```

Soln.

```
> db.employee.updateMany(
    { "Hrs_worked": { $gt: 30 } },
    { $set: { "Overtime": true } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 13,
    modifiedCount: 13,
    upsertedCount: 0
  }
```

8. find an employee by their ID.

Ans.

```
> db.employee.find({ "Id": 21 })
< {
    _id: ObjectId('67f163ee53f6b261d007b4b1'),
    Id: 21,
    Name: 'John Doe',
    Project_id: 2,
    Hrs_worked: 40,
    Overtime: true
  }
```

9. How would you retrieve all employees who are assigned to a specific project ID?

Soln.

```
> db.employee.find({ "Project_id": 3 })
< {
    _id: ObjectId('67f12c506e7ee3d0db1b1b1e'),
    Id: 9,
    Name: 'Charlie Johnson',
    Age: 49,
    Gender: 'Male',
    Project_id: 3,
    Hrs_worked: 32,
    Overtime: true
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b23'),
    Id: 14,
    Name: 'Isaac Wilson',
    Age: 21,
    Gender: 'Female',
    Project_id: 3,
    Hrs_worked: 32,
    Overtime: true
  }
  {
    _id: ObjectId('67f1642353f6b261d007b4b3'),
    Id: 23,
    Name: 'Alice Johnson',
    Project_id: 3,
    Hrs_worked: 42,
    Overtime: true
  }
```

10. Write a query to find employees who have worked more than 30 hours.

Soln.

```
> db.employee.find({ "Hrs_worked": { $gt: 30 } }
< {
    _id: ObjectId('67f12c506e7ee3d0db1b1b18'),
    Id: 3,
    Name: 'Hannah Brown',
    Age: 33,
    Gender: 'Female',
    Project_id: 2,
    Hrs_worked: 36,
    Overtime: true
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b1d'),
    Id: 8,
    Name: 'David Williams',
    Age: 60,
    Gender: 'Male',
    Project_id: 2,
    Hrs_worked: 34,
    Overtime: true
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b1e'),
    Id: 9,
    Name: 'Charlie Johnson',
    Age: 49,
    Gender: 'Male',
    Project_id: 3,
    Hrs_worked: 32,
    Overtime: true
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b1f'),
    Id: 10,
    Name: 'Charlie Johnson',
    Age: 60,
    Gender: 'Male',
    Project_id: 4,
    Hrs_worked: 36,
    Overtime: true
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b22'),
    Id: 13,
    Name: 'Emma Jones',
    Age: 52,
    Gender: 'Male',
    Project_id: 1,
    Hrs_worked: 33,
    Overtime: true
  }
  {
```

11. Can you demonstrate how to use the $gt operator to find employees who are older than 40?

Soln.

```
> db.employee.find({ "Age": { $gt: 40 } })
< {
    _id: ObjectId('67f12c506e7ee3d0db1b1b16'),
    Id: 1,
    Name: 'Charlie Moore',
    Age: 44,
    Gender: 'Male',
    Project_id: 4,
    Hrs_worked: 12
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b1a'),
    Id: 5,
    Name: 'Bob Davis',
    Age: 41,
    Gender: 'Female',
    Project_id: 4,
    Hrs_worked: 16
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b1b'),
    Id: 6,
    Name: 'Hannah Davis',
    Age: 57,
    Gender: 'Male',
    Project_id: 2,
    Hrs_worked: 28
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b1c'),
    Id: 7,
    Name: 'Frank Brown',
    Age: 49,
    Gender: 'Male',
    Project_id: 4,
    Hrs_worked: 30
  }
```

12. Explain the purpose of sorting in MongoDB queries.

Soln.

Sorting in MongoDB queries is used to organize the results returned from a query in a specific order. This can enhance the readability of the data and make it easier to analyze. For example, sorting employee records by age can help quickly identify the youngest or oldest employees. Sorting can be done in ascending or descending order based on one or more fields.

13. Sort the Employee table based on Age in Ascending order and display.

Soln.

```
> db.employee.find().sort({ "Age": 1 })
< {
    _id: ObjectId('67f163ee53f6b261d007b4b1'),
    Id: 21,
    Name: 'John Doe',
    Project_id: 2,
    Hrs_worked: 40,
    Overtime: true
  }
  {
    _id: ObjectId('67f1642353f6b261d007b4b2'),
    Id: 22,
    Name: 'Jane Smith',
    Project_id: 1,
    Hrs_worked: 28
  }
  {
    _id: ObjectId('67f1642353f6b261d007b4b3'),
    Id: 23,
    Name: 'Alice Johnson',
    Project_id: 3,
    Hrs_worked: 42,
    Overtime: true
  }
```

14. Sort the Employee table based on Hrs_worked in Descending order and display.

Soln.

```
> db.employee.find().sort({ "Hrs_worked": -1 })
< {
    _id: ObjectId('67f1642353f6b261d007b4b3'),
    Id: 23,
    Name: 'Alice Johnson',
    Project_id: 3,
    Hrs_worked: 42,
    Overtime: true
  }
  {
    _id: ObjectId('67f163ee53f6b261d007b4b1'),
    Id: 21,
    Name: 'John Doe',
    Project_id: 2,
    Hrs_worked: 40,
    Overtime: true
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b25'),
    Id: 16,
    Name: 'Alice Moore',
    Age: 29,
    Gender: 'Male',
    Project_id: 1,
    Hrs_worked: 39,
    Overtime: true
  }
  {
```

15. Find Employee whose age is greater then 30 and Has_Worked greater then 20.

Ans.

```
> db.employee.find({
    $and: [
      { "Age": { $gt: 30 } },
      { "Hrs_worked": { $gt: 20 } }
    ]
  })
< {
    _id: ObjectId('67f12c506e7ee3d0db1b1b18'),
    Id: 3,
    Name: 'Hannah Brown',
    Age: 33,
    Gender: 'Female',
    Project_id: 2,
    Hrs_worked: 36,
    Overtime: true
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b19'),
    Id: 4,
    Name: 'Hannah Davis',
    Age: 31,
    Gender: 'Female',
    Project_id: 2,
    Hrs_worked: 22
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b1b'),
    Id: 6,
    Name: 'Hannah Davis',
    Age: 57,
    Gender: 'Male',
    Project_id: 2,
    Hrs_worked: 28
  }
```

16. Find Employee whose Gender is Male or Has_Worked greater then 25.

Soln.

```
> db.employee.find({
    $or: [
      { "Gender": "Male" },
      { "Hrs_worked": { $gt: 25 } }
    ]
  })
< {
    _id: ObjectId('67f12c506e7ee3d0db1b1b16'),
    Id: 1,
    Name: 'Charlie Moore',
    Age: 44,
    Gender: 'Male',
    Project_id: 4,
    Hrs_worked: 12
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b18'),
    Id: 3,
    Name: 'Hannah Brown',
    Age: 33,
    Gender: 'Female',
    Project_id: 2,
    Hrs_worked: 36,
    Overtime: true
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b1b'),
    Id: 6,
    Name: 'Hannah Davis',
    Age: 57,
    Gender: 'Male',
    Project_id: 2,
    Hrs_worked: 28
  }
```

17. Find Employee whose Project_id is not 3.

Soln.

```
> db.employee.find({ "Project_id": { $ne: 3 } })
< {
    _id: ObjectId('67f12c506e7ee3d0db1b1b16'),
    Id: 1,
    Name: 'Charlie Moore',
    Age: 44,
    Gender: 'Male',
    Project_id: 4,
    Hrs_worked: 12
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b17'),
    Id: 2,
    Name: 'Frank Smith',
    Age: 36,
    Gender: 'Female',
    Project_id: 1,
    Hrs_worked: 12
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b18'),
    Id: 3,
    Name: 'Hannah Brown',
    Age: 33,
    Gender: 'Female',
    Project_id: 2,
    Hrs_worked: 36,
    Overtime: true
  }
```

18. Write a MongoDB query to find all employees who are between the ages of 25 and 35.

Soln.

```
> db.employee.find({ "Age": { $gte: 25, $lte: 35 }
< {
    _id: ObjectId('67f12c506e7ee3d0db1b1b18'),
    Id: 3,
    Name: 'Hannah Brown',
    Age: 33,
    Gender: 'Female',
    Project_id: 2,
    Hrs_worked: 36,
    Overtime: true
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b19'),
    Id: 4,
    Name: 'Hannah Davis',
    Age: 31,
    Gender: 'Female',
    Project_id: 2,
    Hrs_worked: 22
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b21'),
    Id: 12,
    Name: 'Isaac Smith',
    Age: 32,
    Gender: 'Male',
    Project_id: 2,
    Hrs_worked: 23
  }
```

19. How would you retrieve employees who have worked between 20 and 30 hours?

Soln.

```
> db.employee.find({ "Hrs_worked": { $gte: 20, $lte: 30 } })
< {
    _id: ObjectId('67f12c506e7ee3d0db1b1b19'),
    Id: 4,
    Name: 'Hannah Davis',
    Age: 31,
    Gender: 'Female',
    Project_id: 2,
    Hrs_worked: 22
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b1b'),
    Id: 6,
    Name: 'Hannah Davis',
    Age: 57,
    Gender: 'Male',
    Project_id: 2,
    Hrs_worked: 28
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b1c'),
    Id: 7,
    Name: 'Frank Brown',
    Age: 49,
    Gender: 'Male',
    Project_id: 4,
    Hrs_worked: 30
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b21'),
    Id: 12,
    Name: 'Isaac Smith',
    Age: 32,
    Gender: 'Male',
    Project_id: 2,
    Hrs_worked: 23
  }
```

20. Write a query to find employees who are either working on Project 1 or Project 2.
Soln.

```
> db.employee.find({ "Project_id": { $in: [1, 2] } })
< {
    _id: ObjectId('67f12c506e7ee3d0db1b1b17'),
    Id: 2,
    Name: 'Frank Smith',
    Age: 36,
    Gender: 'Female',
    Project_id: 1,
    Hrs_worked: 12
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b18'),
    Id: 3,
    Name: 'Hannah Brown',
    Age: 33,
    Gender: 'Female',
    Project_id: 2,
    Hrs_worked: 36,
    Overtime: true
  }
  {
    _id: ObjectId('67f12c506e7ee3d0db1b1b19'),
    Id: 4,
    Name: 'Hannah Davis',
    Age: 31,
    Gender: 'Female',
    Project_id: 2,
    Hrs_worked: 22
  }
```

Journal write-up must include the objective and Answer to following
1. Explain the difference between insertOne and insertMany in MongoDB.

Soln. The **insertOne** and **insertMany** methods in MongoDB are both used to add documents to a collection, but they serve different purposes. **insertOne** is designed to insert a single document into a collection. It takes a single document as an argument and returns the result of the operation, including the inserted document's ID. On the other hand, **insertMany** is used to insert multiple documents at once. It accepts an array of documents and adds them to the collection in a single operation. This can be more efficient than inserting documents one at a time, especially when dealing with large datasets.

2. When would you use insertOne over insertMany, and vice versa?

Soln. You would use **insertOne** when you need to add a single document to a collection, such as when a new user registers or a new product is added. It is straightforward and efficient for single entries.
Conversely, **insertMany** is preferable when you have multiple documents to insert simultaneously, such as importing a batch of records from a file or when you need to add several related documents at once.
Using **insertMany** reduces the number of operations sent to the database, which can improve performance and reduce overhead.

3. Can you provide an example of using insertOne to insert a single document into a MongoDB collection?

Soln.

```
db.employees.insertOne({
    "Id": 21,
    "Name": "John Doe",
    "Age": 30,
    "Gender": "Male",
    "Project_id": 2,
    "Hrs_worked": 40
});
```

This command adds a new employee document to the employees collection with the specified fields.

4. How would you use insertMany to insert multiple documents into a MongoDB collection?

Soln.

```
db.employees.insertMany([
    { "Id": 22, "Name": "Jane Smith", "Age": 28, "Gender": "Female", "Project_id": 1,
"Hrs_worked": 35 },
    { "Id": 23, "Name": "Alice Johnson", "Age": 32, "Gender": "Female", "Project_id": 3,
"Hrs_worked": 42 }
]);
```

This command inserts two new employee documents into the employees collection in a single operation.

5. Describe the purpose of updateOne and updateMany in MongoDB.

Soln. The updateOne and updateMany methods are used to modify existing documents in a MongoDB collection. updateOne updates the first document that matches the specified filter criteria, allowing you to change specific fields within that document. In contrast, updateMany updates all documents that match the filter criteria, making it useful for bulk updates. Both methods utilize the $set operator to specify which fields to update and their new values.

6. What is the significance of the filter parameter in updateOne and updateMany?

Soln. The filter parameter in both **updateOne** and **updateMany** is crucial because it determines which documents will be affected by the update operation. Without a filter, all documents in the collection would be updated, which is usually not the desired outcome. The filter allows for precise targeting of documents based on specific criteria, such as matching a field value or a combination of conditions.

7. How can you use updateOne to update a specific field in a document?

Soln.

```
db.employees.updateOne(
    { "Id": 21 },
    { $set: { "Hrs_worked": 45 } }
);
```

8. Provide an example of using updateMany to update multiple documents based on a common condition.

Soln.

```
db.employees.updateMany(
    { "Hrs_worked": { $gt: 30 } },
    { $set: { "Overtime": true } }
);
```

9. Explain the purpose of sorting in MongoDB queries.

Soln. Sorting in MongoDB queries is used to organize the results returned from a query in a specific order. This can enhance the readability of the data and make it easier to analyze. For example, sorting employee records by age can help quickly identify the youngest or oldest employees. Sorting can be done in ascending or descending order based on one or more fields.

10. Explain the role of logical expressions and operators in MongoDB queries.

Soln. Logical expressions and operators, such as **$and**, **$or**, and **$not**, allow for the construction of complex queries in MongoDB. They enable users to combine multiple conditions to filter documents more effectively. For instance, using **$or** allows you to retrieve documents that meet at least one of several criteria, while **$and** ensures that all specified conditions must be true for a document to be included in the results.

11. Explain the purpose of indexing in MongoDB and how it improves query performance.

Soln. Indexing in MongoDB is a technique used to improve the speed of data retrieval operations on a collection. An index is a special data structure that maintains a sorted order of the indexed fields, allowing the database to quickly locate documents without scanning the entire collection. This significantly enhances query performance, especially for large datasets, as it reduces the amount of data that needs to be examined to fulfill a query.

12. Write a query to list all indexes on the employee collection.
Soln.

```
db.employees.getIndexes();
```

This command returns an array of documents, each representing an index on the collection, along with details such as the indexed fields and the index type.