What is MongoDB?

- MongoDB is a NoSQL database designed for scalability and flexibility.
- **Key Features:**
  - Document-oriented
  - Schema-less
  - Horizontal scalability

Performing CRUD Operations

- **CRUD Overview:**
  - **C**reate
  - **R**ead
  - **U**pdate
  - **D**elete

Creating Records

- **Example:** Inserting a document into a collection.

```
db.users.insertOne({
  name: "Alice",
  age: 30,
  email: "alice@example.com"
});
```

- This command creates a new user document.

Accessing Data

```
db.users.find({ age: { $gt: 25 } });
```

- This query fetches users older than 25.

: Updating and Deleting Data

- **Updating Example:**

```
db.users.updateOne(
  { name: "Alice" },
  { $set: { age: 31 } }
);
```

- **Deleting Example:**

```
db.users.deleteOne({ name: "Alice" });
```

- Update changes the age, and delete removes the document.

Working with Language Bindings

- **Overview:** MongoDB supports various programming languages (Python, Java, Node.js).
- **Example in Node.js:**

```
const MongoClient = require('mongodb').MongoClient;
// Connect and perform CRUD operations
```

Querying NoSQL Stores

- **NoSQL Queries:** Flexibility with data structures.
- **Example Query:**

```
db.products.find({ category: "Electronics" });
```

- **MongoDB Operations**:
  - "MongoDB Operations"
    - Insert: Add new documents to a collection.
    - Find: Retrieve documents based on criteria.
    - FindOne: Retrieve a single document.
    - Logical Operators: AND, OR, NOT for complex queries.
    - Distinct: Retrieve unique values.
    - Group: Aggregate documents.
    - Upsert: Update a document or insert if it doesn’t exist.
    - Update: Modify existing documents.
    - Remove: Delete documents from a collection.
- **Create Database**:
  - "Create Database"
    - MongoDB Query:

      ```
      use myDatabase
      ```
    - "This command switches to the specified database, creating it if it doesn’t exist."
- **Create Collection (Table) - Books**:
  - "Create Collection (Table) - Books"

      ```
      db.createCollection("books")
      ```
    - "This command creates a new collection named 'books' within the current database."
- **Find Restaurants with Score > 90**:
  - "Find Restaurants with Score > 90"

      ```
      db.restaurants.find({ "score": { $gt: 90 } })
      ```
    - "This query retrieves all restaurants where the score is greater than 90."
- **Find Restaurants without 'American' Cuisine and Score > 70**:
  - "Find Restaurants without 'American' Cuisine and Score > 70"

      ```
      db.restaurants.find({
        "cuisine": { $ne: "American" },
        "score": { $gt: 70 },
        "longitude": { $lt: -65.754168 }
      })
      ```

    - "This query finds restaurants that do not serve 'American' cuisine, have a score greater than 70, and are located west of -65.754168 longitude."

- **Slide 7 - Find Restaurants without 'American' Cuisine, Grade A, Not in Brooklyn**:

  ```
  db.restaurants.find({
    "cuisine": { $ne: "American" },
    "grade": "A",
    "borough": { $ne: "Brooklyn" }
  }).sort({ "cuisine": -1 })
  ```

    - "This query finds restaurants that do not serve 'American' cuisine, have a grade point of 'A', are not located in Brooklyn, and sorts the results by cuisine in descending order."

Summary:

- "Summary"
    - MongoDB provides powerful querying capabilities.
    - Logical operators and aggregation functions enhance data retrieval.
    - Understanding syntax and operations is essential for effective database management.

Create Database

Command to Create Database:

```
use myDatabase
```

This command switches to the specified database, creating it if it doesn’t exist.

Create Collections

Command to Create Collections:

```
db.createCollection("trains")
db.createCollection("patients")
db.createCollection("bookings")
```

Creates collections for railway, hospital, and travel data.

Insert Documents - Railway

Insert Sample Documents:

```
db.trains.insertOne({ "train_number": "A1", "passenger_count": 200,
"duration": 180 })
db.trains.insertOne({ "train_number": "B2", "passenger_count": 150,
"duration": 120 })
db.trains.insertOne({ "train_number": "C3", "passenger_count": 250,
"duration": 150 })
```

Adds train records to the `trains` collection.

Insert Documents - Hospital

Insert Sample Documents:

```
db.patients.insertOne({ "patient_id": "P001", "age": 30, "bill_amount":
1500 })
db.patients.insertOne({ "patient_id": "P002", "age": 45, "bill_amount":
2000 })
db.patients.insertOne({ "patient_id": "P003", "age": 25, "bill_amount": 500
})
```

Adds patient records to the `patients` collection.

Insert Documents - Travel

Insert Sample Documents:

```
db.bookings.insertOne({ "booking_id": "B001", "traveler_count": 3, "cost":
500 })
db.bookings.insertOne({ "booking_id": "B002", "traveler_count": 2, "cost":
300 })
db.bookings.insertOne({ "booking_id": "B003", "traveler_count": 5, "cost":
700 })
```

Adds booking records to the `bookings` collection.

Update Documents - Railway

Update a Document:

```
db.trains.updateOne(
  { "train_number": "A1" },
  { $set: { "passenger_count": 220 } }
)
```

Updates the passenger count for train A1.

Update Documents - Hospital

Update a Document:

```
db.patients.updateOne(
  { "patient_id": "P001" },
  { $set: { "bill_amount": 1600 } }
)
```

Updates the bill amount for patient P001.

Update Documents - Travel

Update a Document:

```
db.bookings.updateOne(
  { "booking_id": "B002" },
  { $set: { "cost": 350 } }
)
```

Updates the cost for booking B002.

Delete Documents - Railway

Delete a Document:

```
db.trains.deleteOne({ "train_number": "B2" })
```

Deletes train record B2.

Delete Documents - Hospital

Delete a Document:

```
db.patients.deleteOne({ "patient_id": "P003" })
```

Deletes patient record P003.

Delete Documents - Travel

Delete a Document:

```
db.bookings.deleteOne({ "booking_id": "B001" })
```

Deletes booking record B001.

1. **students** Collection:
   - Stores information about students such as name, student ID, and course ID.
2. **courses** Collection:
   - Stores information about courses such as course name, course ID, and instructor.

Step 1: Create the Collections and Insert Records

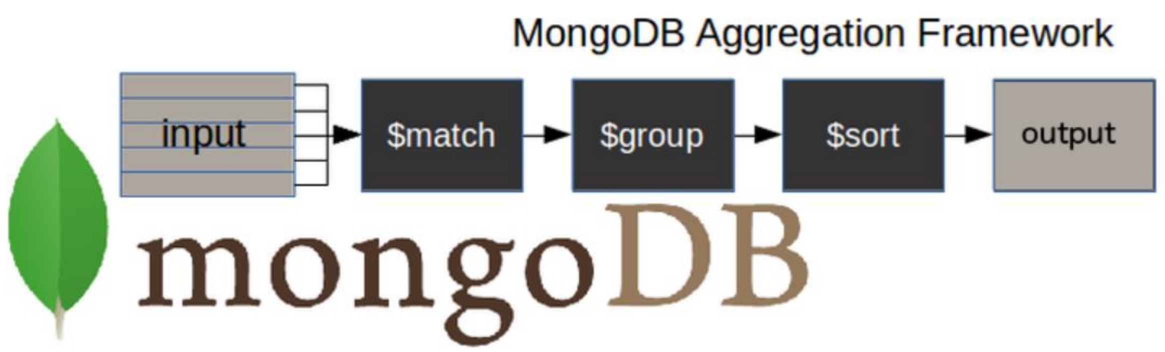
Create the `students` Collection and Insert Data

```
db.students.insertMany([
  { "_id": 1, "name": "John", "age": 20, "course_id": 101 },
  { "_id": 2, "name": "Jane", "age": 22, "course_id": 102 },
  { "_id": 3, "name": "Alice", "age": 24, "course_id": 101 },
  { "_id": 4, "name": "Bob", "age": 21, "course_id": 103 },
  { "_id": 5, "name": "Eve", "age": 23, "course_id": 104 }
])
```

Create the `courses` Collection and Insert Data

```
db.courses.insertMany([
  { "_id": 101, "course_name": "Mathematics", "instructor": "Dr. Smith" },
  { "_id": 102, "course_name": "Physics", "instructor": "Dr. Brown" },
  { "_id": 103, "course_name": "Chemistry", "instructor": "Dr. Johnson" },
  { "_id": 104, "course_name": "Biology", "instructor": "Dr. Davis" },
  { "_id": 105, "course_name": "History", "instructor": "Dr. Wilson" }
])
```

MongoDB aggregation pipeline.



Step 2: Perform Join Operations Using `$lookup`

Now, we will perform various join operations between `students` and `courses` using the `$lookup` operator.

Example 1: Basic Join

Join `students` with `courses` to display each student's course details.

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",           // The collection to join
      localField: "course_id",  // The field from the `students` collection
      foreignField: "_id",      // The field from the `courses` collection
      as: "course_info"         // The output field
    }
  }
])
```

Explanation:

- This basic join links the `students` collection's `course_id` with the `_id` field in the `courses` collection.
- The output will contain all student information, along with their course details in a field named `course_info`.

Sample Output:

```
{
  "_id": 1,
  "name": "John",
  "age": 20,
  "course_id": 101,
  "course_info": [
    {
      "_id": 101,
      "course_name": "Mathematics",
      "instructor": "Dr. Smith"
    }
  ]
}
```

```
db.courses.aggregate([
  {
    $lookup: {
      from: "students",// The collection to join
      localField: "_id",// The field from the `students` collection
      foreignField: "course_id", // The field from the `courses` collection
      as: "Student_info"         // The output field
    }
  }
])
```

Example 2: Join with Fields Filtering

Return only the course name and instructor in the joined result.

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "course_id",
      foreignField: "_id",
      as: "course_info"
    }
  },
  {
    $project: {   // Filter specific fields from the `course_info`
      "name": 1,
      "age": 1,
      "course_info.course_name": 1,
      "course_info.instructor": 1
    }
  }
])
```

Explanation:

- The `$project` stage allows us to specify which fields to display in the result. Here, only `course_name` and `instructor` from the `course_info` field are shown, along with the student's `name` and `age`.

Filter with Conditions

Find students who are enrolled in the "Physics" course.

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "course_id",
      foreignField: "_id",
      as: "course_info"
    }
  },
  {
    $match: { "course_info.course_name": "Physics" }
  }
])
```

Explanation:

- After the join, the `$match` stage filters the documents to show only students who are enrolled in "Physics."

Left Join with Non-matching Records

Show all students, including those who are not enrolled in any existing courses.

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "course_id",
      foreignField: "_id",
      as: "course_info"
    }
  },
  {
    $match: { "course_info": { $eq: [] } } // Matches students with no
course information
  }
])
```

Explanation:

- This query will return students whose `course_id` does not match any record in the `courses` collection, meaning they are not enrolled in a course.

Join and Unwind

Join and flatten the result by unwinding the array of course details.

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "course_id",
      foreignField: "_id",
      as: "course_info"
    }
  },
  {
    $unwind: "$course_info"  // Flatten the `course_info` array
  }
])
```

Explanation:

- The `$unwind` operator deconstructs the array (in this case, `course_info`) so that each student document will contain only one course.

Group Students by Course

Group students based on the courses they are enrolled in.

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "course_id",
      foreignField: "_id",
      as: "course_info"
    }
  },
  {
    $unwind: "$course_info"
  },
  {
    $group: {
      _id: "$course_info.course_name", // Group by course name
      students: { $push: "$name" }     // Collect students' names
    }
  }
])
```

Explanation:

- The `$group` operator groups students by course and aggregates their names into an array.

Join and Sort

Join and sort the students based on their course names.

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "course_id",
      foreignField: "_id",
      as: "course_info"
    }
  },
  {
    $unwind: "$course_info"
  },
  {
    $sort: { "course_info.course_name": 1 }  // Sort by course name
(ascending)
  }
])
```

Explanation:

- This query joins the two collections and sorts the students based on the course names in ascending order.

Join with Conditional Field Matching

Join students and courses where the instructor is "Dr. Smith."

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "course_id",
      foreignField: "_id",
      as: "course_info"
    }
  },
  {
    $match: { "course_info.instructor": "Dr. Smith" }
  }
])
```

Explanation:

- The `$match` operator filters the result to include only students whose course instructor is "Dr. Smith."

Count Students per Course

Find how many students are enrolled in each course.

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "course_id",
      foreignField: "_id",
      as: "course_info"
    }
  },
  {
    $unwind: "$course_info"
  },
  {
    $group: {
      _id: "$course_info.course_name",
      student_count: { $sum: 1 }
    }
  }
])
```

Explanation:

- The `$group` operator counts the number of students enrolled in each course by summing the total records per course.

Join with Multiple Fields

Join students with courses based on multiple fields.

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      let: { courseId: "$course_id" },
      pipeline: [
        { $match: { $expr: { $eq: ["$_id", "$$courseId"] } } },
        { $project: { course_name: 1, instructor: 1 } }
      ],
      as: "course_info"
    }
  }
])
```

Explanation:

- This query uses the `let` and `$expr` operators to match on multiple conditions (e.g., course ID and instructor). It returns only the `course_name` and `instructor` fields from the `courses` collection.

Similarities Between SQL and MongoDB

- **Comparative Features:**
  - Both use queries to manipulate data.
  - Filtering and sorting capabilities are similar.
- **Example:** SQL `SELECT` vs. MongoDB `find()`.

Accessing Data from Column-Oriented Databases (HBase)

- **Overview of HBase:** A NoSQL database that stores data in column families.
- **Data Access:** Uses similar principles to MongoDB for data retrieval.

Querying Redis Data Stores

- **Overview of Redis:** An in-memory data structure store.
- **Basic Commands:**

```
SET key value
GET key
```

- **Usage:** Fast access and caching.

MongoDB Internals

- **Architecture:** Document storage, collections, databases.
- **Storage Engine:** How data is stored and retrieved.

Essential Concepts behind a Database Index

- **Definition of Index:** A data structure that improves the speed of data retrieval operations.
- **Types of Indexes:**
  - Single field
  - Compound
  - Geospatial

Indexing and Ordering in MongoDB

- **Ordering:** How documents are sorted based on index.
- **Example:**

```
db.users.createIndex({ age: 1 });  // Ascending order
```

Creating and Using Indexes in MongoDB

- **Creating an Index:**

```
db.products.createIndex({ name: 1, price: -1 });
```

- **Using an Index:** Automatically used by queries to improve performance.

MongoDB Queries Overview

- **Create Operations:**

```
db.orders.insertOne({ item: "Book", qty: 1 });
```

- **Read Operations:**

```
db.orders.find({ item: "Book" });
```

SData Aggregation Operations

- **Aggregation Pipeline:**

```
db.sales.aggregate([
  { $match: { amount: { $gt: 100 } } },
  { $group: { _id: "$item", total: { $sum: "$amount" } } }
]);
```

- Filters and groups data to calculate totals.
