skillzam

www.skillzam.com

MongoDB Practice Exercises

Basic CRUD Operations

Insert a document into a collection.

- **Question:** Insert a document into the `users` collection.
- **Solution:**

```
db.users.insertOne({ name: "John", age: 30 });
```

Find all documents in a collection.

- **Question:** Retrieve all documents from the `users` collection.
- **Solution:**

```
db.users.find();
```

Find documents with specific criteria.

- **Question:** Retrieve users whose age is greater than 25.
- **Solution:**

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

Update a document.

- **Question:** Update the age of a user with name "John" to 31.
- **Solution:**

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

Delete a document.

- **Question:** Remove a user with the name "John" from the collection.
- **Solution:**

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

Insert multiple documents at once.

- **Question:** Insert multiple users into the collection.
- **Solution:**

```
db.users.insertMany([{ name: "Alice", age: 25 }, { name: "Bob", age: 32 }]);
```

Count the number of documents in a collection.

- **Question:** Get the count of users with age greater than 30.
- **Solution:**

```
db.users.countDocuments({ age: { $gt: 30 } });
```

Limit the number of results returned.

- **Question:** Retrieve only the first 3 users.
- **Solution:**

```
db.users.find().limit(3);
```

Sort documents by field.

- **Question:** Sort users by age in descending order.
- **Solution:**

```
db.users.find().sort({ age: -1 });
```

Distinct values in a field.

- **Question:** Get all distinct ages in the users collection.
- **Solution:**

```
db.users.distinct("age");
```

Query Operators

$eq (Equality)

Question: Find all users with age equal to 30.

Solution:

```
db.users.find({ age: { $eq: 30 } });
```

$gt (Greater than)

Question: Find all users older than 25.

Solution:

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

$gte (Greater than or equal to)

Question: Find all users aged 18 or older.

Solution:

```
db.users.find({ age: { $gte: 18 } });
```

$lt (Less than)

Question: Find all users younger than 30.

Solution:

```
db.users.find({ age: { $lt: 30 } });
```

$lte (Less than or equal to)

Question: Find all users aged 25 or younger.

Solution:

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

$ne (Not equal)

Question: Find all users whose name is not "Alice".

Solution:

```
db.users.find({ name: { $ne: "Alice" } });
```

$in (Value is in an array)

Question: Find all users whose age is either 25, 30, or 35.

Solution:

```
db.users.find({ age: { $in: [25, 30, 35] } });
```

$nin (Value is not in an array)

Question: Find all users whose age is not 25, 30, or 35.

Solution:

```
db.users.find({ age: { $nin: [25, 30, 35] } });
```

$or (Logical OR)

Question: Find all users whose age is less than 20 or greater than 60.

Solution:

```
db.users.find({ $or: [{ age: { $lt: 20 } }, { age: { $gt: 60 } }] });
```

$and (Logical AND)

Question: Find all users whose age is greater than 20 and less than 40.

Solution:

```
db.users.find({ $and: [{ age: { $gt: 20 } }, { age: { $lt: 40 } }] });
```

$exists (Field existence check)

Question: Find all users who have a `phone` field.

Solution:

```
db.users.find({ phone: { $exists: true } });
```

$type (Field data type check)

Question: Find all users whose `age` field is an integer.

Solution:

```
db.users.find({ age: { $type: "int" } });
```

$regex (Regular expression search)

Question: Find all users whose name starts with "A".

Solution:

```
db.users.find({ name: { $regex: "^A" } });
```

$text (Full-text search)

Question: Find all products containing the word "laptop" in their description (Assuming a text index exists on `description`).

Solution:

```
db.products.find({ $text: { $search: "laptop" } });
```

Update Operators

$set (Update fields)

Question: Update the name of the user with `userId` to "John Doe".

Solution:

```
db.users.updateOne({ _id: ObjectId("userId") }, { $set: { name: "John Doe" } });
```

$unset (Remove a field)

Question: Remove the `phone` field from a user document.

Solution:

```
db.users.updateOne({ name: "John" }, { $unset: { phone: "" } });
```

$inc (Increment a field by a specified value)

Question: Increment the age of a user named "Alice" by 1.

Solution:

```
db.users.updateOne({ name: "Alice" }, { $inc: { age: 1 } });
```

$mul (Multiply a field by a specified value)

Question: Multiply the `salary` of a user by 1.1 (i.e., increase by 10%).

Solution:

```
db.users.updateOne({ name: "John" }, { $mul: { salary: 1.1 } });
```

$push (Add an element to an array)

Question: Add a new `hobby` to the `hobbies` array for the user "John".

Solution:

```
db.users.updateOne({ name: "John" }, { $push: { hobbies: "reading" } });
```

$addToSet (Add a unique element to an array)

Question: Add a new `hobby` to the `hobbies` array, ensuring no duplicates for the user "John".

Solution:

```
db.users.updateOne({ name: "John" }, { $addToSet: { hobbies: "swimming" } });
```

$pull (Remove an element from an array)

Question: Remove "swimming" from the `hobbies` array of the user "John".

Solution:

```
db.users.updateOne({ name: "John" }, { $pull: { hobbies: "swimming" } });
```

$pop (Remove the first or last element from an array)

Question: Remove the last element from the `hobbies` array of the user "John".

Solution:

```
db.users.updateOne({ name: "John" }, { $pop: { hobbies: 1 } });
```

$shift (Remove the first or last element from an array)

Question: Remove the first element from the `hobbies` array of the user "John".

Solution:

```
db.users.updateOne({ name: "John" }, { $pop: { hobbies: -1 } });
```

Advanced Queries & Aggregation

Use of `$match` in aggregation.

- **Question:** Find users older than 30 using the aggregation pipeline.
- **Solution:**

```
db.users.aggregate([
  { $match: { age: { $gt: 30 } } }
]);
```

Group documents with aggregation.

- **Question:** Group users by age and count the number of users in each age group.
- **Solution:**

```
db.users.aggregate([
  { $group: { _id: "$age", count: { $sum: 1 } } }
]);
```

Sort and limit in aggregation.

- **Question:** Get the top 3 oldest users.
- **Solution:**

```
db.users.aggregate([
  { $sort: { age: -1 } },
  { $limit: 3 }
]);
```

Use `$project` to include or exclude fields.

- **Question:** Get a list of users with only their name and age.
- **Solution:**

```
db.users.aggregate([
  { $project: { name: 1, age: 1, _id: 0 } }
]);
```

Join collections with `$lookup`.

- **Question:** Retrieve orders and their corresponding user details.
- **Solution:**

```
db.orders.aggregate([
  { $lookup: { from: "users", localField: "userId", foreignField: "_id", as: "userDet
]);
```

Add computed fields with `$addFields`.

- **Question:** Add a field `isAdult` that checks if the user is 18 or older.
- **Solution:**

```
db.users.aggregate([
  { $addFields: { isAdult: { $gte: ["$age", 18] } } }
]);
```

Use `$unwind` to deconstruct arrays.

- **Question:** Unwind an array of products in an order.
- **Solution:**

```
db.orders.aggregate([
  { $unwind: "$products" }
]);
```

Use `$facet` for multiple aggregation pipelines.

- **Question:** Run two different pipelines: one for counting users and one for calculating the average age.
- **Solution:**

```
db.users.aggregate([
  {
    $facet: {
      count: [{ $count: "totalUsers" }],
      avgAge: [{ $group: { _id: null, averageAge: { $avg: "$age" } } }]
    }
  }
]);
```

Limit results in aggregation after multiple stages.

- **Question:** After grouping users by age, limit to the first 5 groups.
- **Solution:**

```
db.users.aggregate([
  { $group: { _id: "$age", count: { $sum: 1 } } },
  { $limit: 5 }
]);
```

Using `$merge` to store aggregation results.

- **Question:** Store the aggregation result into a new collection `age_groups`.
- **Solution:**

```
db.users.aggregate([
  { $group: { _id: "$age", count: { $sum: 1 } } },
  { $merge: { into: "age_groups" } }
]);
```

Indexing and Performance

Create an index.

- **Question:** Create an index on the `name` field in the `users` collection.
- **Solution:**

```
db.users.createIndex({ name: 1 });
```

Create a compound index.

- **Question:** Create an index on both `age` and `name` fields.
- **Solution:**

```
db.users.createIndex({ age: 1, name: 1 });
```

Drop an index.

- **Question:** Drop the index on the `name` field.
- **Solution:**

```
db.users.dropIndex("name_1");
```

List all indexes on a collection.

- **Question:** List all indexes for the `users` collection.
- **Solution:**

```
db.users.getIndexes();
```

Use of `$explain` to analyze query performance.

- **Question:** Explain the query to retrieve users older than 30.
- **Solution:**

```
db.users.find({ age: { $gt: 30 } }).explain("executionStats");
```

Use of covered queries.

- **Question:** Create a query that only retrieves indexed fields to be a covered query.
- **Solution:**

```
db.users.createIndex({ age: 1 });
db.users.find({ age: 25 }).explain("executionStats");
```

Use `$text` index for full-text search.

- **Question:** Create a text index on the `description` field and search for a term.
- **Solution:**

```
db.products.createIndex({ description: "text" });
db.products.find({ $text: { $search: "electronics" } });
```

Use of `$geoNear` for geospatial queries.

- **Question:** Find stores near a given location.
- **Solution:**

```
db.stores.aggregate([
  {
    $geoNear: {
      near: { type: "Point", coordinates: [-73.97, 40.77] },
      distanceField: "dist.calculated",
      maxDistance: 5000,
      spherical: true
    }
  }
]);
```

Ensure index uniqueness.

- **Question:** Create a unique index on the `email` field.
- **Solution:**

```
db.users.createIndex({ email: 1 }, { unique: true });
```

Use `$hint` to force an index on a query.

- **Question:** Force MongoDB to use a specific index for a query.
- **Solution:**

```
db.users.find({ age: { $gt: 30 } }).hint({ age: 1 });
```

Aggregation Operations and Querying

Aggregation with `$sum`.

- **Question:** Calculate the total value of all orders.
- **Solution:**

```
db.orders.aggregate([
  { $group: { _id: null, totalValue: { $sum: "$value" } } }
]);
```

Using `$avg` in aggregation.

- **Question:** Calculate the average age of users.
- **Solution:**

```
db.users.aggregate([
  { $group: { _id: null, averageAge: { $avg: "$age" } } }
]);
```

Using `$min` and `$max`.

- **Question:** Find the youngest and oldest user.
- **Solution:**

```
db.users.aggregate([
  { $group: { _id: null, youngest: { $min: "$age" }, oldest: { $max: "$age" } } }
]);
```

$addToSet to remove duplicates.

- **Question:** Find the distinct list of product categories from orders.
- **Solution:**

```
db.orders.aggregate([
  { $group: { _id: null, uniqueCategories: { $addToSet: "$category" } } }
]);
```

$arrayElemAt to access array elements.

- **Question:** Get the first product from the `products` array in the orders.
- **Solution:**

```
db.orders.aggregate([
  { $project: { firstProduct: { $arrayElemAt: ["$products", 0] } } }
]);
```

Data Modeling

Embed documents in a collection.

- **Question:** Embed user addresses within a user document.
- **Solution:**

```
db.users.insertOne({ name: "John", addresses: [{ street: "123 Elm", city: "NY" }] });
```

Reference documents in a collection.

- **Question:** Use `userId` in orders to reference users.
- **Solution:**

```
db.orders.insertOne({ userId: ObjectId("user_id_here"), value: 100 });
```

Data normalization using references.

- **Question:** Normalize product data by storing product IDs in orders.
- **Solution:**

```
db.orders.insertOne({ productIds: [ObjectId("product_id_here")] });
```

---

skillzam

CONTACT US

80500 80399

EMAIL

apply@skillzam.com

WEBSITE

www.skillzam.com

OFFICE ADDRESS

1st Floor, Pearl Plaza,
Plot# 271, Shivbasava Nagar,
Nehru Nagar, Belagavi,
Karnataka 590016

Landmark: Above Reliance smart point

Google Map