Priyanshu Dhyani

Mongo DB Concepts and Queries

Introduction to MongoDB

MongoDB is a NoSQL database designed for scalability, flexibility, and performance. It stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.

Document: A record in MongoDB, stored in BSON format (Binary JSON).

Collection: A group of MongoDB documents, akin to a table in relational databases.

Database: A container for collections.

Data Modeling

Data modeling in MongoDB involves defining the structure of your documents and collections.

Schema Design

Embedded Documents: Store related data within a single document. This is useful for data that is typically accessed together.

```
{
  "_id": 1,
  "name": "Alice",
  "address": {
    "street": "123 Main St",
    "city": "Springfield",
    "state": "IL"
  }
}
```

References: Use references for relationships between documents, useful for data accessed separately or to avoid duplication.

```
{
  "_id": 1,
  "name": "Alice",
  "address_id": 123
}
```

Priyanshu Dhyani

```
{
  "_id": 123,
  "street": "123 Main St",
  "city": "Springfield",
  "state": "IL"
}
```

Design Considerations

Data Access Patterns: Optimize schema based on how your application queries and updates data.

Write vs. Read Performance: Balance between fast writes (embedded) and efficient reads (references).

Data Consistency: Embedded documents ensure consistency but can lead to larger document sizes.

CRUD Operations

CRUD operations in MongoDB allow you to create, read, update, and delete documents.

Create

Insert One:

```
db.collection('users').insertOne({ name: 'Alice', age: 25 });
```

Insert many:

```
db.collection('users').insertMany([{ name: 'Bob', age: 30 }, { name: 'Charlie', age: 35 }]);
```

Read

Find One:

```
db.collection('users').findOne({ name: 'Alice' });
```

Find many:

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

Update

Update one:

```
db.collection('users').updateOne({ name: 'Alice' }, { $set: { age: 26 } });
```

Update many:

```
db.collection('users').updateMany({ age: { $gt: 25 } }, { $set: { status: 'active' } });
```

Priyanshu Dhyani

Delete

Delete one:

db.collection('users').deleteOne({ name: 'Alice' });

Delete many:

db.collection('users').deleteMany({ status: 'inactive' });

Indexing

Indexes support efficient query execution. Without indexes, MongoDB must scan every document in a collection to select those that match the query.

Creating Indexes

Single Field Index:

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

Compound Index:

db.collection('users').createIndex({ name: 1, age: -1 });

Index Types

Unique Index:

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

TTL index:

db.collection('sessions').createIndex({ createdAt: 1 }, { expireAfterSeconds: 3600 });

Index Optimization

- Analyze and ensure indexes align with query patterns.
- Use compound indexes to support multiple fields in queries.
- Limit the number of indexes to balance write performance.
-

Aggregation Framework

The aggregation framework processes data records and returns computed results. It uses a pipeline approach to transform documents.

Pipeline Stages

$match: Filters documents.

{ $match: { status: 'active' } }

Priyanshu Dhyani

$group: Groups documents by a specified expression and outputs to the next stage a document for each group.

```
{ $group: { _id: '$age', count: { $sum: 1 } } }
```

$project: Reshapes documents by including, excluding, or adding new fields.

```
{ $project: { name: 1, year: { $year: '$createdAt' } } }
```

$sort: Sorts documents.

```
{ $sort: { count: -1 } }
```

Example:

```
db.collection('orders').aggregate([
  { $match: { status: 'completed' } },
  { $group: { _id: '$customerId', totalAmount: { $sum: '$amount' } } },
  { $sort: { totalAmount: -1 } }
]);
```

Query Optimization

Optimizing queries involves using indexes effectively and structuring queries to minimize the amount of data processed.

Strategies

Use Projections: Retrieve only necessary fields.

```
db.collection('users').find({}, { name: 1, age: 1 });
```

Indexing: Ensure queries use indexes.

```
db.collection('users').find({ name: 'Alice' }).explain('executionStats');
```

Limit and Skip: Reduce the amount of data returned.

```
db.collection('users').find().limit(10).skip(20);
```

Analyzing Performance

Explain Method: Provides information on how MongoDB executes a query.

```
db.collection('users').find({ name: 'Alice' }).explain('executionStats');
```

Priyanshu Dhyani

Profiling: Monitor the performance of operations.

```
db.setProfilingLevel(2);
db.system.profile.find().sort({ ts: -1 });
```

Common Use Cases

MongoDB is versatile and used across various industries and applications.

Use Cases

- **Content Management Systems:** Flexible schema supports varied content types.
- **Real-time Analytics:** Efficiently handles large volumes of streaming data.
- **IoT Applications:** Store and process data from numerous devices.
- **E-commerce:** Manage product catalogs, customer data, and order transactions.

Best Practices

Data Modeling Best Practices

- **Embed vs Reference:** Embed for one-to-few relationships and reference for one-to-many relationships.
- **Schema Design:** Design schema according to application requirements and query patterns.
- **Document Size:** Keep documents under 16MB, MongoDB's document size limit.

Indexing Best Practices

- **Create Indexes for Frequent Queries:** Ensure queries that run frequently have appropriate indexes.
- **Limit Indexes:** Avoid over-indexing as it impacts write performance.
- **Use Compound Indexes:** Combine fields in a single index for complex queries.

Operational Best Practices

- **Backup and Restore:** Regularly backup your database and test restore processes.
- **Sharding:** Distribute data across multiple servers for horizontal scaling.
- **Monitoring:** Use monitoring tools to track performance and set alerts for potential issues.

Security Best Practices

- **Authentication:** Enable access control and create users with specific roles.
- **Encryption:** Encrypt data at rest and in transit.
- **Network Security:** Use firewalls and VPNs to secure access to your MongoDB instances.