MongoDB Indexes - Complete Guide with Examples

Indexes in MongoDB improve query performance by allowing the database to quickly locate documents instead of scanning the entire collection.

Step 1: Creating a Sample Collection

Let's create a collection called `employees` and insert sample data.

Insert Sample Data

```
db.employees.insertMany([
  { _id: 1, name: "Alice", department: "HR", salary: 50000, joinDate: ISODate("2020-01-15") },
  { _id: 2, name: "Bob", department: "IT", salary: 75000, joinDate: ISODate("2019-07-23") },
  { _id: 3, name: "Charlie", department: "Finance", salary: 62000, joinDate: ISODate("2021-06-12") },
  { _id: 4, name: "David", department: "IT", salary: 80000, joinDate: ISODate("2018-09-30") },
  { _id: 5, name: "Emma", department: "HR", salary: 52000, joinDate: ISODate("2019-12-05") }
]);
```

Step 2: Understanding Indexes

Indexes store a small portion of the collection's data in an optimized structure, allowing faster searches.

Types of Indexes in MongoDB

1. **Default Index (_id)**
2. **Single Field Index**
3. **Compound Index**
4. **Multikey Index**
5. **Text Index**
6. **Hashed Index**
7. **Wildcard Index**
8. **Unique Index**
9. **Partial Index**
10. **TTL (Time-To-Live) Index**

Step 3: Creating and Using Indexes

1. Default `_id` Index

Every MongoDB collection has a unique index on the `_id` field by default.

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

Output:

```
[
  { "v": 2, "key": { "_id": 1 }, "name": "_id_" }
]
```

🔹 `_id` is indexed automatically, ensuring uniqueness.

---

2. Single Field Index

Creates an index on a single field to speed up queries.

Example: Create an index on the `name` field

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

- `1` means **ascending order** (for sorting).
- `-1` means **descending order**.

Query Before Index

```
db.employees.find({ name: "Alice" }).explain("executionStats");
```

🔹 Before indexing, MongoDB **scans all documents** (COLLSCAN).

Query After Index

```
db.employees.find({ name: "Alice" }).explain("executionStats");
```

🔹 After indexing, MongoDB uses **IXSCAN** (Index Scan), making the query faster.

---

3. Compound Index

Indexes multiple fields to optimize queries using those fields.

Example: Index `department` and `salary`

```
db.employees.createIndex({ department: 1, salary: -1 });
```

🔹 Useful for queries involving both `department` and `salary`.

Query Using Compound Index

```
db.employees.find({ department: "IT", salary: { $gt: 70000 }
}).explain("executionStats");
```

---

4. Multikey Index

Used when indexing **array fields**.

Example: Add Skills Field

```
db.employees.updateMany({}, { $set: { skills: ["MongoDB", "Python", "Excel"] } });
db.employees.createIndex({ skills: 1 });
```

🔹 Now, searching within an **array** is fast.

Query

```
db.employees.find({ skills: "Python" }).explain("executionStats");
```

---

5. Text Index

Used for **full-text search**.

Example: Create Text Index on `name`

```
db.employees.createIndex({ name: "text" });
```

🔹 Enables searching using `text` queries.

Search Example

```
db.employees.find({ $text: { $search: "Alice" } });
```

---

6. Hashed Index

Used for **sharding** and **equality queries**.

Example: Create Hashed Index on `department`

```
db.employees.createIndex({ department: "hashed" });
```

🔹 Optimizes equality searches like:

```
db.employees.find({ department: "IT" }).explain("executionStats");
```

---

7. Wildcard Index

Indexes **all fields dynamically**.

Example

```
db.employees.createIndex({ "$**": 1 });
```

🔹 Useful for dynamic fields.

---

8. Unique Index

Ensures **no duplicate values**.

Example: Unique Index on `name`

```
db.employees.createIndex({ name: 1 }, { unique: true });
```

🔹 Prevents duplicate names.

---

9. Partial Index

Indexes **only documents that meet a condition**.

Example: Index Only High Salary Employees

```
db.employees.createIndex({ salary: 1 }, { partialFilterExpression: { salary: { $gt: 60000 } } });
```

🔹 Indexes only employees with `salary > 60000`.

---

10. TTL (Time-To-Live) Index

Automatically deletes documents after a specified time.

Example: Expire Documents After 10 Seconds

```
db.employees.createIndex({ joinDate: 1 }, { expireAfterSeconds: 10 });
```

🔹 Deletes documents **10 seconds** after `joinDate`.

Step 4: Viewing and Deleting Indexes

View All Indexes

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

Drop an Index

```
db.employees.dropIndex({ name: 1 });
```

Drop All Indexes

```
db.employees.dropIndexes();
```

Step 5: Performance Testing

Run queries with `.explain("executionStats")` to see index usage.

```
db.employees.find({ name: "Alice" }).explain("executionStats");
```

Sumary

✓ Indexes Improve Performance
✓ Use the Right Index for Each Query
✓ Too Many Indexes Can Slow Down Writes