Indexes in MongoDB: Single Field, Compound, TTL, and Text Indexes

1. Problem Statement

Efficient data retrieval is critical for database performance, especially as collections grow. Without indexes, MongoDB must scan every document to answer queries, resulting in slow performance. Indexes are essential for optimizing query speed, sorting, and even data lifecycle management.

The challenge:
How do you use different types of indexes in MongoDB—single field, compound, TTL, and text indexes—to optimize queries, support advanced search, and automate data expiration?

2. Learning Objectives

By the end of this lesson, you will be able to:

- Explain the purpose and mechanics of indexes in MongoDB.
- Create and use single field, compound, TTL, and text indexes.
- Understand when to use each index type for optimal performance.
- Apply index properties and options for advanced use cases.
- Use indexes to support efficient queries, sorting, and automatic document expiration.

3. Concept Introduction with Analogy

Analogy: Library Card Catalogs

- **Single Field Index:** Like organizing library cards alphabetically by author—quickly find all books by a specific author.
- **Compound Index:** Like organizing cards by author, then by publication year—find all books by an author published in a certain year.
- **TTL Index:** Like setting a timer to automatically remove old magazines from the shelf.
- **Text Index:** Like a full-text search in a digital library—find all books mentioning a keyword in their description.

4. Technical Deep Dive

Single Field Indexes

- Indexes a single field in a collection.
- Created using `db.collection.createIndex({ field: 1 })` (1 for ascending, -1 for descending).
- Supports efficient queries, sorting, and range operations on that field.
- Can be created on top-level, embedded, or array fields.

Example:

```
db.books.createIndex({ title: 1 }) // Index on the 'title' field
```

Compound Indexes

- Indexes multiple fields together as a compound key.
- Created using `db.collection.createIndex({ field1: 1, field2: -1 })`.
- Supports queries and sorting on combinations of the indexed fields, especially those matching the index prefix.
- Order of fields is important: the index supports queries on the prefix fields.

Example:

```
db.students.createIndex({ name: 1, gpa: -1 })
```

TTL (Time-To-Live) Indexes

- Special single-field index that automatically removes documents after a specified time.
- Use for expiring data like sessions, logs, or temporary records.
- Created with `expireAfterSeconds` option.

Example:

```
db.sessions.createIndex({ createdAt: 1 }, { expireAfterSeconds: 86400 }) // Expires after 24 hours
```

- TTL indexes only work on date fields and cannot be created on the `_id` field.
- Can be partial (expire only documents matching a filter).
- Deletion is handled by a background process and may not be immediate.

Text Indexes

- Support text search queries on string fields.
- Created using `db.collection.createIndex({ field: "text" })`.
- Only one text index per collection, but it can cover multiple fields.
- Enables `$text` queries for searching words or phrases.

Example:

```
db.blog.createIndex({ content: "text", about: "text" })
db.blog.find({ $text: { $search: "coffee" } })
```

---

5. Step-by-Step Data Modeling & Code Walkthrough

1. Create a Single Field Index

```
db.books.createIndex({ author: 1 })
```

2. Create a Compound Index

```
db.movies.createIndex({ title: 1, year: 1 })
```

3. Create a TTL Index

```
db.eventlog.createIndex({ lastModifiedDate: 1 }, { expireAfterSeconds: 3600 })
```

4. Create a Text Index

```
db.stores.createIndex({ name: "text", description: "text" })
```

5. View All Indexes

```
db.books.getIndexes()
```

6. Drop an Index

```
db.books.dropIndex({ author: 1 })
```

---

6. Interactive Challenge / Mini-Project

Your Turn!

- Create a `users` collection and add a single field index on `email`.
- Create a compound index on `firstName` and `lastName` in the `users` collection.
- Add a TTL index to a `logs` collection to expire documents after 7 days.
- Create a text index on the `description` field of a `products` collection and perform a text search for the word "wireless".
- View all indexes in the `products` collection and drop a specific index.

---

7. Common Pitfalls & Best Practices

- **Don't over-index:** Each index adds storage overhead and slows down writes.
- **Choose fields wisely:** Index fields that are frequently queried, sorted, or filtered.
- **Compound index order matters:** The index supports queries on the prefix fields.
- **TTL indexes are not instant:** Deletion may be delayed; don't rely on immediate removal.
- **Text indexes are RAM-intensive:** Use with care on large collections.
- **Monitor index usage:** Regularly review which indexes are used and drop unused ones.

---

8. Quick Recap & Key Takeaways

- Indexes dramatically improve query performance in MongoDB.
- Single field indexes are simple and effective for common queries.
- Compound indexes optimize queries involving multiple fields.
- TTL indexes automate document expiration for time-sensitive data.
- Text indexes enable efficient full-text search on string fields.
- Use indexes thoughtfully to balance performance, storage, and write speed.

Practice Coding Challenges: MongoDB Indexes

---

1. Create a Single Field Index

Task:
Create a single field index on the `username` field in a `users` collection. Query users by username and observe the performance.

---

2. Create a Compound Index

Task:
Create a compound index on the `category` and `price` fields in a `products` collection. Write queries that use both fields.

---

3. Add a TTL Index

Task:
Add a TTL index to a `sessions` collection to expire documents 30 minutes after the `lastActive` field.

---

4. Create and Use a Text Index

Task:
Create a text index on the `description` and `tags` fields in a `blog` collection. Search for posts containing the word "mongodb".

---

5. Drop and View Indexes

Task:
Drop an index from a collection and use `getIndexes()` to confirm the change.

---

6. Index Embedded Fields

Task:
Create an index on the `address.city` field in a `customers` collection and query customers by city.

---

7. Create a Partial TTL Index

Task:
Create a TTL index on the `logs` collection that only expires documents where `level` is `"debug"`.

---

8. Compound Index Prefix

Task:
Create a compound index on `author`, `publishedYear`, and `genre` in a `books` collection. Write queries that use different prefixes of the index.

---

9. Text Search with Relevance

Task:
Perform a text search in a collection and sort results by relevance using the `$meta: "textScore"` projection.

---

10. Unique Index

Task:
Create a unique index on the `email` field in a `users` collection to prevent duplicate registrations.
