Course code : **CSE3009**
Course title : **No SQL Data Bases**
Module : **4**
Topic : **3**

Indexing

Objectives

This session will give the knowledge about

- Indexing

Indexing

Indexes allow efficient execution of MongoDB queries.

If we don’t have indexes, MongoDB has to scan all the documents in the collection to select those documents that match the criteria.

If proper indexing is used, MongoDB can limit the scanning of documents and select documents efficiently.

Indexes are a special data structure that store some field values of documents in an easy-to-traverse way.

Indexing

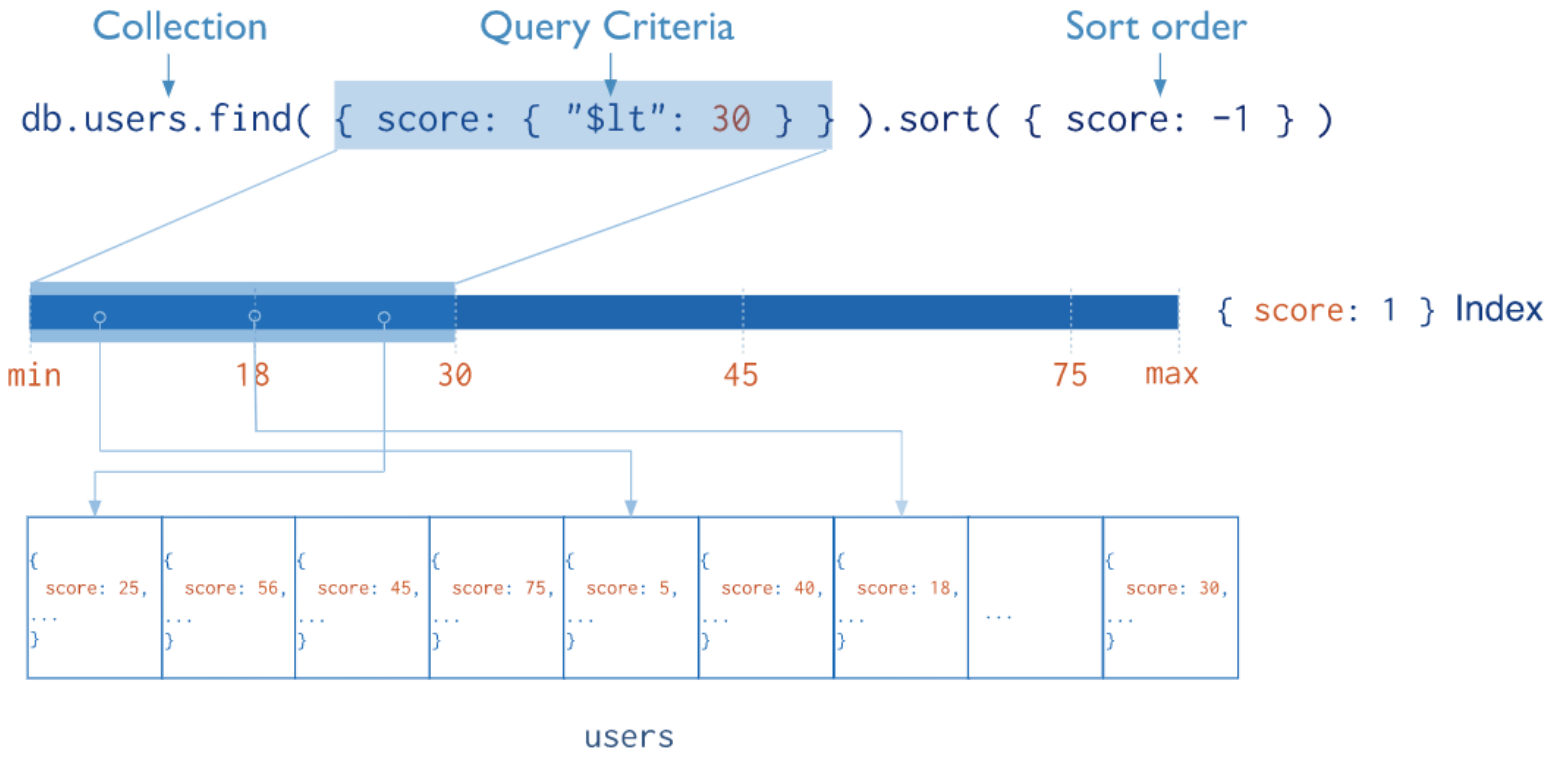
Indexes store the values of specific fields or sets of fields, ordered by the values of fields.

The ordering of field values allows us to apply effective algorithms of traversing, such as the mid-search algorithm, and also supports range-based operations effectively. In addition, MongoDB can return sorted results easily.

Indexes in MongoDB are the same as indexes in other database systems. MongoDB defines indexes at the collection level and supports indexes on fields and sub-fields of documents.

Indexing

The following diagram illustrates a query that selects and orders the matching documents using an index.



Indexing

The default _id index

- MongoDB creates the default _id index when creating a document.
- The _id index prevents users from inserting two documents with the same _id value. You cannot drop an index on an _id field.

The following syntax is used to create an index in MongoDB:

```
>db.collection.createIndex(, );
```

- The preceding method creates an index only if an index with the same specification does not exist.
- MongoDB indexes use the B-tree data structure.

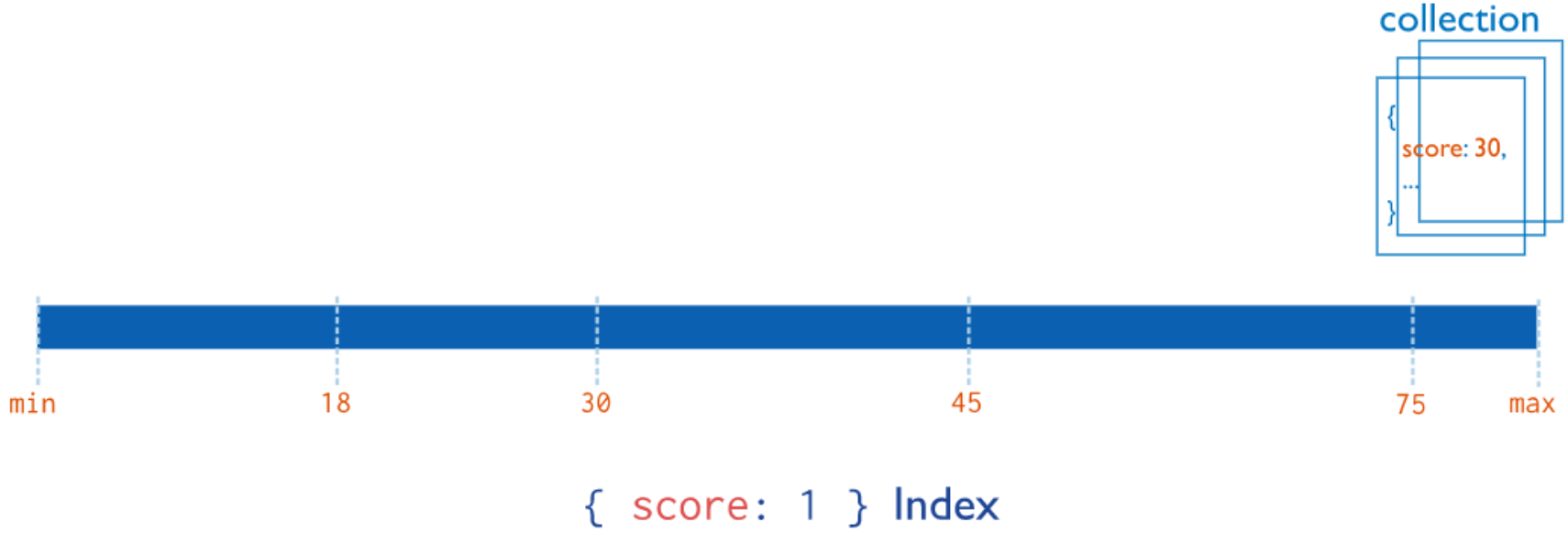
Type of Indexing

Single field

- In addition to the _id field index, MongoDB allows the creation of an index on any single field in ascending or descending order.
- For a single field index, the order of the index does not matter as MongoDB can traverse indexes in any order.
- The following is an example of creating an index on the single field where we are creating an index on the firstName field of the user_profiles collection:

Type of Indexing

For a single-field index and sort operations, the sort order (i.e. ascending or descending) of the index key does not matter because MongoDB can traverse the index in either direction.



Type of Indexing

```
db.user_profiles.createIndex({  "firstName" : 1 });
```

The query gives acknowledgment after creating the index:

```
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1.0
}
```

This will create an ascending index on the firstName field. To create a descending index, we have to provide -1 instead of 1.

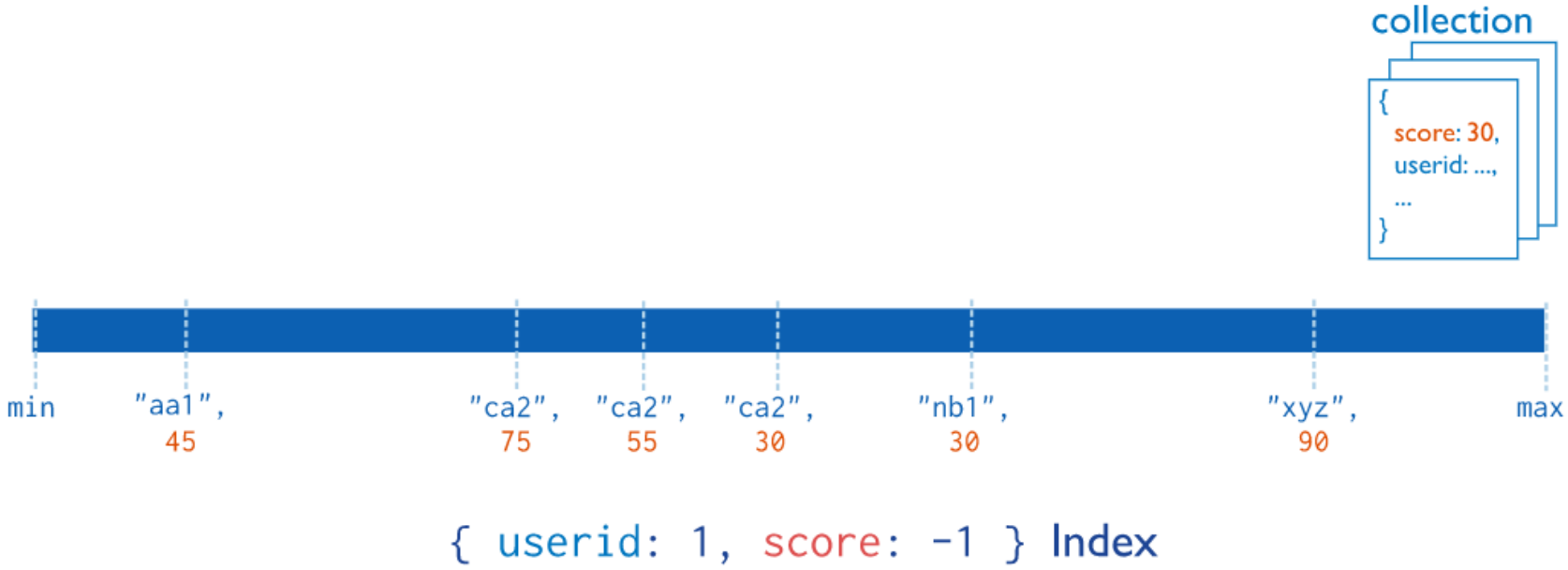
Type of Indexing

Compound index:

- MongoDB also supports user-defined indexes on multiple fields.
- The order of fields defined while creating an index has a significant effect.
- For instance, if a compound index consists of { userid: 1, score: -1 }, the index sorts first by userid and then, within each userid value, sorts by score.
- Compound indexes can be used with text indexes to define an ascending or descending order of the index.

Indexing

For compound indexes and sort operations, the sort order (i.e. ascending or descending) of the index keys can determine whether the index can support a sort operation. See Sort Order for more information on the impact of index order on results in compound indexes.

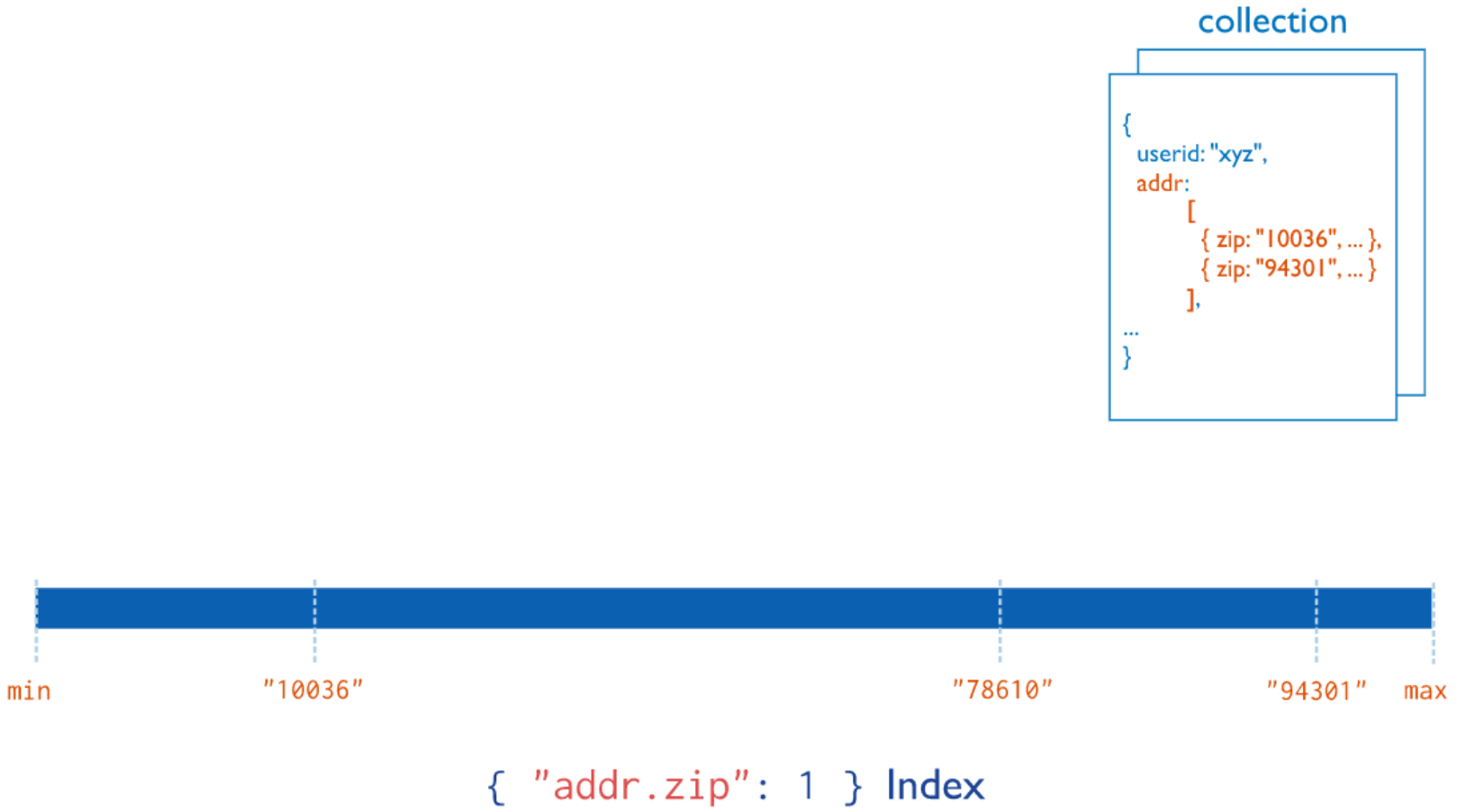


Type of Indexing

Multikey index:

- MongoDB uses multi-key indexes to index the content in the array.
- If you index the field that contains the array values, MongoDB creates an index for each field in the object of an array.
- These indexes allow queries to select the document by matching the element or set of elements of the array.
- MongoDB automatically decides whether to create multi-key indexes or not.

Type of Indexing



Type of Indexing

Text indexes:

- MongoDB provides text indexes that support the searching of string contents in the MongoDB collection.
- To create text indexes, we have to use the db.collection.createIndex() method, but we need to pass a text string literal in the query:

```
db.reviews.createIndex({"comments" : "text"})
```

Type of Indexing

You can also create text indexes on multiple fields, for example:

```
db.reviews.createIndex({
        "comments" : "text",
        "subject" : "text"
});
```

Once the index is created, we get an acknowledgment:

```
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1.0
}
```

Type of Indexing

Hashed index:

- To support hash-based sharding, MongoDB supports hashed indexes.
- In this approach, indexes store the hash value and query, and the select operation checks the hashed indexes.
- Hashed indexes can support only equality-based operations.
- They are limited in their performance of range-based operations.

Indexing Properties

Unique indexes:

- Indexes should maintain uniqueness. This makes MongoDB drop the duplicate value from indexes.

Partial Indexes:

- Partial indexes apply the index on documents of a collection that match a specified condition.
- By applying an index on the subset of documents in the collection, partial indexes have a lower storage requirement as well as a reduced performance cost.

Indexing Properties

Sparse index:

- In the sparse index, MongoDB includes only those documents in the index in which the index field is present, other documents are discarded.
- We can combine unique indexes with a sparse index to reject documents that have duplicate values but ignore documents that have an indexed key.

Indexing Properties

TTL index:

- TTL indexes are a special type of indexes where MongoDB will automatically remove the document from the collection after a certain amount of time.
- Such indexes are ideal to remove machine-generated data, logs, and session information that we need for a finite duration.
- The following TTL index will automatically delete data from the log table after 3000 seconds:

Indexing Properties

```
db.log.createIndex(
    {
        "createdAt" : 1
    },{
        "expireAfterSeconds":3000
    });
```

Once the index is created, we get an acknowledgment message:

```
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1.0
}
```

The limitations of indexes:

- A single collection can have up to 64 indexes only.
- The qualified index name is ..$ and cannot have more than 128 characters. By default, the index name is a combination of index type and field name.
- You can specify an index name while using the createIndex() method to ensure that the fully-qualified name does not exceed the limit.
- There can be no more than 31 fields in the compound index.

The limitations of indexes:

- The query cannot use both text and geospatial indexes. You cannot combine the $text operator, which requires text indexes, with some other query operator required for special indexes.

  For example, you cannot combine the $text operator with the $near operator.

- Fields with 2d sphere indexes can only hold geometry data. 2d sphere indexes are specially provided for geometric data operations.

  For example, to perform operations on co-ordinate, we have to provide data as points on a planer co-ordinate system, [x, y]. For non-geometries, the data query operation will fail.

The limitation on data:

- The maximum number of documents in a capped collection must be less than 2^32. We should define it by the max parameter while creating it. If you do not specify, the capped collection can have any number of documents, which will slow down the queries.
- The MMAPv1 storage engine will allow 16,000 data files per database, which means it provides the maximum size of 32 TB.
- We can set the storage.mmapv1.smallfile parameter to reduce the size of the database to 8 TB only.
- Replica sets can have up to 50 members.
- Shard keys cannot exceed 512 bytes.

Summary

This session will give the knowledge about

- Indexing
- References:
  - https://docs.mongodb.com/manual/tutorial/
  - “Seven NoSQL Databases in a Week” Author: Aaron Ploetz Aaron