CSCU9YQ - NoSQL Databases
Lecture 6.b: Indexes in MongoDB

Gabriela Ochoa

Indexes

- Indexes can improve the efficiency of read (query) operations.
- Without indexes, each document in a collection must be scanned to select those that match the query statement
- Defined at the *collection* level
- Supported on any field or sub-field of the documents

Indexes

- Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form.
- They are kept in memory
- The index stores the value of a specific field or set of fields, **ordered** by the value of the field.
- Each entry in the index points to the record in the DB

https://docs.mongodb.com/manual/indexes/

Just like the Index in a Book

A
Ant 100
Armadillo.... 129
Apple 140
…
B
Bee 210
…
C
…

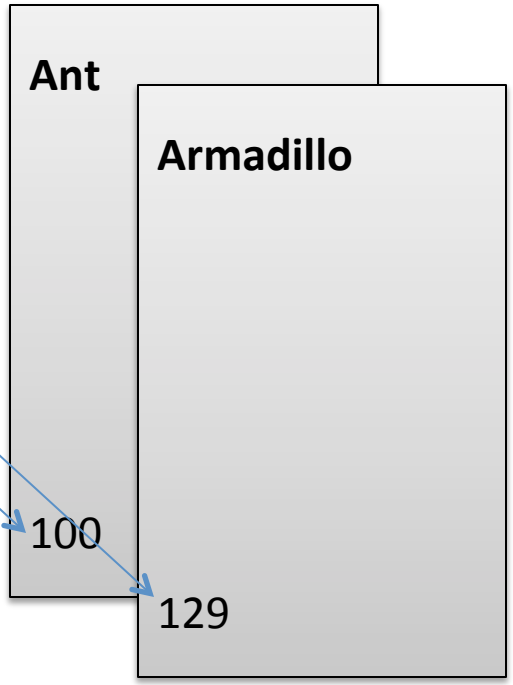
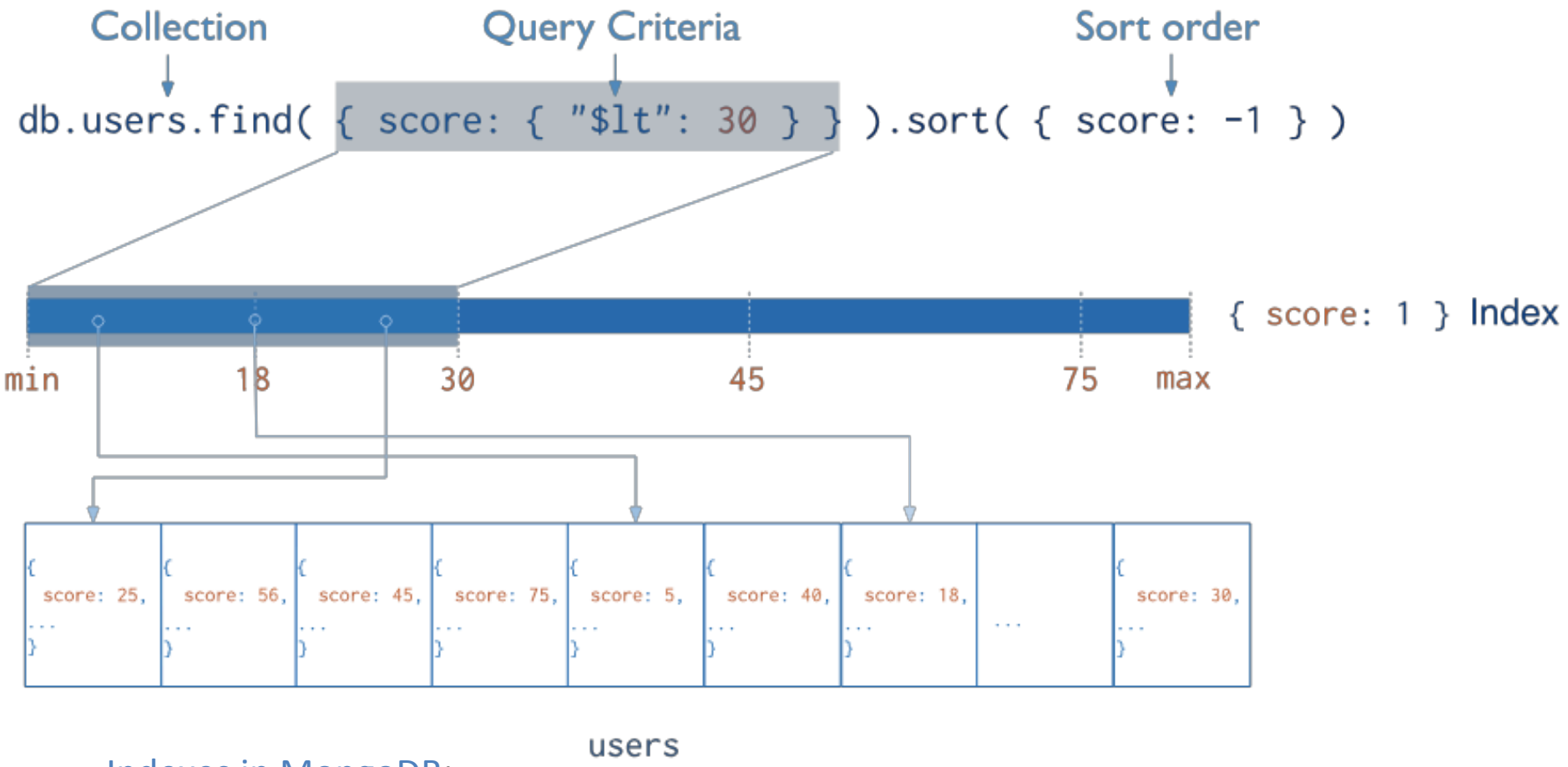


Illustration of query that selects and orders the matching documents using an index:

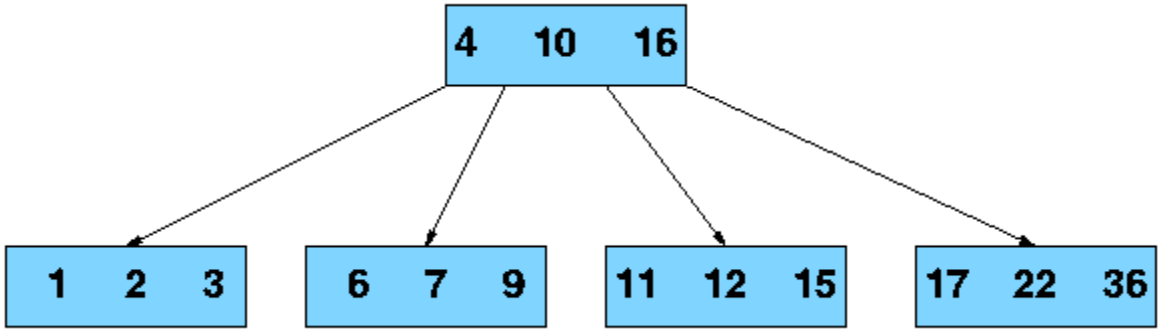


Indexes in MongoDB:

- Similar to indexes in other database systems
- Defined at the collection level
- Can be defined on any field or sub-field of the documents

Index Structure

- Uses B-Trees, which are common DB index structures



- B-Trees maintain sorted data. Generalisation of Binary search tree, nodes can have more than two children
- Search, insert and delete are all possible in O(log n) time

Why Index

- Searching every record is O(n) – every field must be searched
- Searching B-Tree index is O(log n) – much faster
- Other benefits:
  - Results already sorted without the need for an additional sort operation after a query
  - Covered results don’t need DB access at all, just return the data from the index (e.g. Count the scores over 30) if the score field has an index
  - Range search easy

Index Syntax

db.collection.createIndex(<keys>, <options>)

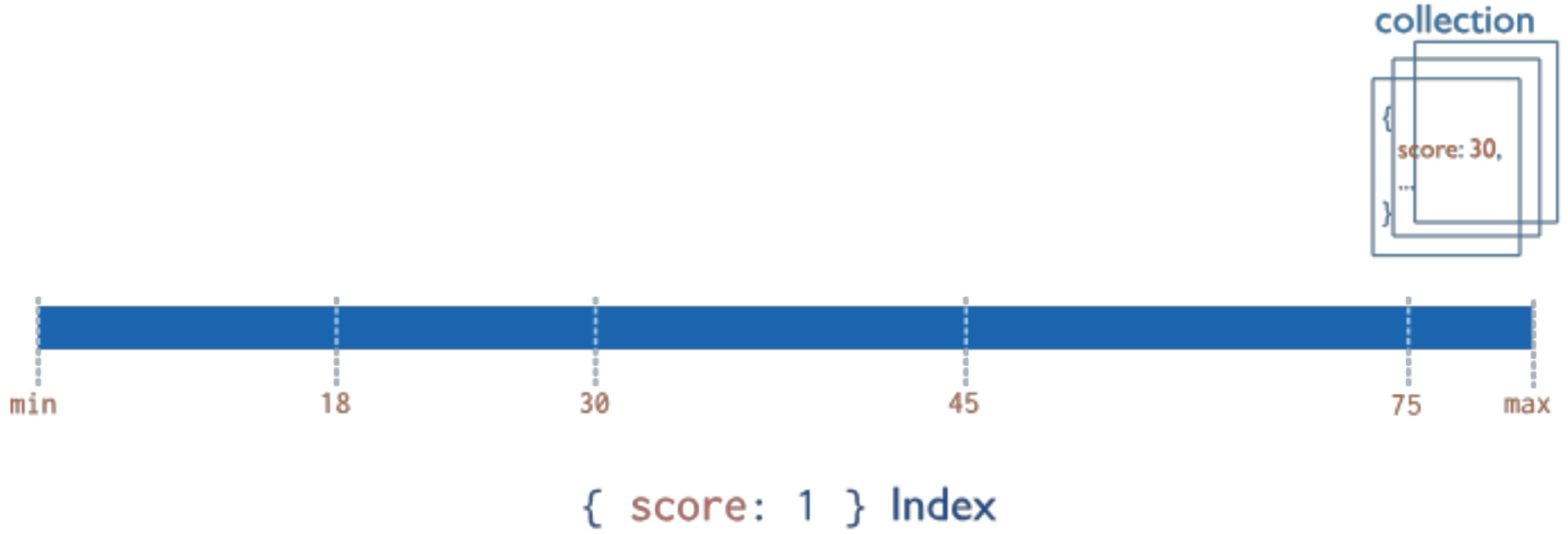
Parameter	Type	Description
Keys	JSON Doc	Field and value pairs where the field is the index key and the value describes the type of index. For an ascending index, value of 1; for descending index, value of -1. Example Index Types: text, geospatial, hashed indexes.
Options	JSON Doc	Contains a set of options that controls the creation of the index. Example options: unique, name, sparse

https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/#db.collection.createIndex

Index Types

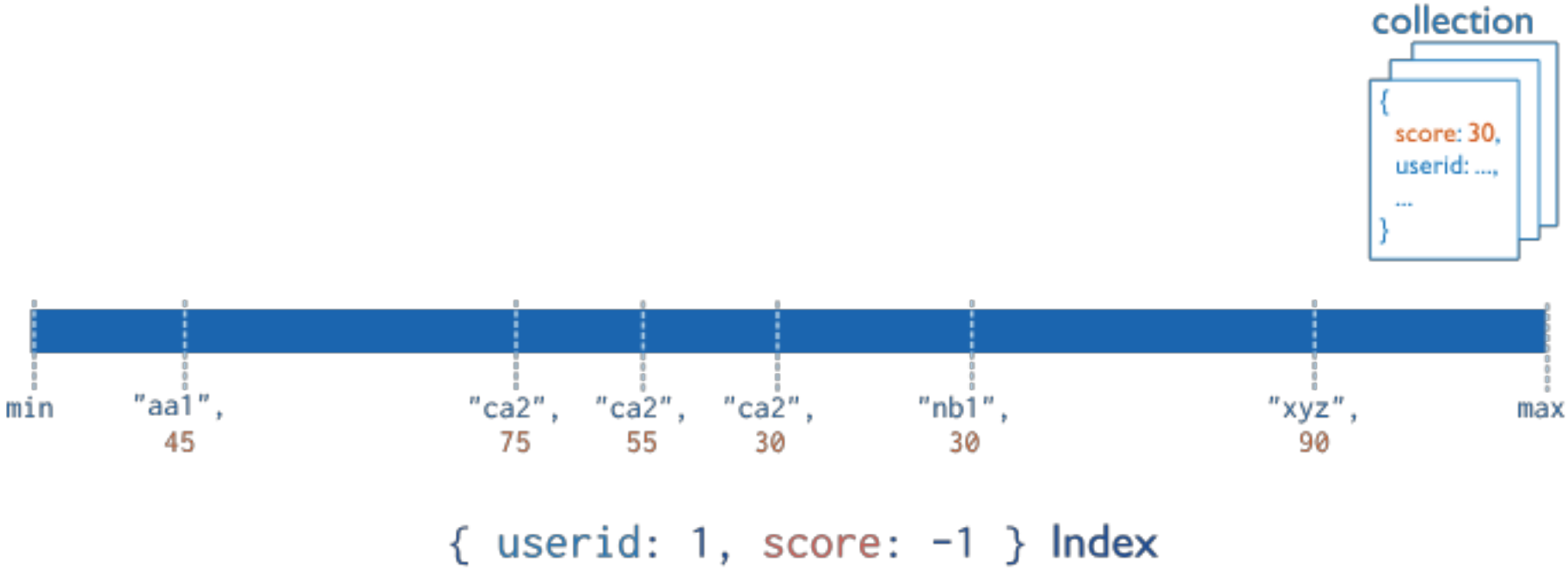
- **Default _id** – The ID given to each field by MongoDB is indexed by default
- **Single field** – User defined indexes on single fields, e.g username
- **Compound Index** – Sorts first by one index, then by the next etc. Useful if values are duplicated or coverage needed across more than one field

Single Field Index



- db.collection.createIndex({score: 1})
- Sort in an ascending order, but for single field MongoDB can traverse the index in either direction.

Compound Index



- db.collection.createIndex({userid:1, score: -1})
- The index sorts first by userid and then, within each userid value, sorts by score.

Other Types of Indexes

- **Multikey indexes** to index the content stored in arrays.
- **Geospatial Indexes** to support efficient queries of geospatial coordinate data**,**
- **Text indexes** – ignore words like ‘a’,’the’ etc. See http://docs.mongodb.org/manual/core/index-text/
- **Hashed Indexes** – Indexes the hash of a field to support hash bases sharding

Index Properties (Options)

- **Unique Indexes** reject duplicate values for the index field
- **Sparse Indexes** only index entries that have the index field, so only contain a subset of the whole collection
- **Non-sparse indexes** contain every document in the collection, using a null for those that do not contain the indexed field

Other Index Operations

```
db.collection.getIndexes()

db.collection.dropIndex(name)
```

- Name is the one you gave it or, by default, fieldname_order (1 if unique, 2
- E.g. `orderDate_1`

Example: Text Index Search

- Perform text searchers over properties in your documents
  1. First create and index
  2. Use the index in a special $text filter


```
[
   {"name":"Connie Coffman","company":"Discount Bicycle Specialists"},
   {"name":"Gytis Barzdukas","company":"Transportation Options"},
   {"name":"Paul Alcorn","company":"Major Sport Suppliers"},
   {"name":"Shanay Steelman”,"company":"One Bike Company"},
   {"name":"Shane Kim","company":"Twelfth Bike Store"},
  {"name":"Cornelius Brandon","company":"Initial Bike Company"},
  {"name":"Terry Eminhizer","company":"Action Bicycle Specialists"},
  {"name":"Jean Handley","company":"Central Discount Store"}
]
```

Example: Text Index Search

- Create a text index using the company field
  - db.customers.createIndex({ company: "text" })
- Search query using the find operation and the $text operator
  - db.customers.find({ $text: { $search: "Bike" }})
  - Returns all documents with the word “Bike” in the company field
- Search for all documents that match any word in the search string
  - db.customers.find({ $text: { $search: "Bicycle Bike" }})
  - Returns documents documents that match Bicycle or Bike

Example: Text Index Search

- Search using a phrase consisting of multiple words. Encapsulate phrase in quotations
  - db.customers.find({ $text: { $search: "\"Bike Company\"" }})
  - Returns all documents with text “Bike Company”
- Queries are case insensitive by default. You can modify this at the index or you can manually indicate that a search must be case sensitive:
  - db.customers.find({ $text: { $search: "bike", $caseSensitive: true }})

Choosing Indexes

- Think about common queries
  - Compound indexes speed queries across multiple fields
- Indexes are stored in RAM for speed
- Don’t build so many that RAM is filled
- Also, Indexes can slow down writes as the index needs to be altered too, so only index what you need

Summary

- Indexes can improve the efficiency of read (query) operations.
- Mongo DB provides several types
  - Default _id
  - Single field
  - Compound Index
  - Multikey, Geospatial, Hashed

db.collection.createIndex(<keys>, <options>)