Big Data Indexing

Prepared by Jeong-Hun Kim

Table of Content

- **In the last lecture**
- What is index?
- Single field indexes
- Compound indexes
- Multi-key indexes
- Practice

In the last lecture

- Aggregation Framework
- Aggregation Pipeline
- Aggregation Pipeline Stages
  - $match
  - $project
  - $group
  - $unwind
  - $sort, $skip and $limit

Table of Content

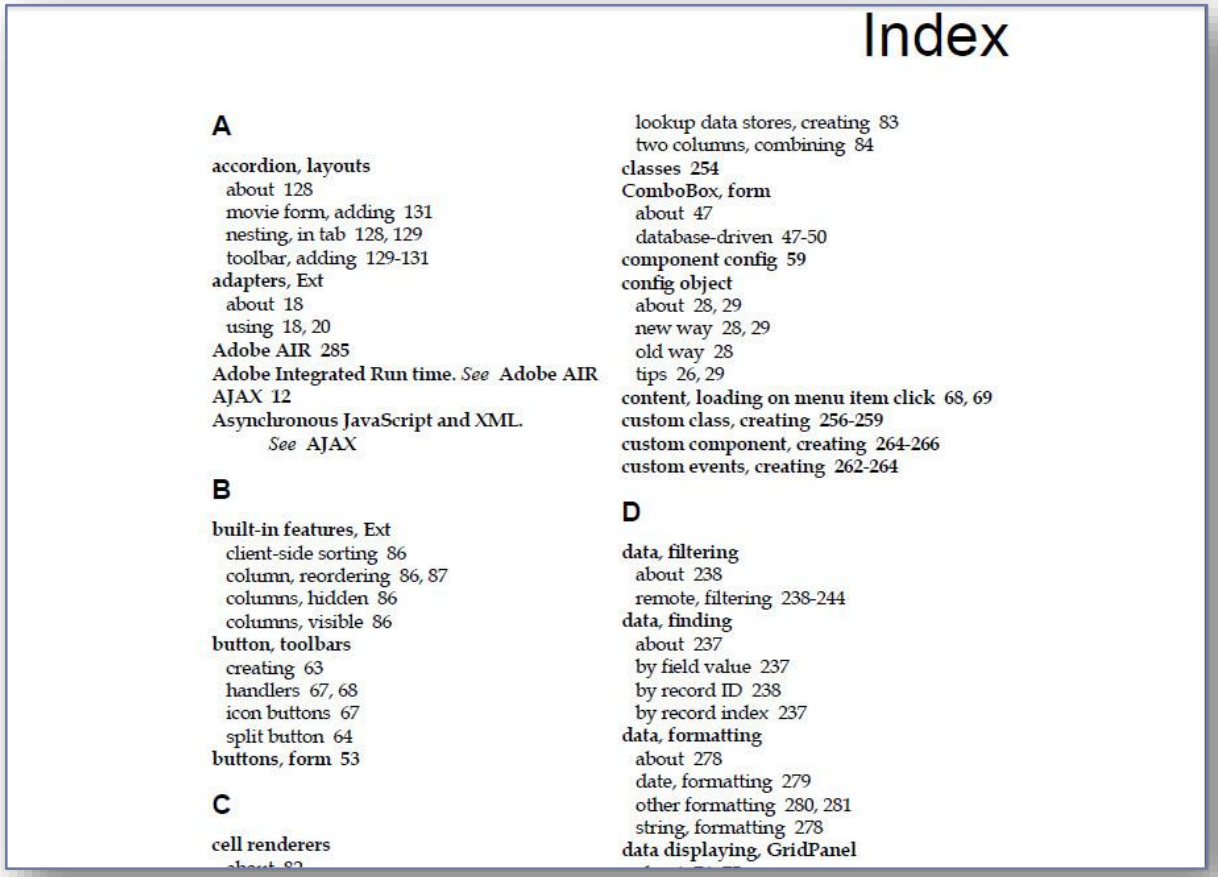
- In the last lecture
- **What is index?**
- Single field indexes
- Compound indexes
- Multi-key indexes

What is indexing?

- Introduction to indexing
  - Similar to a book's index
  - Instead of looking through the whole book
    - The database takes a shortcut and just looks at an ordered list that points to the content
  - Allows it to query orders of magnitude faster
  - A query that does not use an index is called a **table scan**
    - The server has to "look through the whole book" to find a query's results

What is indexing?

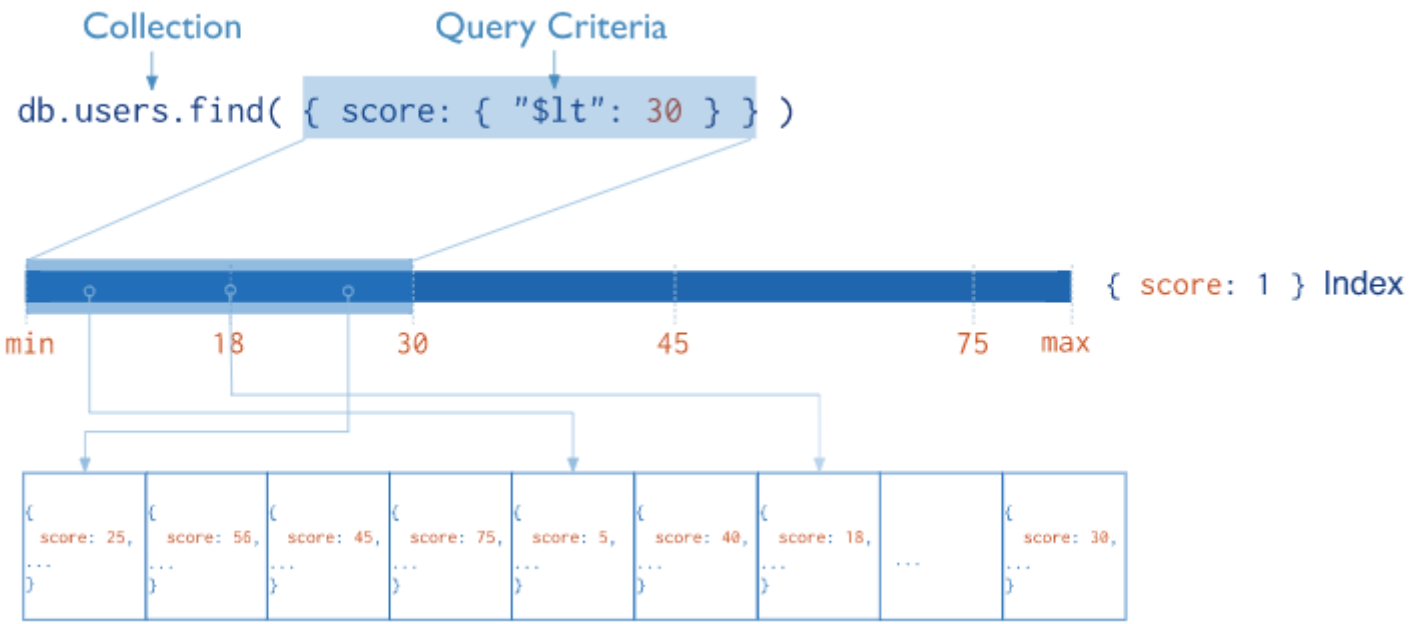
- Introduction to indexing



Index

What is indexing?

- Indexing in MongoDB
  - MongoDB indexes use a B-tree data structure

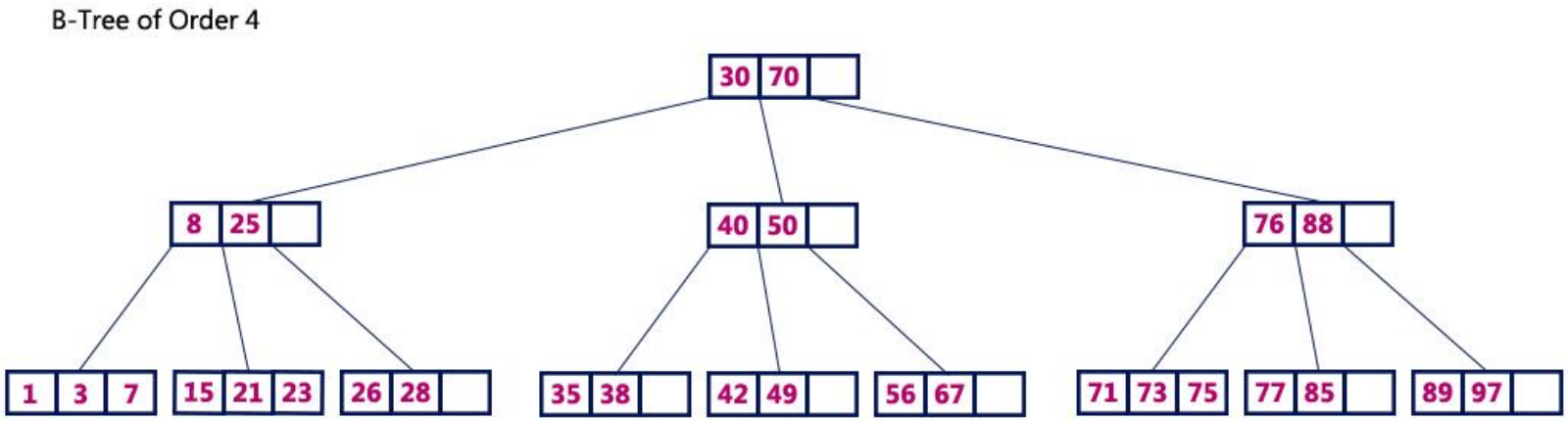


What is indexing?

- MongoDB index types
  - Single field
  - Compound index
  - Multikey index
  - Text index
  - Geospatial index (2dsphere index)
  - Hashed index

What is indexing?

- MongoDB uses B-Tree
  - A self-balanced search tree with multiple keys in every node and more than two children for every node.

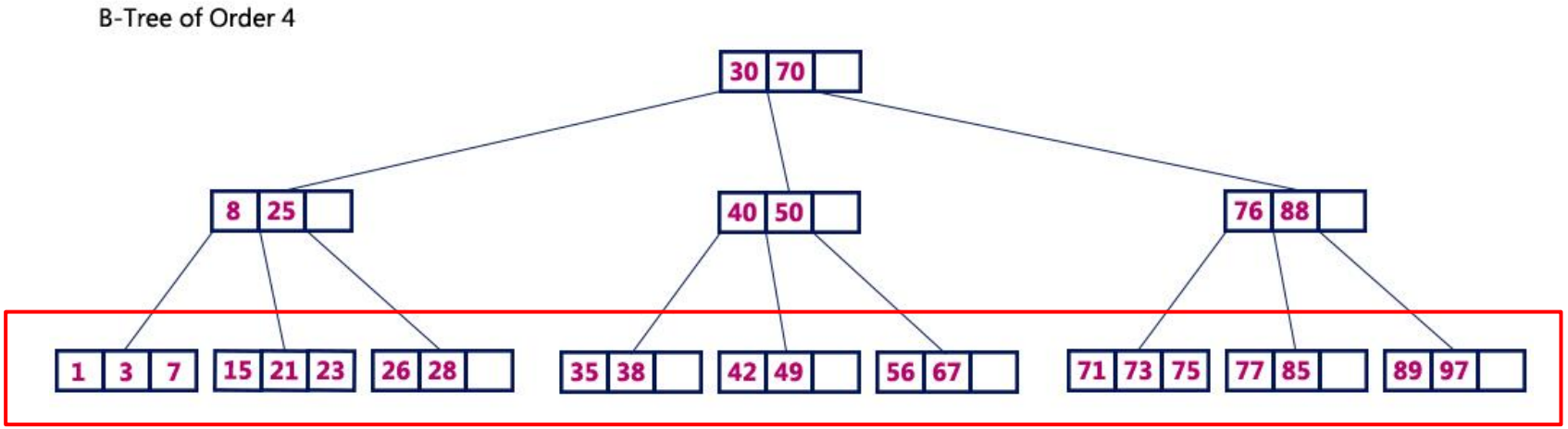


What is indexing?

- B-Tree of Order m has the following properties
  1. All the **leaf nodes** must be **at the same level.**
  2. All nodes except root must have at least **[m/2]-1** keys and maximum of **m-1** keys.
  3. All non leaf nodes except root (i.e. all internal nodes) must have at least m/2 children.
  4. If the root node is a non-leaf node, then it must have **at least 2** children.
  5. A non leaf node with **n-1** keys must have **n** number of children.
  6. All the **key values within a node** must be in **Ascending Order**.

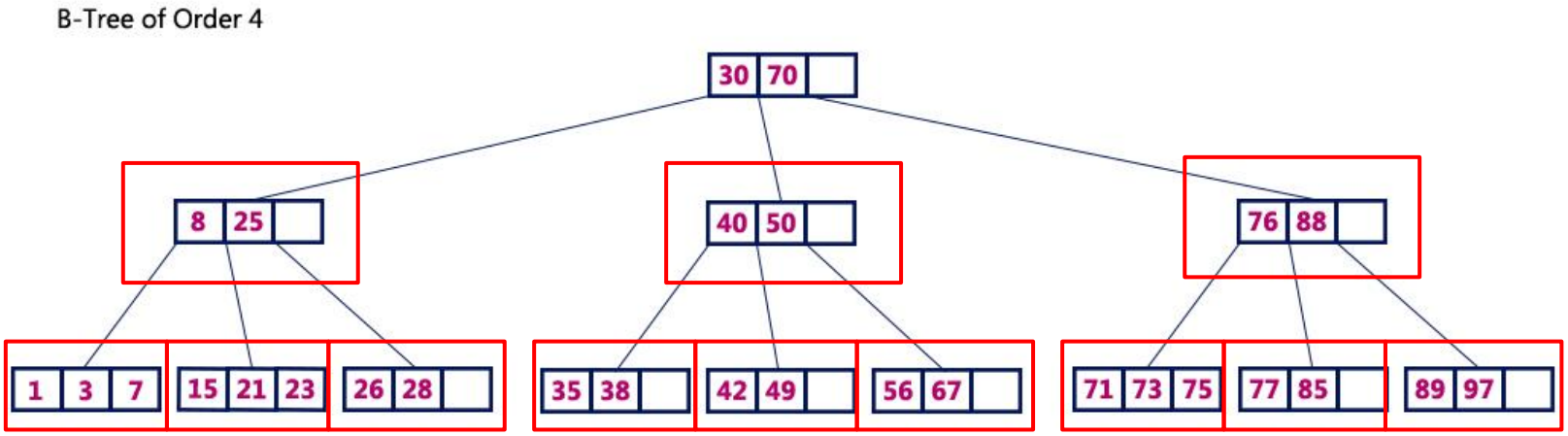
What is indexing?

- B-Tree of Order m has the following properties
  - Property 1: All the **leaf nodes** must be **at the same level.**



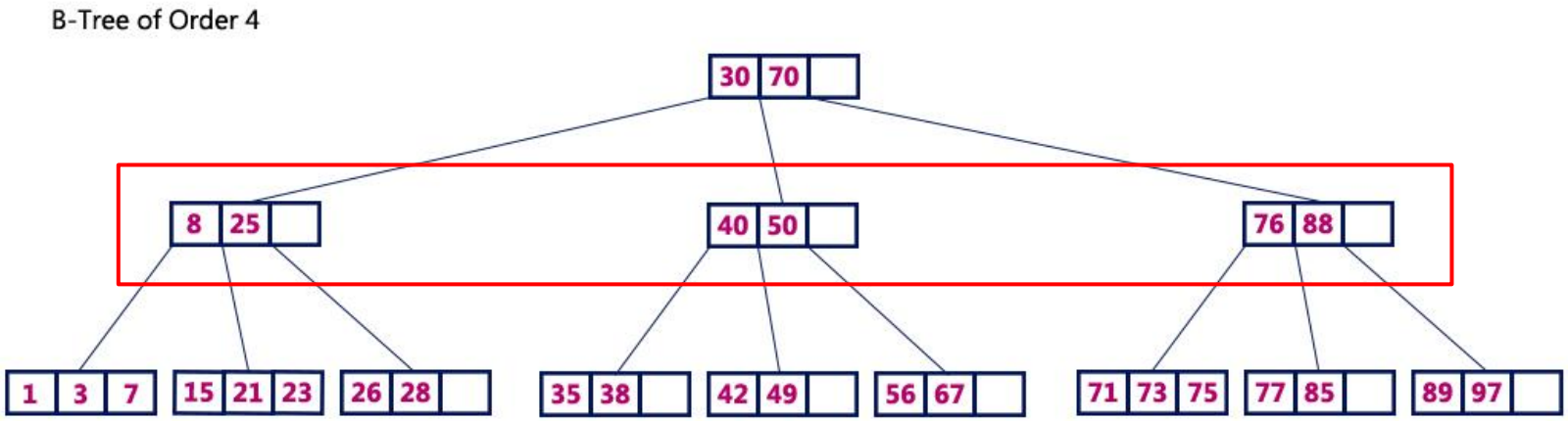
What is indexing?

- B-Tree of Order m has the following properties
  - Property 2: All nodes except root must have at least **[m/2]-1** keys and maximum of **m-1** keys.



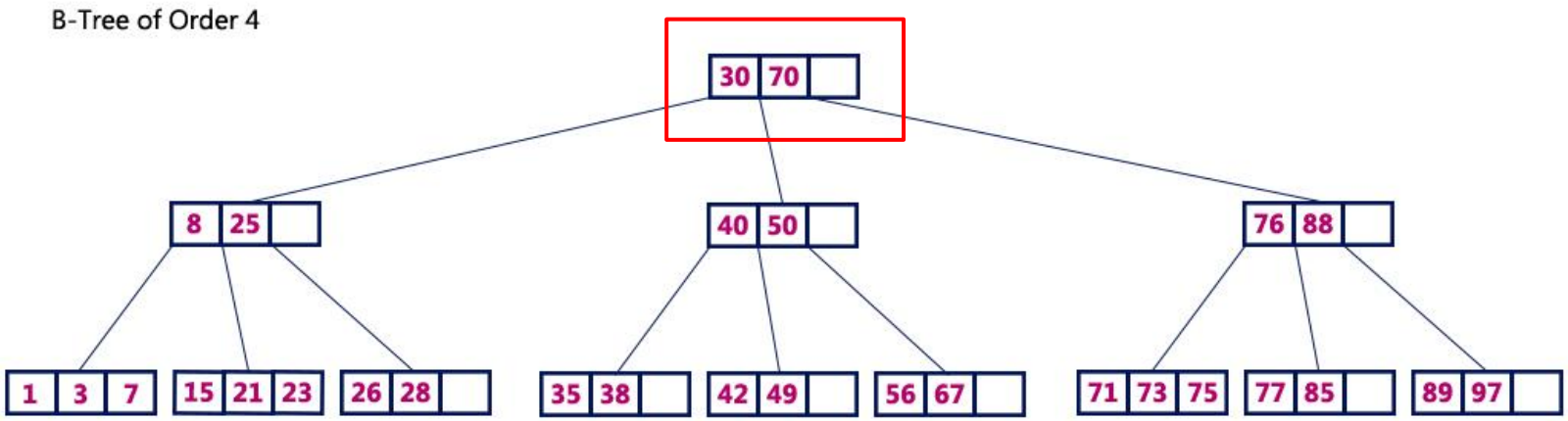
What is indexing?

- B-Tree of Order m has the following properties
  - Property 3: All non-leaf nodes except root (i.e. all internal nodes) must have at least m/2 children.



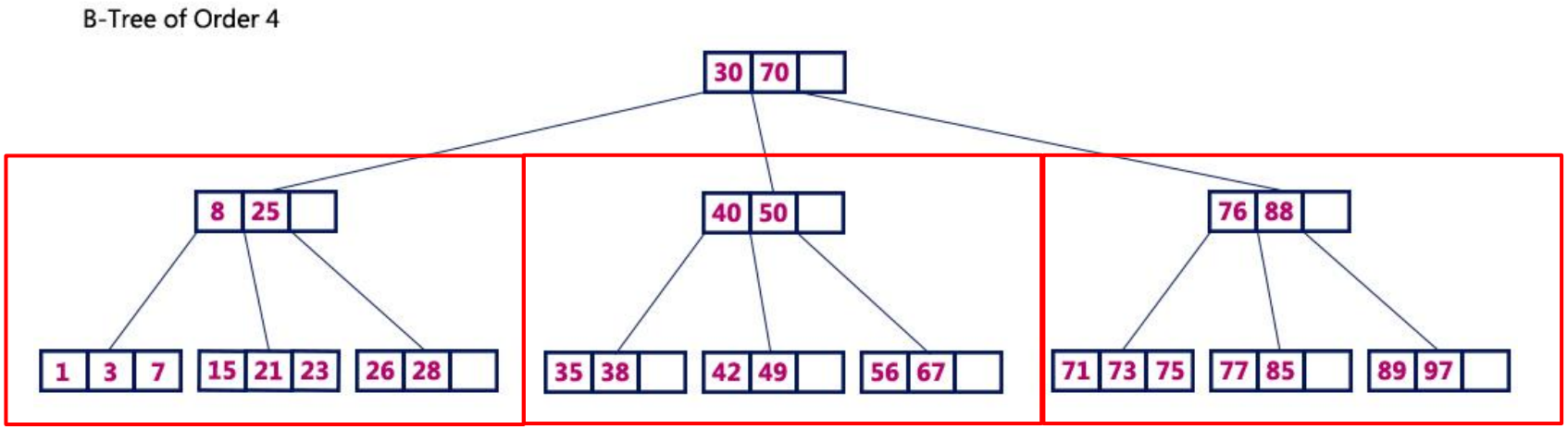
What is indexing?

- B-Tree of Order m has the following properties
  - Property 4: If the root node is a non-leaf node, then it must have **at least 2** children



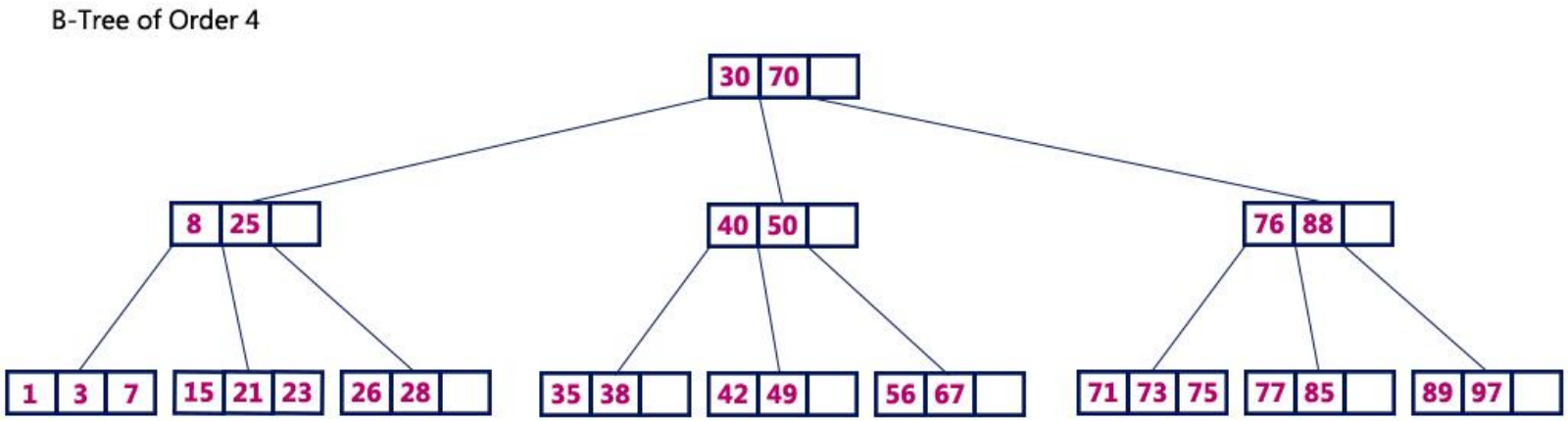
What is indexing?

- B-Tree of Order m has the following properties
  - Property 5: A non-leaf node with **n-1** keys must have **n** number of children.



What is indexing?

- B-Tree of Order m has the following properties
  - Property 6: All the **key values within a node** must be in **Ascending Order**.



What is indexing?

- In a B-Tree, the new element must be added only at leaf node.
  1. If tree is **Empty**, then create a new node with new key value and insert into the tree as a root node.
  2. If tree is **Not Empty**, then find a leaf node to which the new key value can be added using Binary Search Tree logic.
  3. If that leaf node has an empty position, then add the new key value to that leaf node by maintaining **ascending order** of key value within the node.
  4. If that leaf node is already full, then split that leaf node by sending **middle value** to its parent node. Repeat that same until sending value is fixed into a node.
  5. If the splitting is occurring to the root node, then **the middle value becomes new root node** for the tree and the height of the tree is increased by one.

Example

- Example: Construct a B-Tree of Order 3 by inserting numbers from 1 to 10.

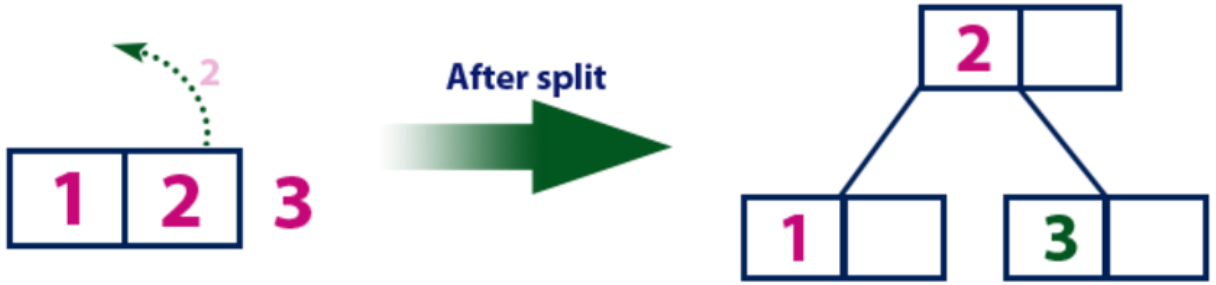
- **Insert '1'**
  - Rule 1:
    - If tree is **Empty**, then create a new node with new key value and insert into the tree as a root node.

Example

- **Insert '2'**
  - Rule 2:
    - If tree is **Not Empty**, then find a leaf node to which the new key value can be added using Binary Search Tree logic.
  - Rule 3:
    - If that leaf node has an empty position, then add the new key value to that leaf node by maintaining ascending order of key value within the node.

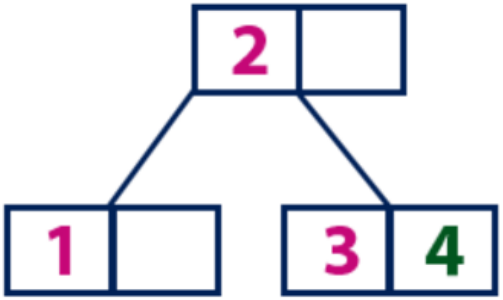
Example

- Insert '3'
  - Rule 4:
    - If that leaf node is already full, then split that leaf node by sending middle value to its parent node.
  - Rule 5:
    - If the splitting is occurring to the root node, then the middle value becomes new root node for the tree and the height of the tree is increased by one.



Example

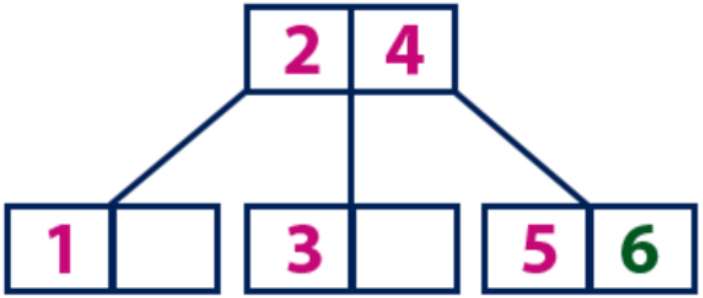
- Insert '4'



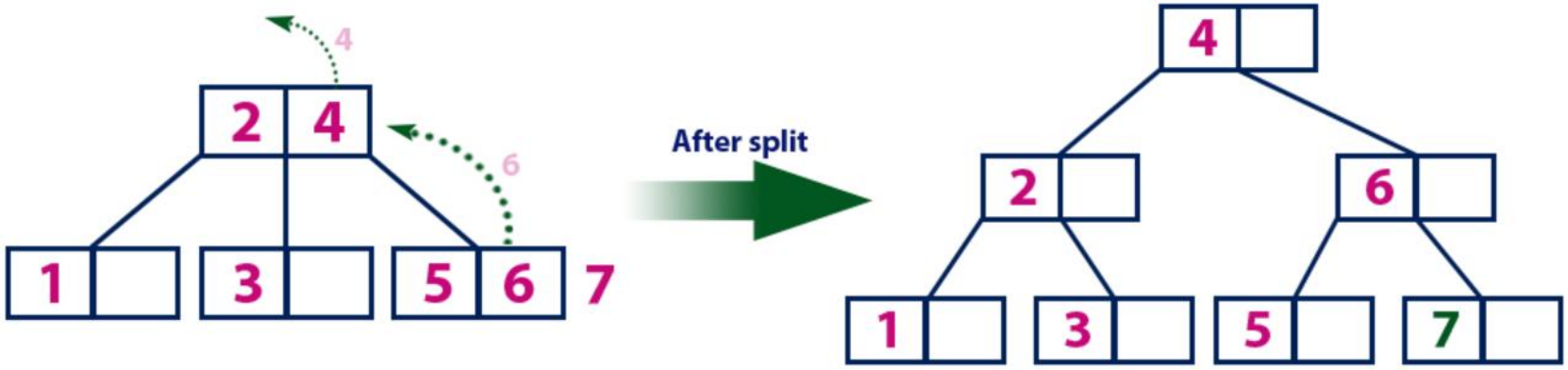
- Insert '5'

Example

- Insert '6'

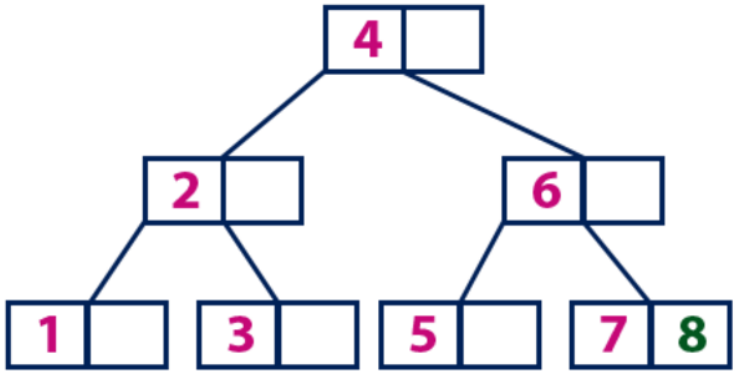


- Insert '7'

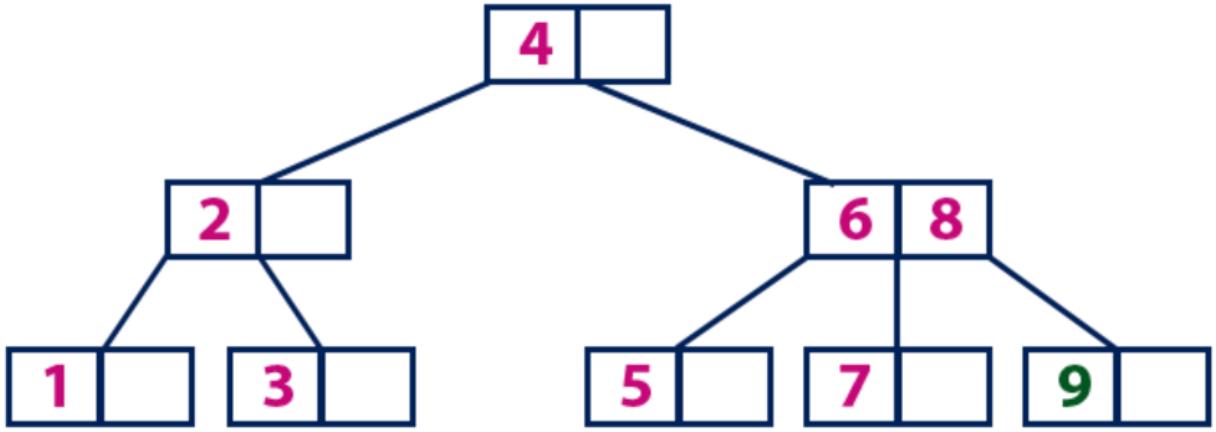


Example

- Insert '8'

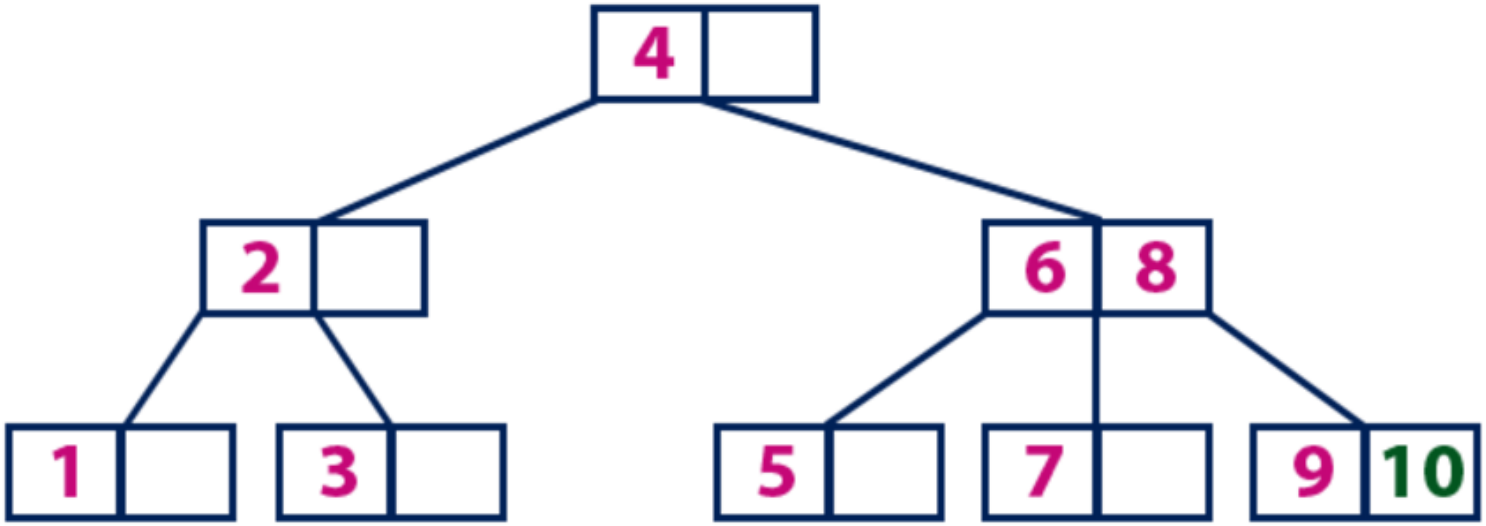


- Insert '9'



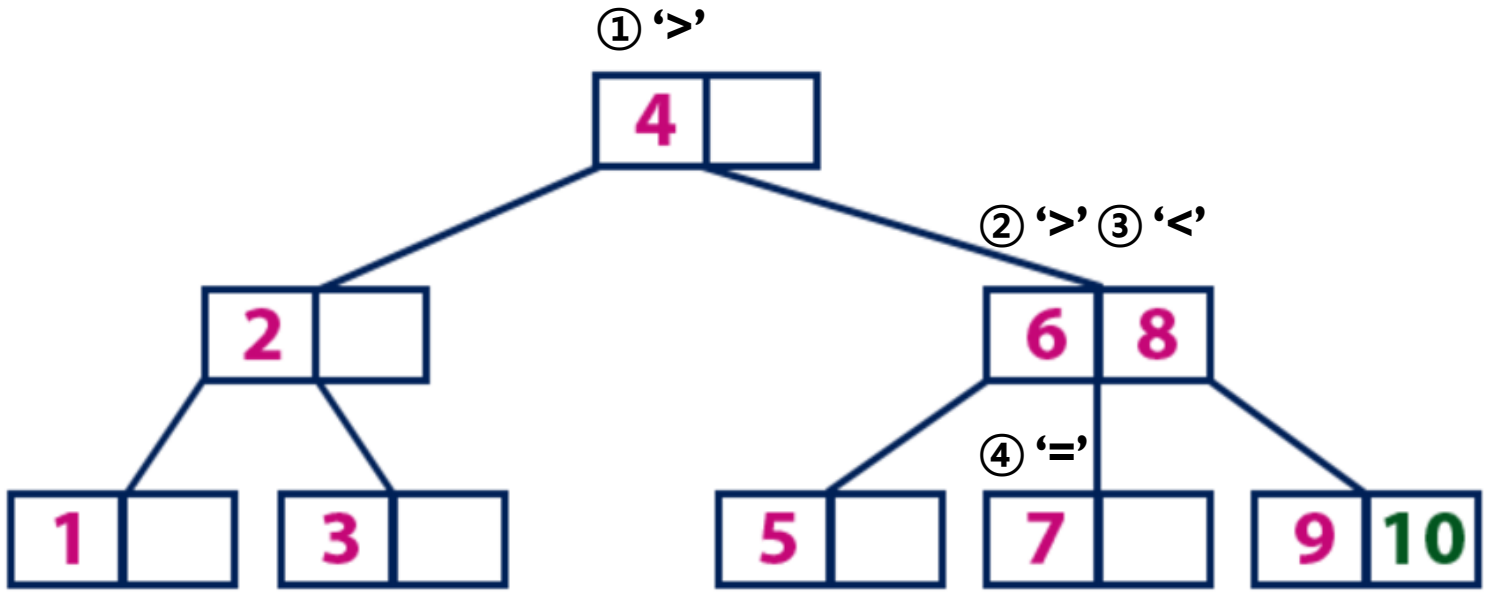
Example

- Insert '10'



How to search a key using B-Tree

- B-tree searches for the key top-down from the root node to the leaf node
- Search '7'



Disadvantages of index

- Although indexes are intended to enhance a database's performance, there are times when they should be avoided.
  - Indexes should not be used on small collections.
  - Collections that have frequent, large batch update or insert operations.
  - Indexes should not be used on fields that contain a high number of NULL values.
  - Fields that are frequently manipulated should not be indexed.

Table of Content

- In the last lecture
- What is index?
- **Single field indexes**
- Compound indexes
- Multi-key indexes
- Practice

Single Field Indexes

- Indexes
  - By default, all collections have an index on the _id field
  - createIndex() is used to create indexes on collections
    - *db.collection.createIndex(keys, options)*
      - keys is a document ({<field>: value})
      - options is a document that contains a set of options that controls the creation of the index

Single Field Indexes

- Create an Ascending Index on a Single Field
  - Suppose that a collection named records is given

```
{
  "_id": ObjectId("570c04a4ad233577f97dc459"),
  "score": 1034,
  "location": { state: "NY", city: "New York" }
}
```

  - Create an ascending index on the score field of the records collection
    - db.records.createIndex({ score: 1 })
  - Queries
    - db.records.find({ score: 2 })
    - db.records.find({ score: { $gt: 10 } })

Single Field Indexes

- Create an Index on an Embedded Field
  - Suppose that a collection named records is given

```
{
  "_id": ObjectId("570c04a4ad233577f97dc459"),
  "score": 1034,
  "location": { state: "NY", city: "New York" }
}
```

  - The following operation creates an index on the location.state field
    - db.records.createIndex({ "location.state": 1 })
  - Queries
    - db.records.find({ "location.state": "CA" })
    - db.records.find({ "location.city": "Albany", "location.state": "NY" })

Table of Content

- In the last lecture
- What is index?
- Single field indexes
- **Compound indexes**
- Multi-key indexes
- Practice

Compound indexes

- What is compound index?
  - A single index structure holds references to multiple fields within a collection's documents
  - Compound indexes can support queries that match on multiple fields
    - MongoDB imposes a limit of 32 fields for any compound index
  - To create a compound index use an operation that resembles the following prototype
    - *db.collection.createIndex({ <field1>: <type>, <field2>: <type2>, ... })*

Compound indexes

- Example
  - Suppose that the following collection PRODUCTS is given

    {"id":"1", "product":"chips", "manufacturer":"lays", "price":20}

    {"id":"2", "product":"pringles", "manufacturer":"Kellogg's", "price":99}

    {"id":"3", "product":"chips", "manufacturer":"lays", "price":10}

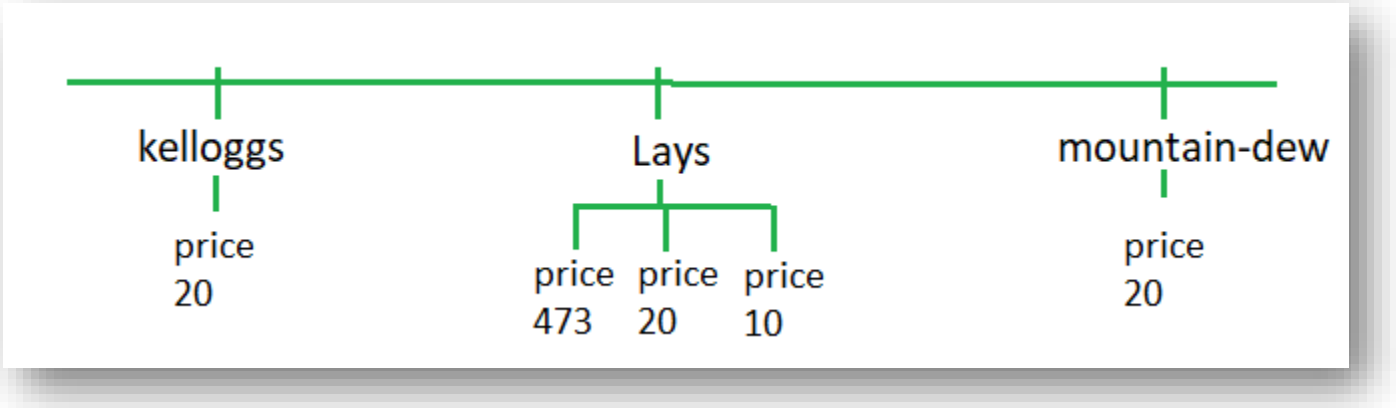
    {"id":"4", "product":"chips", "manufacturer":"lays", "price":473}

    {"id":"5", "product":"coldrink", "manufacturer":"mountain-dew", "price":20}

  - Creating a compound index
    - db.products.createIndex(manufacturer:1, price:-1)

Compound indexes

- Example
  1. Sorted first by the values of the manufacturer field in ascending order
  2. Sorted by values of the price field within manufacturing field in descending order



Compound indexes

- A compound index includes a set of **prefixes**
  - Term for the beginning combination of fields in the index
  - The order of the fields in the compound index is important
    - It dictates the index prefixes available for querying
  - Example
    - db.products.createIndex(product:1, manufacturer:1, price:-1)
    - This index has the following prefixes:
      - {product: 1}
      - {product: 1, manufacturer: 1}

Compound indexes

- Index prefix
  - Example
    - This means our compound query supports queries on
      - product
      - product and manufacturer
      - product and manufacturer and price
    - This also means that our compound query DOES NOT support queries on
      - manufacturer
      - price
      - manufacturer and price
  - Two birds one stone with index prefix
    - No need to create a single field index on product field

Compound indexes

- The Equality, Sort, Range (ESR) rule
  - Compound indexed fields should be in order of equality, then sorting, then range
  - Example
    - Given the following compound index, how to write a query to efficiently utilize the index
      - db.products.createIndex(product:1, manufacturer:1, price:-1)
    - Query
      - db.products.find({product: "chips", price: {$lt: 50}}).sort({manufacturer: 1})
    - Meaning
      - The equality (product) comes first, followed by sort (manufacturer), followed by range (price)

Compound indexes

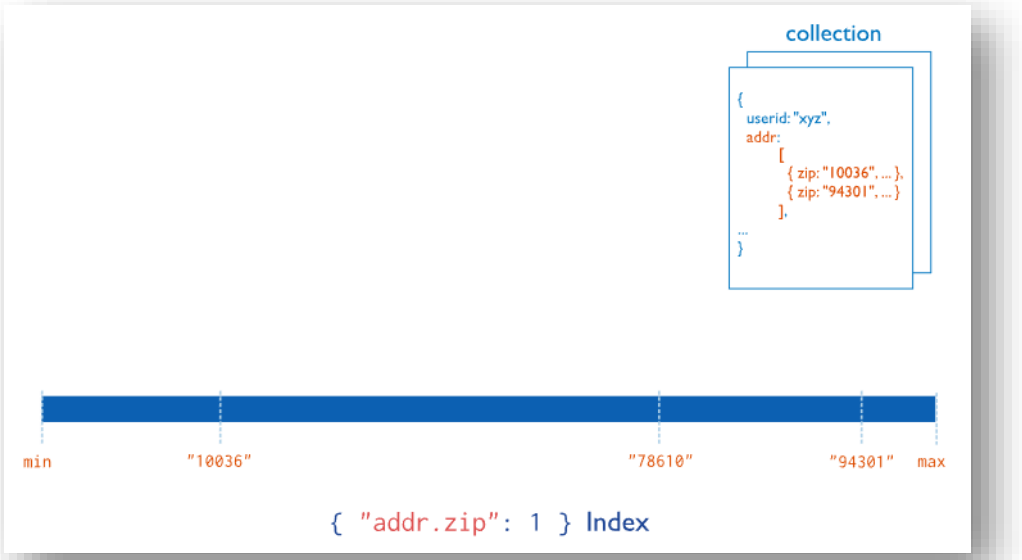
- The Equality, Sort, Range (ESR) rule
  - Example
    - Suppose that you want make the following query for searching cars collection
    - Query
      - db.cars.find({manufacturer: 'Ford', cost: { $gt: 10000 }}).sort({ model: 1 })
    - Application of ESR Rule
      - manufacturer: 'Ford' is an equality based match
      - cost: { $gt: 10000 } is a range based match, and
      - model is used for sorting
    - How to create an optimal index for the query using ESR rule
      - db.cars.createIndex{ manufacturer: 1, model: 1, cost: 1 }

Table of Content

- In the last lecture
- What is index?
- Single field indexes
- Compound indexes
- **Multi-key indexes**
- Practice

Multi-key Indexes

- To index a field that holds an array value, MongoDB creates an index key for each element in the array



- Syntax
  - db.coll.createIndex({ <field>: < 1 or -1 > })

Multi-key Indexes

- Create an Index on Array of Values (Multikey Index)
  - Suppose that a collection named inventory is given

    { _id: 5, type: "food", item: "aaa", ratings: [5, 8, 9] }
    { _id: 6, type: "food", item: "bbb", ratings: [5, 9] }
    { _id: 7, type: "food", item: "ccc", ratings: [9, 5, 8] }
    { _id: 8, type: "food", item: "ddd", ratings: [9, 5] }
    { _id: 9, type: "food", item: "eee", ratings: [5, 9, 5] }

  - The collection has a multikey index on the ratings field:
    - db.inventory.createIndex({ ratings: 1 })
  - Query
    - db.inventory.find({ ratings: [5, 9] })

```
> db.multikey.find({ratings: [5, 9]})
< {
    _id: 6,
    type: 'food',
    item: 'bbb',
    ratings: [
      5,
      9
    ]
  }
```

Multi-key Indexes

- Create an Index on Array of Values (Multikey Index)
  - Query
    - db.inventory.find({ ratings: {$lte: 6, $gte: 8}}) ???
  - Result

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
{ _id: 8, type: "food", item: "ddd", ratings: [ 9, 5 ] }
{ _id: 9, type: "food", item: "eee", ratings: [ 5, 9, 5 ] }
```

    - Why? The meaning of query is not equal to $6 \leq x \leq 8$
      - Suppose A is a result of the query {ratings: {$lte: 6}} and
        B is a result of the query {ratings: {$gte: 8}}

        the result of {ratings: {$lte: 6, $gte: 8}} is $A \cup B$

Multi-key Indexes

- Create an Index on Array of Documents
  - Suppose that a collection named inventory is given

```
{
  _id: 1,
  item: "abc",
  stock: [
    { size: "S", color: "red", quantity: 25 },
    { size: "S", color: "blue", quantity: 10 },
    { size: "M", color: "blue", quantity: 50 }
  ]
}
```

  - The following operation creates a multikey index on the stock.size
    - db.inventory.createIndex({ "stock.size": 1 })
  - Query
    - db.inventory.find({ "stock.size": "M" })

Questions?

See you next time!