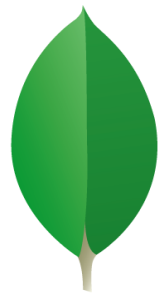


# NoSQL and MongoDB



mongoDB

Advanced

Eng .Hany Saad  
SD Dept.  
ITI – Assiut Branch

# Indexes

---

- Indexes support the **efficient resolution of queries**.
- **Without** indexes, MongoDB must **scan every document of a collection** to select those documents that match the query statement. This scan is highly inefficient and require MongoDB to process a large volume of data.
- Indexes are **special data structures, that store a small portion of the data set in an easy-to-traverse form**.
- The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.
- To create an index you need to use `ensureIndex()` method of MongoDB.
- Syntax:
  - `>db.COLLECTION_NAME.ensureIndex({KEY:1})`
  - 1 for ascending order, -1 for descending order.

# Indexes (Cont.)

- **ensureIndex()** method also accepts list of options (which are optional).

Following is the list

Parameter	Type	Description
background	Boolean	Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is <b>false</b> .
unique	Boolean	Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create a unique index. The default value is <b>false</b> .
name	string	The name of the index. If unspecified, MongoDB generates an index name by concatenating the names of the indexed fields and the sort order.
sparse	Boolean	If true, the index only references documents with the specified field. These indexes use less space but behave differently in some situations (particularly sorts). The default value is <b>false</b> .
expireAfterSeconds	integer	Specifies a value, in seconds, as a TTL to control how long MongoDB retains documents in this collection.

# Aggregation

---

- Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.
- Syntax:
  - `>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)`

# Aggregation (Cont.)

- Aggregation expressions:

Expression	Description	Example
\$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$sum : "\$likes"}}}])
\$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$avg : "\$likes"}}}])
\$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$min : "\$likes"}}}])
\$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$max : "\$likes"}}}])
\$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{\$group : {_id : "\$by_user", url : {\$push : "\$url"}}}])
\$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{\$group : {_id : "\$by_user", url : {\$addToSet : "\$url"}}}])
\$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "\$sort"-stage.	db.mycol.aggregate([{\$group : {_id : "\$by_user", first_url : {\$first : "\$url"}}}])
\$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "\$sort"-stage.	db.mycol.aggregate([{\$group : {_id : "\$by_user", last_url : {\$last : "\$url"}}}])

# Relationships

---

- Relationships can be modeled via **Embedded** and **Referenced** approaches.
- Modeling Embedded Relationships:**
  - This approach maintains all the related data in a single document, which makes it easy to retrieve and maintain. The whole document can be retrieved in a single query:
    - `>db.users.findOne({"name":"Tom Benzamin"}, {"address":1})`
  - The drawback is that if the embedded document keeps on growing too much in size, it can impact the read/write performance.

```
{
  "_id": ObjectId("52ffc33cd85242f436000001"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin",
  "address": [
    {
      "building": "22 A, Indiana Apt",
      "pincode": 123456,
      "city": "Los Angeles",
      "state": "California"
    },
    {
      "building": "170 A, Acropolis Apt",
      "pincode": 456789,
      "city": "Chicago",
      "state": "Illinois"
    }
  ]
}
```

# Relationships (Cont.)

---

- **Modeling Referenced Relationships:**

- In this approach, both the user and address documents will be maintained separately but the user document will contain a field that will reference the address document's **id** field.
- With this approach, we will need two queries:
  - **>var result = db.users.findOne({"name":"Tom Benzamin"}, {"address\_ids":1})**
  - **>var addresses = db.address.find({"\_id":{"\$in":result["address\_ids"]}})**

```
{
  "_id":ObjectId("52ffc33cd85242f436000001"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin",
  "address_ids": [
    ObjectId("52ffc4a5d85242602e000000"),
    ObjectId("52ffc4a5d85242602e000001")
  ]
}
```

# Relationships (Cont.)

---

- **Modeling Referenced Relationships using MongoDB DBRefs.**

- The previous approach is called **Manual References** in which we manually store the referenced document's id inside other document.
- However, in cases where a document contains references from different collections, we can use **MongoDB DBRefs**.
- **Using DBRefs:** There are three fields in DBRefs :
  - **\$ref** – This field specifies the collection of the referenced document
  - **\$id** – This field specifies the `_id` field of the referenced document
  - **\$db** – This is an optional field and contains the name of the database in which the referenced document lies

```
{
  "_id": ObjectId("53402597d852426020000002"),
  "address": {
    "$ref": "address_home",
    "$id": ObjectId("534009e4d852427820000002"),
    "$db": "tutorialspoint"},
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin"
}
```



# Relationships (Cont.)

---

- **Modeling Referenced Relationships using MongoDB DBRefs. (Cont.)**
  - The following code dynamically looks in the collection specified by **\$ref** parameter:
    - `>var user = db.users.findOne({"name":"Tom Benzamin"}) >var dbRef = user.address`
    - `>db[dbRef.$ref].findOne({"_id":(dbRef.$id)})`


# References

---

- **Textbook references:**

- MongoDB: the definitive guide
- The Little MongoDB

- **Online Tutorials:**

- <https://code.tutsplus.com/tutorials/getting-started-with-mongodb-part-1--net-22879>
  - <https://www.tutorialspoint.com/mongodb/>
  - <https://docs.mongodb.com/manual/tutorial/>
- 

A decorative horizontal band at the top of the slide featuring a complex, overlapping pattern of green and yellow geometric shapes, creating a sense of depth and movement.

# Thanks...

A decorative triangular pattern in the bottom-left corner, composed of green and yellow geometric shapes that mirror the style of the top band.

**Eng .Hany Saad**  
**SD Dept.**  
**ITI – Assiut Branch**