Creating and Listing Indexes

```
db.collection.createIndex(
keys,
options
)

db.movies.createIndex(
    {year: 1}
)

db.movies.getIndexes()

db.theaters.createIndex(
    {theaterId : -1},
    {name : "myTheaterIdIndex"}
);

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

Then drop it:

```
db.movies.dropIndex(
     {title: 1}
)

db.theaters.dropIndexes()


db.collection.hideIndex(indexNameOrSpecification)
db.collection.unhideIndex(indexNameOrSpecification)
```

Type of Indexes

Default Indexes

Single-Key Indexes

Compound Indexes

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

Multikey Indexes (An index created on the fields of an array type)

```
db.movies.createIndex(
   {"languages" : 1}
)
```

```
db.movies.explain("executionStats").count(
   {"languages": "Cantonese"}
)
```

Text Indexes

```
db.users.createIndex(
   { name : "text"}
)
```

Indexes on Nested Documents

```
db.theaters.createIndex(
   { "location.address.zipcode" : 1}
)
```

```
db.theaters.createIndex(
   { "location" : 1}
)
```

Wildcard Indexes

```
db.products.createIndex(
   { "specifications.$**" : 1}
)
```

This query uses special wildcard characters (**`$**`**) to create indexes on the **`specifications`** field. It will create indexes on all the fields under **`specifications`**. If new nested fields are added in the future, they will be automatically indexed.

```
db.products.createIndex(
   { "$**" : 1 }
)
```

The preceding command creates indexes on all fields of all documents. Thus, all the new fields added to the documents will be indexed by default.

```
db.products.createIndex(
   { "$**" : 1 },
   {
      "wildcardProjection" : { "name" : 0 }
   }
)
```

The preceding query creates a wildcard index on all the fields of a collection, excluding the **`name`** field. To explicitly include the **`name`** field, excluding all the others, you can pass it with a value of **`1`**.

Properties of Indexes

Unique Indexes

Index properties are passed as an option to the **createdIndex** function. We will be looking at unique indexes, TTL (time to live) indexes, sparse indexes, and finally, partial indexes.

```
db.collection.createIndex(
   { field: type},
   { unique: true }
)
```

The **`{ unique: true }`** option is used to create a unique index.

Creating a Unique Index

In this exercise, you will enforce the uniqueness of the **`theaterId`** field in the **`theaters`** collection in the **`sample_mflix`** database:

```
db.theaters.createIndex(
   {theaterId : 1},
   {unique : true}
)
```

Now that the field has a unique index, try inserting a duplicate record, as follows:

```
db.theaters.insertOne(
   {theaterId : 1012}
);
```

TTL Indexes

TTL (or **Time to Live**) indexes put an expiry on documents.

```
db.collection.createIndex({ field: type}, { expireAfterSeconds: seconds })
```

Creating a TTL index using Mongo Shell

```
db.reviews.insert(
   {"reviewer" : "Eliyana A" , "movie" : "Cast Away","review" : "Interesting plot", "reviewDate" : new
Date() }
);
db.reviews.insert(
   {"reviewer" : "Zaid A" , "movie" : "Sully","review" : "Captivating", "reviewDate" : new Date() }
);

db.reviews.find().pretty();

db.reviews.createIndex(
   { reviewDate: 1},
   { expireAfterSeconds: 60 }
)
```

After 60 seconds, execute the **`find`** query again:

```
db.reviews.find().pretty();
```

Sparse Indexes

A sparse index will not have entries from the collection where the indexed field does not exist, and that is why this type of index is called sparse.

```
db.collection.createIndex({ field1 : type, field2: type2, ...}, { sparse: true })
```

Creating a Sparse Index Using Mongo Shell

```
db.reviews.createIndex(
   {review: 1},
   {sparse : true}
)

db.reviews.stats();
```

Insert a document that does not have the review field, as follows:

```
db.reviews.insert(
   {"reviewer" : "Jamshed A" , "movie" : "Gladiator"}
);

db.reviews.stats()
```

You can see that the size of the **review_1** index (highlighted) has not changed.

Now, insert a document that contains the review field:

```
db.reviews.insert(
   {"reviewer" : "Javed A" , "movie" : "The Pursuit of Happyness", "review": "Inspirational"}
);
```

db.reviews.stats()
As you can see, the sparse index size has changed.

Case-Insensitive Indexes

Exercise 9.07: Creating a Case-Insensitive Index Using the Mongo Shell

Perform a case-insensitive search and verify that the expected document is not returned:

```
db.movies.find(
   {"title" : "goodFEllas"},
   {"title" : 1}
)
```

```
db.movies.createIndex(
   {title: 1},
   {
      collation: {
         locale: 'en', strength: 2
      }
   }
)
```

Rerun the command in step 2 to confirm that the correct movie is returned:

```
db.movies.find(
   {"title" : "goodFEllas"}
).collation({ locale: 'en', strength: 2});
```