For multifield index → if array mongodb decomposes it & ~~each~~ each value is stored uniquely

Multikey Index → Any Index where one of the Indexed fields contains an array, The array can hold nested objects or other field types. only one.

In a compound field one of them can only be array.

Compound Index → Index on multiple fields.
Can be multikey index if it includes an array field.
Support queries that match on the prefix of the Index fields

The order of fields in compound Index matters
Follow this order → Equality, Sort & Range
ex.

The sort order of the field values in the Index matters.

⇒ Equality → Test exact matches, should be placed first, reduce query time, retrieves fewer docs.
⇒ Sort → Order of results determines, eliminates In memory sort, help when query results are sorted by more than 1 field and they mix sort orders
ex → some fields in ascending & others in descen

We can use projection with Indexing to improve the performance.

ex → createIndex({active: 1, birthdate: -1, name: 1, username: 1, address: -1})

Index improve performance, but too many can increase the write costs.

Deleting Index → ① Make sure it is unused
② Recreating takes time to check performance first by hiding
③ db.collection.hideIndex(<index>)
↓
Either by name, or key.

db.customers.hideIndex('active_1_birthdate_-1_name_1')
or
db. " " ({active: 1, birthdate: -1, name: 1});

) same way use dropIndex on name or keys to delete an index.

db.coll.dropIndexes() → to delete all with exception of the default.

or

db.coll.dropIndexes(['name1', 'name2', ...]);

Aggregation

An analysis & summary of data.
Stage → An aggregation operation performed on the data.

Aggregation pipeline → A series of stages completed one at a time in order.

pipeline → where data can be filtered, stored, grouped & transformed.

Stage

$match → filters the data that matches criteria
$group → Groups documents based on "
$ sort → Puts ~~data~~ documents in specific order.

```
db.collection.aggregate ([
{$stage-name: {<expression>}},
{$stage-name: "
]);
```

ex
```
$set {
    departmentname: { $concat: ["$firstname",
                      "$lastname"] } }.
```

If using match → place it as early as possible becoz it can use Indexes then reducing time.

$group → groups docs by key, output one doc for each unique Value of the group key

```
{
  $group:
    {_id : <exp—>  //group key
    <field> : { {<accumulator : <expression>}
  }
}  ,
{ group:
    {_id: "$city",
     totalzips: { $count: { }}}
  }
}
```

ex.

```
db.sightings.aggregate([
   {$match: {species_common: 'Eastern Bluebird'} },
   {$group: {_id: '$location.coor',
        number_of_sig: { $count: {} } } }]);
```

$sort → sorts all input docs & passes them through pipeline in sorted order.

$limit → limit the no of docs that are passed on to the next aggregation stage.

Aggregation stages

① $project, $set & $count

$project → Determines the output shape, similar to find() operation, should be the last stage to format the output

either include or exclude the fields

1 to include, 0 to exclude.

new value or new fields

ex

```
db.coll.aggregate([
   {$project: { state: 1, zip: 1, population: "$pop",
     _id: 0} }]);
```

$set -> Adds or modifies pipeline to change or add new other to be used in next ~~p~~ stages

$count -> total no of docs in the pipeline.

$out -> Writes documents that are returned by an aggregation pipeline into collection

Must be the last stage.
Creates a new collection if it does not exist.

if exists it replaces the existing collection with new data

```
$out {
        db: "<db>",
        coll : "<newcollection>"
   }.
```

ex -> { $out : "small_stats" }

Stages -> $match, $group, $sort, $limit, $project, $set, $count & $out.

Java Aggregation

Aggregation is superset of find