INTRODUCTION TO MONGODB INDEXES

In computers, an **index** is data outside of the data table that stores the location on disk (or address in memory, or cached data) of a fields inside a record. In other words, if you have a field **product=456**, and product is an indexed field, you can search for it quickly, because the computer knows that record is at, say, disk location FFFFFFX. If you don't have an index then MongoDB has to scan each record, which database programmers would call a **full table scan**.

MongoDB fields can be indexed in various ways. Here are some examples. To follow along, first open up the shell and create some data.

```
use products
db.createCollection("products")
db.products.insert({product: 123, count: 100})
db.products.insert({product: 456, count: 100})
db.products.insert({product: 789, count: 100})
db.products.insert({product: 1123, count: 100})
db.products.insert({product: 1456, count: 100})
```

Single Field Index

Create a **Single Field Index** in ascending order on the field **product**. The 1 means ascending order. -1 means descending order.

```
db.products.createIndex({product:1})
```

List the products. Notice that they are listed in the order created.

```
db.products.find()
{ "_id" : ObjectId("5cb8ccb664ae78fe855e9431"), "product" : 123, "count" : 96
}
{ "_id" : ObjectId("5cc340e1120707f8e83bb076"), "product" : 456, "count" :
100 }
{ "_id" : ObjectId("5cc340e1120707f8e83bb077"), "product" : 789, "count" :
100 }
{ "_id" : ObjectId("5cc340e1120707f8e83bb078"), "product" : 1123, "count" :
100 }
{ "_id" : ObjectId("5cc340e1120707f8e83bb079"), "product" : 1456, "count" :
100 }
```

Now list them in ascending order with the sort() operator.

```
db.products.find().sort({product: 1})
{ "_id" : ObjectId("5cb8ccb664ae78fe855e9431"), "product" : 123, "count" : 96
}
{ "_id" : ObjectId("5cc340e1120707f8e83bb076"), "product" : 456, "count" :
100 }
{ "_id" : ObjectId("5cc340e1120707f8e83bb077"), "product" : 789, "count" :
100 }
{ "_id" : ObjectId("5cc340e1120707f8e83bb078"), "product" : 1123, "count" :
100 }
{ "_id" : ObjectId("5cc340e1120707f8e83bb079"), "product" : 1456, "count" :
100 }
```

Sparse Index

A Sparse Index does not create an index when the document does not contain that field. So it does not needlessly index documents with blank values.

```
db.products.createIndex({product:-1},{sparse: true})
```

Compound Index

Fields can be sorted inside other fields. Here we divide out student population by age, first indexing their id and then their age.

```
db.createCollection("students")
db.students.insert({id: 123, age: 12})
db.students.insert({id: 456, age: 11})
db.students.insert({id: 789, age: 10})
```

Note that **id** is not the same as **_id**, which is an indexed field assigned to every document by default. As in:

```
{ "_id" : ObjectId("5cb8ccb664ae78fe855e9431"), "product" : 123, "count" : 96
}
```

Now create the index in descending order for the **age** and ascending order for the **id**.

```
db.students.createIndex({id: 1, age: -1})
```

List them, sorting first on id and then on age.

```
db.students.find().sort({id: 1, age: -1})
{ "_id" : ObjectId("5cc342e3120707f8e83bb07a"), "id" : 123, "age" : 12 }
{ "_id" : ObjectId("5cc342e3120707f8e83bb07b"), "id" : 456, "age" : 11 }
{ "_id" : ObjectId("5cc342e3120707f8e83bb07c"), "id" : 789, "age" : 10 }
```

Partial Indexes

You can create indexes on documents that only meet a certain filter.

Let's sort school kids but only if the are in highschool (or what you might call **college** in your country) , i.e., older than 14.

```
db.students.createIndex(
           { age: 1},
           { partialFilterExpression: { age: { $gt: 14}}}
         )
```

Question: does the performance matter when you sort in ascending order yet create the index in descending order? Not for single index indexes. It does for compound indexes as MongoDB must first sort documents on the first field listed and then the second, so says one explanation. You might wonder what difference does that make if it can sort the first field in either direction with equal ease. So why can't it do that for the second? That's a good question. Write us back at blogs@bmc.com if you think you have the answer to that.