Chapter 3: Sharding

Lab - Shard a Collection

Back to the Question

1. Adding a Second Shard

Once `m103-repl-2` is up and running, we exit the mongo shell and connect to mongos. We can add our new shard with the following command:

```
sh.addShard("m103-repl-2/192.168.103.100:27004")
```

COPY

The output of `sh.status()` should look something like this:

```
shards:
  {  "_id" : "m103-repl",  "host" : "m103-
repl/192.168.103.100:27001,192.168.103.100:27002,192.168.103.100:270
03",  "state" : 1 }
  {  "_id" : "m103-repl-2",  "host" : "m103-repl-
2/192.168.103.100:27004,192.168.103.100:27005,192.168.103.100:27006"
,  "state" : 1 }
```

COPY

2. Importing Data onto the Primary Shard

Importing data into a sharded cluster is always done with the mongos. We can import our dataset into `m103.products` with the following command:

```
mongoimport /dataset/products.json --port 26000 -u "m103-admin" \
-p "m103-pass" --authenticationDatabase "admin" \
--db m103 --collection products
```

COPY

We can verify that the entire dataset was imported with `count()`:

```
use m103
db.products.count()
```

COPY

This should return `516784`.

3. Sharding the Collection

We can look at all potential shard keys with `findOne()`:

```
use m103
db.products.findOne()
```

COPY

The output of this command should give us something like this:

```
{
    "_id" : ObjectId("573f706ff29313caab7d7395"),
    "sku" : 1000000749,
    "name" : "Gods And Heroes: Rome Rising - Windows [Digital
Download]",
    "type" : "Software",
    "regularPrice" : 39.95,
    "salePrice" : 39.95,
    "shippingWeight" : "0.01"
}
```

COPY

A trick to determining the correct shard key is process of elimination. We can rule out the potential shard keys which don't follow the rules of cardinality, frequency, rate of change, and query patterns.

We can rule out `_id` because it is rarely used in queries, and we would therefore be wasting an index by sharding on it. In addition, it is monotonically increasing, so it will continue to increase forever and cause hotspotting in our cluster.

We can rule out `type` because this field does not have high cardinality. In fact, it only has four possible values - we can see this by running the following command on `m103.products`:

```
db.products.distinct("type")
```

COPY

We can rule out `regularPrice` and `salePrice` because they are both subject to change and the shard key is immutable. If we sharded on one of these fields, any future updates to that field would result in an error.

We can rule out `shippingWeight` because every document in the collection must have the shard key, and not every document here has a `shippingWeight`.

From this, we have only two good shard keys:

- `name`
- `sku`

Both of these fields have **high cardinality**, **low frequency** and **non-monotonically increasing values**. They are also commonly used in queries.

The validation script will accept either solution.

Before we can shard, we must enable sharding on the `m103` database:

```
sh.enableSharding("m103")
```

COPY

Then, we must create an index on the shard key (in this example, `name`):

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

COPY

To shard on `name`, we specify the collection:

```
db.adminCommand( { shardCollection: "m103.products", key:
```

COPY

Choosing the Correct Shard Key

To choose a different shard key, the collection must be dropped and the dataset must be reimported.

From the mongos shell, we can drop the `products` collection with the following command:

COPY

```
use m103
db.products.drop()
```

Now we exit the mongos shell and reimport the dataset:

COPY

```
mongoimport /dataset/products.json --port 26000 -u "m103-admin" \
-p "m103-pass" --authenticationDatabase "admin" \
--db m103 --collection products
```

Now we can shard the collection again, because the dataset gets imported onto the primary shard.

Proceed to next section