Unique Constraints on Arbitrary Fields If you cannot use a unique field as the shard key or if you need to enforce uniqueness over multiple fields, you must create another *collection* to act as a "proxy collection". This collection must contain both a reference to the original document (i.e. its `ObjectId`) and the unique key.

If you must shard this "proxy" collection, then shard on the unique key using the *above procedure* (page 732); otherwise, you can simply create multiple unique indexes on the collection.

Process Consider the following for the "proxy collection:"

```
{
  "_id" : ObjectId("...")
  "email" ": "..."
}
```

The `_id` field holds the `ObjectId` of the *document* it reflects, and the `email` field is the field on which you want to ensure uniqueness.

To shard this collection, use the following operation using the `email` field as the *shard key*:

```
db.runCommand( { shardCollection : "records.proxy" ,
                 key : { email : 1 } ,
                 unique : true } );
```

If you do not need to shard the proxy collection, use the following command to create a unique index on the `email` field:

```
db.proxy.createIndex( { "email" : 1 }, { unique : true } )
```

You may create multiple unique indexes on this collection if you do not plan to shard the `proxy` collection.

To insert documents, use the following procedure in the *JavaScript shell*:

```
db = db.getSiblingDB('records');

var primary_id = ObjectId();

db.proxy.insert({
   "_id" : primary_id
   "email" : "example@example.net"
})

// if: the above operation returns successfully,
// then continue:

db.information.insert({
   "_id" : primary_id
   "email": "example@example.net"
   // additional information...
})
```

You must insert a document into the `proxy` collection first. If this operation succeeds, the `email` field is unique, and you may continue by inserting the actual document into the `information` collection.

See

The full documentation of: `createIndex()` and `shardCollection`.

Considerations