MONGO DB

For ensuring Uniqueness: you can create Unique Index and then insert.

use mca

```
db.mca.createIndex( { "_id": 1 }, { unique: true } )

db.mca.insert({'_id':1,'name':'4mca'})
```

When you try to insert with same _id

```
db.mca.insert({'_id':1,'name':'msc'})
```

error message:

WriteResult({ "nInserted" : 0, "writeError" : { "code" : 11000, "errmsg" : "E11000 duplicate key error }

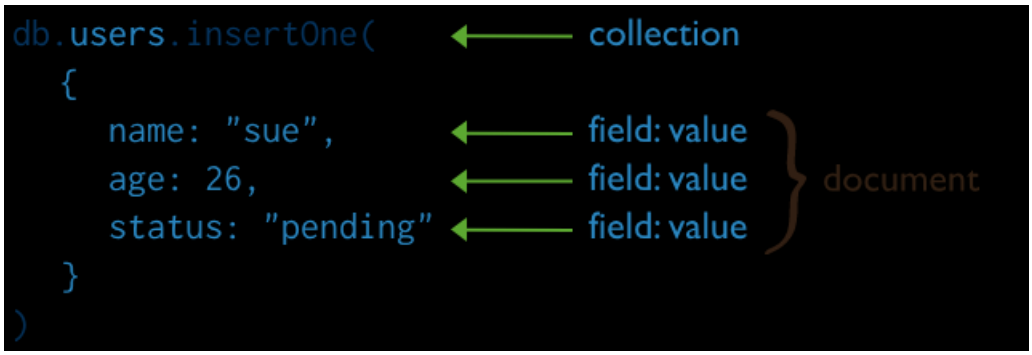
Create Operations

Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

MongoDB provides the following methods to insert documents into a collection:

- db.collection.insertOne()
- db.collection.insertMany()

  In MongoDB, insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.




```
db.products.insertMany( [
    { _id: 10, item: "large box", qty: 20 },
    { _id: 11, item: "small box", qty: 55 },
    { _id: 12, item: "medium box", qty: 30 }
  ] )
```

MongoDB find() Statements

SQL SELECT Statements	MongoDB find() Statements
SELECT * **FROM** people	db.people.find()
SELECT id, user_id, status **FROM** people	db.people.find({ }, { user_id: 1, status: 1 })
SELECT user_id, status **FROM** people	db.people.find({ }, { user_id: 1, status: 1, _id: 0 })
SELECT * **FROM** people **WHERE** status = "A"	db.people.find({ status: "A" })
SELECT user_id, status **FROM** people **WHERE** status = "A"	db.people.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })
SELECT * **FROM** people **WHERE** status != "A"	db.people.find({ status: { $ne: "A" } })
SELECT * **FROM** people **WHERE** status = "A" **AND** age = 50	db.people.find({ status: "A", age: 50 })
SELECT * **FROM** people **WHERE** status = "A" **OR** age = 50	db.people.find({ $or: [{ status: "A" } , { age: 50 }] })
SELECT * **FROM** people **WHERE** age > 25	db.people.find({ age: { $gt: 25 } })
SELECT * **FROM** people **WHERE** age < 25	db.people.find({ age: { $lt: 25 } })
SELECT * **FROM** people **WHERE** age > 25 **AND** age <= 50	db.people.find({ age: { $gt: 25, $lte: 50 } })
SELECT * **FROM** people **WHERE** user_id **like** "%bc%"	db.people.find({ user_id: /bc/ }) -or- db.people.find({ user_id: { $regex: /bc/ } })
SELECT * **FROM** people	db.people.find({ user_id: /^bc/ })

WHERE user_id **like** "bc%"	-or- db.people.find({ user_id: { $regex: /^bc/ } })
SELECT * **FROM** people **WHERE** status = "A" **ORDER BY** user_id **ASC**	db.people.find({ status: "A" }).sort({ user_id: 1 })
SELECT * **FROM** people **WHERE** status = "A" **ORDER BY** user_id **DESC**	db.people.find({ status: "A" }).sort({ user_id: -1 })
SELECT COUNT(*) **FROM** people	db.people.count() *or* db.people.find().count()
SELECT COUNT(user_id) **FROM** people	db.people.count({ user_id: { $exists: **true** } }) *or* db.people.find({ user_id: { $exists: **true** } }).count()
SELECT COUNT(*) **FROM** people **WHERE** age > 30	db.people.count({ age: { $gt: 30 } }) *or* db.people.find({ age: { $gt: 30 } }).count()
SELECT DISTINCT(status) **FROM** people	db.people.aggregate([{ $group : { _id : "$status" } }]) or, for distinct value sets that do not exceed the BSON size limit db.people.distinct("status")
SELECT * **FROM** people **LIMIT** 1	db.people.findOne() *or* db.people.find().limit(1)
SELECT * **FROM** people **LIMIT** 5 SKIP 10	db.people.find().limit(5).skip(10)
SELECT * **FROM** people **WHERE** status = "A"	db.people.find({ status: "A" }).explain()

SQL Update Statements

SQL Update Statements	MongoDB

UPDATE people **SET** status = "C" **WHERE** age > 25	db.people.updateMany({ age: { $gt: 25 } }, { $set: { status: "C" } })
UPDATE people **SET** age = age + 3 **WHERE** status = "A"	db.people.updateMany({ status: "A" } , { $inc: { age: 3 } })

Specify Conditions Using Query Operators

A query filter document can use the query operators to specify conditions in the following form:

```
{ <field1>: { <operator1>: <value1> }, ... }
```

The following example retrieves all documents from the inventory collection where status equals either "A"or "D":

```
db.inventory.find( { status: { $in: [ "A", "D" ] } } )
```

NOTE
Although you can express this query using the $or operator, use the $in operator rather than the $oroperator when performing equality checks on the same field.

The operation corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status in ("A", "D")
```

Refer to the Query and Projection Operators document for the complete list of MongoDB query operators.

Specify AND Conditions

A compound query can specify conditions for more than one field in the collection's documents. Implicitly, a logical AND conjunction connects the clauses of a compound query so that the query selects the documents in the collection that match all the conditions.

The following example retrieves all documents in the inventory collection where the status equals "A" **and**qty is less than ($lt) 30:

```
db.inventory.find( { status: "A", qty: { $lt: 30 } } )
```

The operation corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

See comparison operators for other MongoDB comparison operators.

Specify OR Conditions

Using the $or operator, you can specify a compound query that joins each clause with a logical OR conjunction so that the query selects the documents in the collection that match at least one condition.

The following example retrieves all documents in the collection where the status equals "A" **or** qty is less than ($lt) 30:

```
db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 } } ] } )
```

The operation corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status = "A" OR qty < 30
```