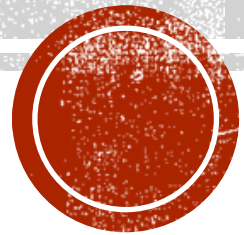
MONGODB QUERY RECAP



QUICK START

The mongo shell is an interactive JavaScript interface to MongoDB

- Switch Database

db refers to your current database

```
db
```

The operation should return test, which is the default database.

To switch databases, type: use <databasename>

```
use mydatabase
```

QUICK START

- Populate a collection

```
db.inventory.insertMany([
        { item: "journal", qty: 25, status: "A", size: { h: 14, w: 21, uom: "cm" }, tags: [ "blank", "red" ] },
        { item: "notebook", qty: 50, status: "A", size: { h: 8.5, w: 11, uom: "in" }, tags: [ "red", "blank" ] },
        { item: "paper", qty: 10, status: "D", size: { h: 8.5, w: 11, uom: "in" }, tags: [ "red", "blank", "plain" ] }
]);
```

The operation returns a document that contains the acknowledgement indicator and an array that contains the _id of each successfully inserted documents.

VIEWS

- A queryable object whose contents are defined by an **aggregation** pipeline on other **collections** or **views**.
- MongoDB does not persist the view contents to disk. A view's content is **computed on-demand**.
- **Read-only** views from existing collections or other views. E.g.:
  - excludes private or confidential data from a collection of employee data
  - adds computed fields from a collection of metrics
  - joins data from two different related collections

```
db.runCommand( {
  create: <view>, viewOn: <source>, pipeline: <pipeline>, collation: <collation> } )
```

- Restrictions
  - immutable Name
  - you can modify a view either by dropping and recreating the view or using the *collMod* command

CRUD OPERATIONS

INTRODUCTION

- **C**reate
- **R**ead
- **U**pdate
- **D**elete

```
db.users.insertOne(                    ← collection
   {
      name: "sue",                     ← field: value
      age: 26,                         ← field: value      } document
      status: "pending"                ← field: value
   }
)
```

```
db.users.find(                         ← collection
   { age: { $gt: 18 } },               ← query criteria
   { name: 1, address: 1 }             ← projection
).limit(5)                             ← cursor modifier
```

```
db.users.updateMany(                   ← collection
   { age: { $lt: 18 } },               ← update filter
   { $set: { status: "reject" } }      ← update action
)
```

```
db.users.deleteMany(                   ← collection
   { status: "reject" }                ← delete filter
)
```

CREATE

- New data needs to be **inserted into** the database.
  - each SQL tuple corresponds to a MongoDB document
- The primary key `_id` is automatically added if the `_id` field is not specified.

MySQL clause	MongoDB operator
INSERT INTO	insertOne()

INSERT ONE DOCUMENT

MySQL clause	MongoDB operator
INSERT INTO	insertOne()

```
INSERT INTO people(user_id,
      age,
      status)
VALUES ("bcd001",
       45,
       "A")
```

```
db.people.insertOne(
   {
     user_id: "bcd001",
     age: 45,
     status: "A"
   }
)
```

INSERT MANY DOCUMENTS

MySQL clause	MongoDB operator
INSERT INTO	insertMany()

```
INSERT INTO
people(user_id,
      age,
      status)
VALUES
     ("abc123", 30, "A"),
     ("abc456", 40, "A"),
     ("abc789", 50, "B");
```

```
db.products.insertMany( [
     { user_id: "abc123", age: 30, status: "A"},
     { user_id: "abc456", age: 40, status: "A"},
     { user_id: "abc789", age: 50, status: "B"}
  ] )
```

READ

- Most of the operations available in SQL language can be expressed in MongoDB language

MySQL clause	MongoDB operator
SELECT	find()
SELECT LIMIT 1	findOne()

- Select one document that satisfies the specified query criteria
  - if multiple documents satisfy the query, it returns the first one according to the natural order which reflects the order of documents on the disk

FILTER DOCUMENTS

- Select documents

```
db.<collection name>.find( {<conditions>}, {<fields of interest>} )
```

- Select the documents satisfying the specified conditions and specifically only the fields specified in fields of interest
  - `<conditions>` are optional
    - conditions take a document with the form:

      `{field1 : <value>, field2 : <value> ... }`

    - conditions may specify a value or a regular expression

PROJECT FIELDS

- Select documents

```
db.<collection name>.find( {<conditions>}, {<fields of interest>} )
```

- Select the documents satisfying the specified conditions and specifically only the fields specified in fields of interest
  - `<fields of interest>` are optional
    - projections take a document with the form:

      `{field1 : <value>, field2 : <value> ... }`
    - 1/true to include the field, 0/false to exclude the field

FIND() OPERATOR

MySQL clause	MongoDB operator
SELECT	find()
WHERE	find({<WHERE CONDITIONS>})

```
SELECT user_id, status
FROM people
WHERE status = "A"
```

```
db.people.find(
   { status: "A" },
   { user_id: 1,
     status: 1,
     _id: 0
   }
)
```

Where Condition

Selection fields

- By default, the _id field is shown.
- To remove it from visualization use: _id: 0

FIND() OPERATOR

MySQL clause	MongoDB operator
SELECT	find()
WHERE	find({<WHERE CONDITIONS>})

	db.people.find(**{"address.city":"Rome" }** **)**

```
{ _id: "A",
  address: {
        street: "Via Torino",
        number: "123/B",
        city: "Rome",
        code: "00184"
    }
}
```

nested document

CONDITIONAL OPERATORS (AND)

MySQL	MongoDB	Description
AND	,	Both verified

SELECT * FROM people WHERE status = "A" AND age = 50	**db.people.find({ status: "A", age: 50 })**

CONDITIONAL OPERATORS (OR)

MySQL	MongoDB	Description
AND	,	Both verified
OR	$or	At least one verified

```
SELECT *
FROM people
WHERE status = "A"
OR age = 50
```

```
db.people.find(
{ $or:
  [ { status: "A" } ,
    { age: 50 }
  ]
}
)
```

EXAMPLES FILTER OPERATORS

```
db.people.find({ age: { $gt: 25, $lte: 50 } })
```

- Age greater than 25 and less than or equal to 50
  - returns all documents having **age > 25 and age <= 50**

```
db.people.find({$or:[{status: "A"},{age: 55}]})
```

- Status = "A" or age = 55
  - returns all documents having **status="A" or age=55**

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

- Status = "A" or status = B
  - returns all documents where the **status** field value is **either "A" or "B"**

QUERY EMBEDDED DOCUMENTS

- Select all documents where the field size equals the document { h: 14, w: 21, uom: "cm" }

```
db.inventory.find( { size: { h: 14, w: 21, uom: "cm" } } )
```

- To specify a query condition on fields in an embedded/nested document, use dot notation

```
db.inventory.find( { "size.uom": "in" } )
```

- Using query operators

```
db.inventory.find( { "size.h": { $lt: 15 } } )
```

QUERY ARRAYS (1)

- Query for all documents where the field tags value is an array with exactly two elements

```
db.inventory.find( { tags: ["red", "blank"] } )
db.inventory.find( { tags: { $all: ["red", "blank"] } } )
```

- Query for all documents where tags is an array that contains the string "red" as one of its elements

```
db.inventory.find( { tags: "red" } )
```

- Query an Array with Compound Filter Conditions on the Array Elements

```
db.inventory.find( { dim_cm: { $gt: 15, $lt: 20 } } )
```

QUERY ARRAYS (2)

- Query for an Array Element that Meets Multiple Criteria

```
db.inventory.find( { dim_cm: { $elemMatch: { $gt: 22, $lt: 30 } } } )
```

- Query for an Element by the Array Index Position

```
db.inventory.find( { "dim_cm.0": { $gt: 25 } } )
```

- Query an Array by Array Length

```
db.inventory.find( { "tags": { $size: 3 } } )
```

QUERY NULL OR MISSING FIELDS

- Equality Filter

```
db.inventory.find( { item: null } )
```

- Type Check

```
db.inventory.find( { item : { $type: 10 } } )
```

- Existence Check

```
db.inventory.find( { item : { $exists: false } } )
```

CURSOR

- `db.collection.find()` **gives back a cursor. It can be used to iterate over the result or as input for next operations**
- **E.g.,**
  - `cursor.sort()`
  - `cursor.count()`
  - `cursor.forEach() //shell method`
  - `cursor.limit()`
  - `cursor.max()`
  - `cursor.min()`
  - `cursor.pretty()`

CURSOR EXAMPLES

```
db.people.find({ status: "A"}).count()
```

- Select documents with status="A" and count them

```
db.people.find({ status: "A"}).forEach(
            function(myDoc) {
             print( "user: "+myDoc.name );
})
```

- `forEach` applies a JavaScript function to apply to each document from the cursor
  - Select documents with status="A" and print the document name