MongoDB Query Language Documentation

This document provides an overview of various query operations in MongoDB, including querying documents, embedded documents, arrays, projections, handling null/missing fields, query timeouts, cursor iteration, and snapshot queries. Examples are based on using the PyMongo Python driver and the MongoDB Atlas UI, but the concepts apply broadly to other drivers and interfaces.

---

1. Querying Documents

In MongoDB, you can retrieve documents from a collection by specifying a query predicate. An empty predicate ({}) returns all documents.

1.1 Select All Documents

```
cursor = db.inventory.find({})
```

Equivalent SQL:

```
SELECT * FROM inventory
```

1.2 Equality Conditions

Specify conditions using key-value pairs.

```
cursor = db.inventory.find({"status": "D"})
```

Equivalent SQL:

```
SELECT * FROM inventory WHERE status = "D"
```

1.3 Query Operators

Using $in Operator

Select documents where status is either "A" or "D".

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

Equivalent SQL:

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

Combining Conditions with AND

MongoDB implicitly uses an AND between multiple field conditions.

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

Equivalent SQL:

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

Combining Conditions with OR

Use the $or operator to match documents that satisfy at least one condition.

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

Equivalent SQL:

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

AND as well as OR Conditions

Mixing both implicit AND and explicit OR conditions:

```
cursor = db.inventory.find({
   "status": "A",
   "$or": [
      {"qty": {"$lt": 30}},
      {"item": {"$regex": "^p"}}
   ]
})
```

Equivalent SQL:

```
SELECT * FROM inventory WHERE status = "A" AND (qty < 30 OR item LIKE "p%")
```

---

2. Querying Embedded/Nested Documents

MongoDB allows querying of documents embedded within documents. Use dot notation to reference nested fields.

2.1 Query on a Nested Field

Equality Match

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

Using Query Operators

For example, query nested field h in size with a comparison operator:

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

Compound Conditions on Nested Fields

```
cursor = db.inventory.find({
   "size.h": {"$lt": 15},
   "size.uom": "in",
   "status": "D"
})
```

Matching an Entire Embedded Document

To match a document exactly (including field order), use an ordered structure like bson.son.SON:

```
from bson.son import SON
cursor = db.inventory.find({
   "size": SON([("h", 14), ("w", 21), ("uom", "cm")])
})
```

Warning: Comparisons on embedded documents require an exact match—including field order. Reordering the keys will not match.

---

3. Querying Arrays

MongoDB provides several methods for querying array fields.

3.1 Matching an Entire Array

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

This finds documents where the tags array is exactly ["red", "blank"] in that order.

3.2 Matching Array Elements

Querying for a Single Element

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

This returns documents where "red" is one of the elements in the tags array.

Using $all Operator

To match arrays that contain all specified elements, regardless of order:

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

Querying with Comparison Operators

For example, query an array field dim_cm for elements greater than a given value:

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

3.3 Compound Conditions on Array Elements

Without $elemMatch

The following matches documents where any element meets one condition and some (possibly different) element meets another:

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

With $elemMatch

To ensure a single array element satisfies all conditions:

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

Array Index Query

You can target an element at a specific index (zero-based indexing):

```
cursor = db.inventory.find({"dim_cm.1": {"$gt": 25}})
```

Query by Array Length

Use the $size operator to match arrays with a specific number of elements:

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

---

4. Querying Arrays of Embedded Documents

When your array contains documents, you can query on specific fields within those embedded documents.

4.1 Matching an Entire Embedded Document

```
from bson.son import SON
cursor = db.inventory.find({
   "instock": SON([("warehouse", "A"), ("qty", 5)])
})
```

Note: The order of fields matters.

4.2 Query Conditions on Fields Within Array Documents

Using Dot Notation

Query for documents where at least one embedded document has qty less than or equal to 20:

```
cursor = db.inventory.find({"instock.qty": {"$lte": 20}})
```

Using an Array Index

Target the first element of the array:

```
cursor = db.inventory.find({"instock.0.qty": {"$lte": 20}})
```

Compound Conditions with $elemMatch

To require that one embedded document matches multiple criteria:

```
cursor = db.inventory.find({
   "instock": {"$elemMatch": {"qty": 5, "warehouse": "A"}}
})
```

Another example with range conditions:

```
cursor = db.inventory.find({
   "instock": {"$elemMatch": {"qty": {"$gt": 10, "$lte": 20}}}
})
```

Combining Separate Conditions

The following matches documents where some document in the array has qty equal to 5 and another (or the same) document has warehouse equal to “A”:

```
cursor = db.inventory.find({
   "instock.qty": 5,
   "instock.warehouse": "A"
})
```

---

5. Projection: Returning Specific Fields

By default, MongoDB returns all fields in matching documents. You can specify a projection document to include or exclude specific fields.

5.1 Including Specific Fields

To return only the item and status fields (plus the default _id):

```
cursor = db.inventory.find({"status": "A"}, {"item": 1, "status": 1})
```

Equivalent SQL:

```
SELECT _id, item, status FROM inventory WHERE status = "A"
```

5.2 Excluding the _id Field

```
cursor = db.inventory.find({"status": "A"}, {"item": 1, "status": 1,
"_id": 0})
```

Equivalent SQL:

```
SELECT item, status FROM inventory WHERE status = "A"
```

5.3 Excluding Specific Fields

To exclude status and instock fields:

```
cursor = db.inventory.find({"status": "A"}, {"status": 0, "instock":
0})
```

Note: Except for _id, you cannot mix inclusion and exclusion in the same projection.

5.4 Projecting Fields in Embedded Documents

Return only the uom field within the embedded size document:

```
cursor = db.inventory.find({"status": "A"}, {"item": 1, "status": 1,
"size.uom": 1})
```

Alternatively, using nested projection syntax:

```
cursor = db.inventory.find({"status": "A"}, {"item": 1, "status": 1,
"size": {"uom": 1}})
```

5.5 Projection on Arrays

Projecting Specific Fields from Array Elements

For example, return only the qty field from documents embedded in the instock array:

```
cursor = db.inventory.find({"status": "A"}, {"item": 1, "status": 1,
"instock.qty": 1})
```

Using Projection Operators for Arrays

To return the last element in the instock array using $slice:

```
cursor = db.inventory.find(
   {"status": "A"},
   {"item": 1, "status": 1, "instock": {"$slice": -1}}
)
```

Projection with Aggregation Expressions

You can even use aggregation expressions to modify returned fields. For example, overriding status and creating new fields:

```
db.inventory.find(
  { },
  {
    _id: 0,
    item: 1,
    status: {
      $switch: {
        branches: [
          { case: { $eq: [ "$status", "A" ] }, then: "Available" },
          { case: { $eq: [ "$status", "D" ] }, then: "Discontinued" }
        ],
        default: "No status found"
      }
    },
    area: {
      $concat: [
        { $toString: { $multiply: [ "$size.h", "$size.w" ] } },
        " ",
        "$size.uom"
      ]
    },
    reportNumber: { $literal: 1 }
  }
)
```

This returns documents with computed fields such as area and a new reportNumber.

---

6. Querying for Null or Missing Fields

MongoDB treats null values in queries differently depending on the operator used.

6.1 Using Equality Filter

Using { field: None } (or null in MongoDB shell) returns documents where the field is either null or does not exist.

```
cursor = db.inventory.find({"item": None})
```

6.2 Non-Equality Filter

To query for fields that exist and are not null, use the $ne operator:

```
cursor = db.inventory.find({"item": {"$ne": None}})
```

6.3 Type Check

Use $type to match documents where the field is exactly BSON type Null (type 10):

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

6.4 Existence Check

To match documents that do not contain a field:

```
cursor = db.inventory.find({"item": {"$exists": False}})
```

---

7. Query Timeouts

You can set a timeout to prevent long-running queries from affecting system performance.

7.1 Setting a Query Time Limit

Use the maxTimeMS() method to specify a query timeout in milliseconds. If the query exceeds this time limit, MongoDB stops the query and no results are returned.

Alternatively, you can set a global default timeout using the defaultMaxTimeMS cluster parameter.

---

8. Iterating a Cursor

The find() method returns a cursor—not an array of documents. You must iterate the cursor to access documents.

8.1 Manual Iteration

Assign the cursor to a variable to prevent automatic iteration (mongosh prints the first 20 documents by default):

```
var myCursor = db.users.find({ type: 2 });
while (myCursor.hasNext()) {
  printjson(myCursor.next());
}
```

8.2 Using forEach()

```
var myCursor = db.users.find({ type: 2 });
myCursor.forEach(printjson);
```

8.3 Converting Cursor to an Array

You can convert the cursor to an array using the toArray() method:

```
var documentArray = db.inventory.find({ type: 2 }).toArray();
var myDocument = documentArray[3];
```

Note: The toArray() method loads all documents into memory.

8.4 Accessing by Index

Some drivers allow you to use array index notation directly on the cursor (this internally calls toArray()):

```
var myDocument = db.users.find({ type: 2 })[1];
```

---

9. Cursor Behaviors

9.1 Cursors Opened Within a Session

Starting in MongoDB 5.0, cursors created in a session will close when the server session ends, times out, or when the client exhausts the cursor. The default session timeout is 30 minutes.

9.2 Cursors Opened Outside a Session

Cursors not associated with a session will automatically close after 10 minutes of inactivity or when fully iterated. To prevent this, use the noCursorTimeout() option:

```
var myCursor = db.users.find().noCursorTimeout();
```

Remember to close the cursor manually or exhaust it.

9.3 Batch Size and GetMore Operations

Query results are returned in batches. The initial batch for find() and aggregate() is 101 documents by default. As you iterate, the cursor issues getMore operations to fetch subsequent batches (up to a 16 MiB message size limit). You can check the remaining documents in the current batch with objsLeftInBatch().

---

10. Snapshot Queries and Read Concern

MongoDB supports snapshot queries using the read concern "snapshot", which lets you read data as it appeared at a single point in time.

10.1 Use Case for Snapshot Queries

Snapshot queries are useful when you need to:

- Run multiple related queries that must reflect the same point in time.
- Ensure consistency in reporting, especially in systems with ongoing writes.

10.2 Example: Multiple Collections

An animal shelter might count adoptable pets from different collections while ensuring consistency:

```
db = client.pets
with client.start_session(snapshot=True) as s:
   adoptablePetsCount = (
      db.cats.aggregate(
         [{"$match": {"adoptable": True}}, {"$count": 
"adoptableCatsCount"}],
         session=s
      ).next()["adoptableCatsCount"]
   )
   adoptablePetsCount += (
      db.dogs.aggregate(
         [{"$match": {"adoptable": True}}, {"$count": 
"adoptableDogsCount"}],
            session=s
        ).next()["adoptableDogsCount"]
    )
print(adoptablePetsCount)
```

10.3 Example: Daily Sales Count

Ensure that a query counting daily sales does not include new sales occurring during the query execution:

```
db = client.retail
with client.start_session(snapshot=True) as s:
    dailySales = (
        db.sales.aggregate(
            [
                {
                    "$match": {
                        "$expr": {
                            "$gt": [
                                "$saleDate",
                                {
                                    "$dateSubtract": {
                                        "startDate": "$$NOW",
                                        "unit": "day",
                                        "amount": 1
                                    }
                                }
                            ]
                        }
                    }
                },
                {"$count": "totalDailySales"}
            ],
            session=s
        ).next()["totalDailySales"]
    )
```

```
print(dailySales)
```

Note: Sessions using "snapshot" read concern must complete within the WiredTiger history retention period (default 300 seconds) unless the retention period is increased.

Conclusion

This guide provides a comprehensive overview of the MongoDB Query Language features, including basic queries, complex operators, array and embedded document queries, projections, and cursor management. MongoDB’s flexible querying capabilities allow you to construct queries that closely resemble SQL while leveraging the document model’s power and flexibility.

By understanding these concepts and examples, you can effectively build robust applications that interact with MongoDB data.