# MongoDB

By

Srikanth Pragada

## What is MongoDB?

- ❑ MongoDB is a NoSQL database.
- ❑ It stores data in the form of **documents**.
- ❑ A document is a data structure composed of field and value pairs.
- ❑ MongoDB documents are similar to **JSON** objects.
- ❑ The values of fields may include other documents, arrays, and arrays of documents.

## Advantages of Document Database

- ❑ Documents correspond to native data types in many programming languages.
- ❑ Embedded documents and arrays reduce need for expensive joins.
- ❑ Intuitive Data Model
- ❑ Flexible Schema
- ❑ Universal JSON documents
- ❑ Distributed Scalable Database – Horizontal Scaling

## Advantages of MongoDB

- ❑ Open Source and Free (Community Ed)
- ❑ High performance as information is embedded in one document
- ❑ Rich Query Language
- ❑ High Availability using Replication
- ❑ Multiple Storage Engine
- ❑ Easy horizontal scale-out with sharding
- ❑ Simple installation – Atlas and on-premises editions
- ❑ Technical support and documentation

# MongoDB Atlas

- ❑ Atlas is a cloud database service.
- ❑ Provide FREE account, which allows you to access MongoDB databases.

## MongoDB in Community Server

- It is possible to install MongoDB Community Server in local system as a standalone database server.
- It is available on Windows, Linux and MacOS.

## Compass

- MongoDB Compass is a powerful GUI for querying, aggregating, and analyzing your MongoDB data in a visual environment.
- Compass is free to use and can be run on macOS, Windows, and Linux.

https://www.mongodb.com/try/download/compass

## MongoDB Shell (mongosh)

- ❑ The MongoDB Shell, mongosh, is a fully functional JavaScript and Node.js 16.x REPL environment for interacting with MongoDB deployments.
- ❑ You can use the MongoDB Shell to test queries and operations directly with your database.

https://www.mongodb.com/try/download/shell

## Connecting to MongoDB

- **mongodb://localhost:27017** is used to connect to MongoDB running on current system with no authentication.
- **mongodb://localhost:27017/demo** connects to database **demo.**
- **mongodb://srikanth:test@localhost:27017** connects using username **srikanth** and password **test.**

# MongoDB Shell Commands

| Command | Description |
|---|---|
| use <db> | Switches to given database |
| db | Shows current database |
| db.createCollection(name) | Creates a new collection |
| db.getCollectionNames() | Returns list of collection names |
| db.collection.drop() | Drops collection |
| show dbs | Lists all databases |
| console.clear() | Clears the console |

## Collection

- ❑ MongoDB stores documents in collections.
- ❑ Collections are analogous to tables in relational databases.

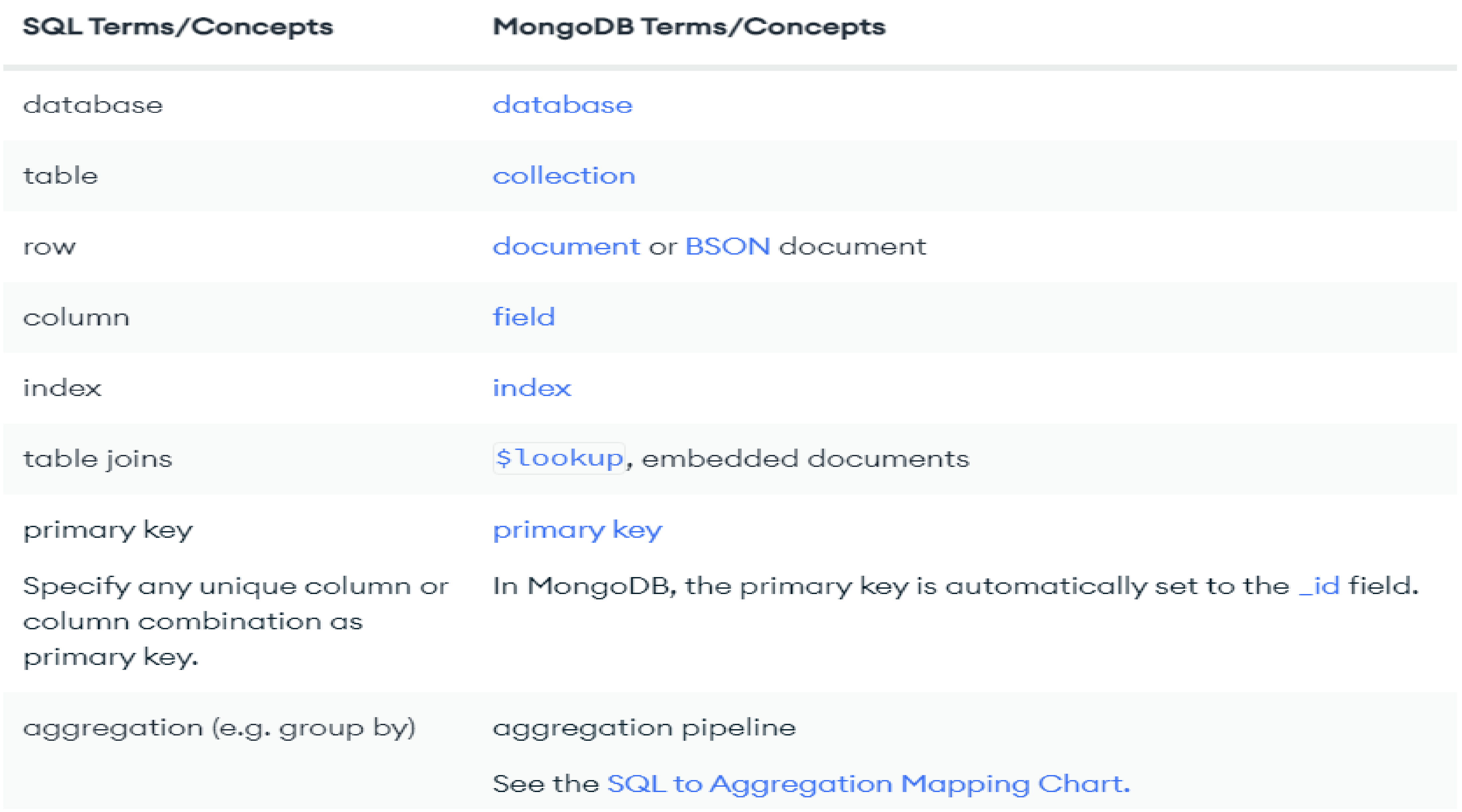
## Document

- MongoDB stores data records as BSON documents.
- Documents are analogous to rows in relational databases.
- BSON is a binary representation of **JSON** documents, though it contains more data types than JSON.
- MongoDB documents are composed of field-and-value pairs.
- The value of a field can be any of the **BSON data types**, including other documents, arrays, and arrays of documents.

```
{
   field1: value1,
   field2: value2,
   ...
   fieldN: valueN
}
```

```
{
  name: {first: "Alan", last: "Cooper"},
  birth: "2000-10-20",
  langs: ["Java", "Python"],
  exp_years: 5
}
```

# SQL vs. MongoDB



| SQL Terms/Concepts | MongoDB Terms/Concepts |
|---|---|
| database | database |
| table | collection |
| row | document or BSON document |
| column | field |
| index | index |
| table joins | `$lookup`, embedded documents |
| primary key<br>Specify any unique column or column combination as primary key. | primary key<br>In MongoDB, the primary key is automatically set to the _id field. |
| aggregation (e.g. group by) | aggregation pipeline<br>See the SQL to Aggregation Mapping Chart. |

# Querying

```
db.collection.find(query, projection, options)
```

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

```
{ <field1>: 1, <field2> : 0}
```

# Projection

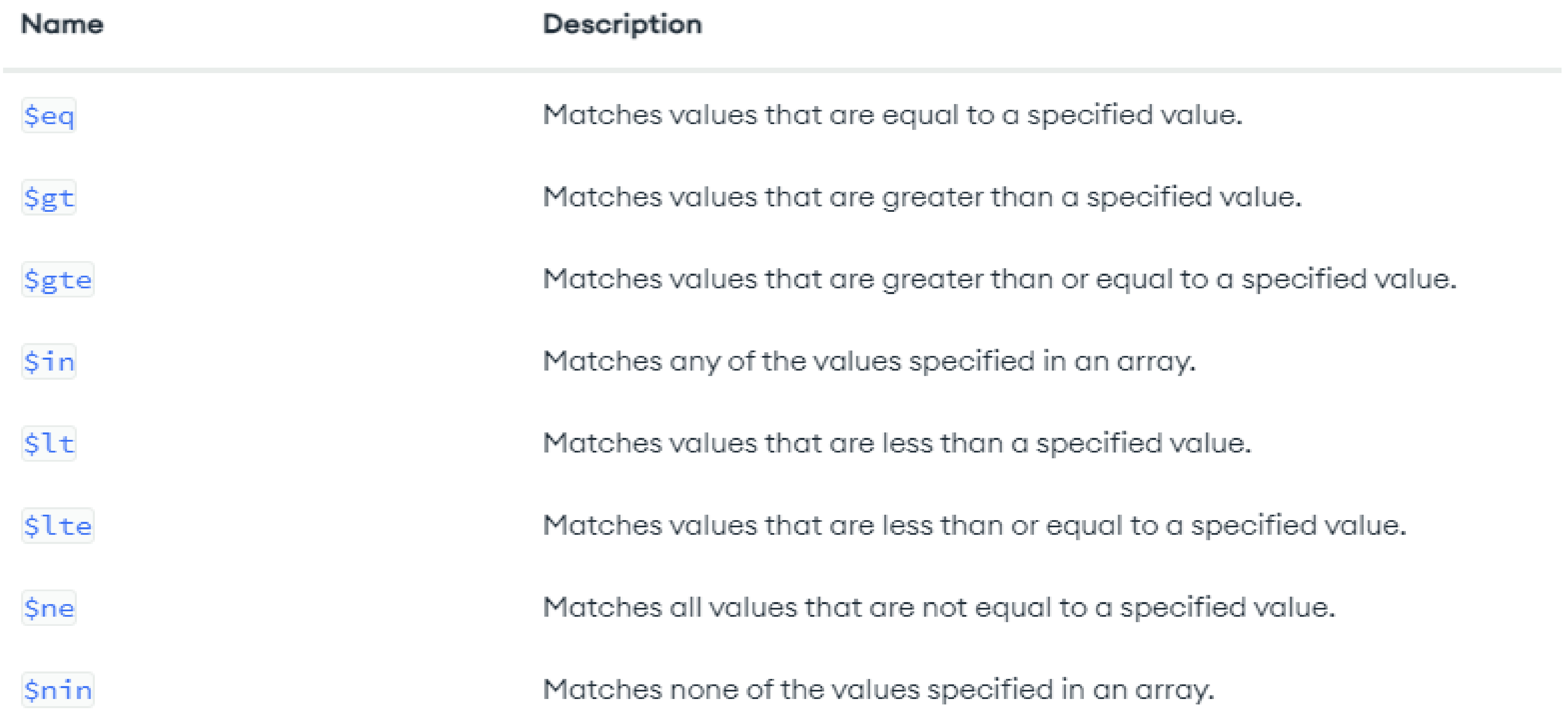
- ❑ By default, queries in MongoDB return all fields in matching documents.
- ❑ To limit the amount of data that MongoDB sends to applications, you can include a projection document to specify or restrict fields to return.
- ❑ A projection can explicitly include several fields by setting the <field> to 1 in the projection document.
- ❑ You can remove a field from the results by setting it to 0 in the projection.
- ❑ Instead of listing the fields to return in the matching document, you can use a projection to exclude specific fields.

```
{ title: 1, price: 1 }
{ _id: 0 }
{ title : 1, "rating.goodreads": 1 }
```

## Query Operator Syntax

```
{  field  : {$operator : value}}
```

## Query Operators

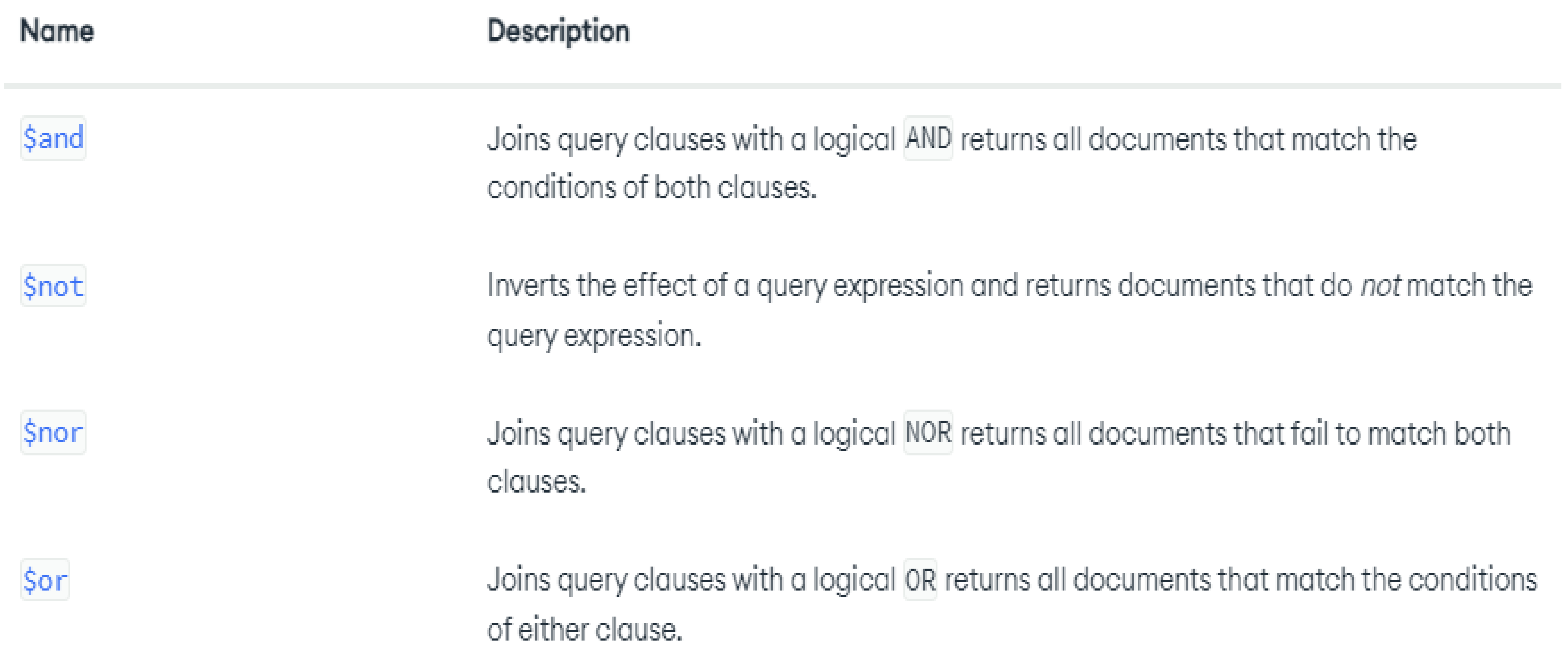


| Name | Description |
|---|---|
| `$eq` | Matches values that are equal to a specified value. |
| `$gt` | Matches values that are greater than a specified value. |
| `$gte` | Matches values that are greater than or equal to a specified value. |
| `$in` | Matches any of the values specified in an array. |
| `$lt` | Matches values that are less than a specified value. |
| `$lte` | Matches values that are less than or equal to a specified value. |
| `$ne` | Matches all values that are not equal to a specified value. |
| `$nin` | Matches none of the values specified in an array. |

## Logical Operator Syntax

```
{  logicaloperator : [ {expression}, {expression}, … ] }
```

## Logical Operators



| Name | Description |
|---|---|
| `$and` | Joins query clauses with a logical `AND` returns all documents that match the conditions of both clauses. |
| `$not` | Inverts the effect of a query expression and returns documents that do *not* match the query expression. |
| `$nor` | Joins query clauses with a logical `NOR` returns all documents that fail to match both clauses. |
| `$or` | Joins query clauses with a logical `OR` returns all documents that match the conditions of either clause. |

## Query Examples

```
{"cover" : "p", "price" : { $lt : 500 } }

{"cover": "p", $or: [{ "price" : {$gt: 500}}, {"rating.goodreads": 4.0 }]}

{"price": null}                    // Whether field is null

{"cover" : { $exists: false } } )  // Whether field exists

{"cover" : {$in : ['p', 'h']}})

{"title" : {$in : [/^B/]}}         // Regular expression

{$or: [ { "cover" : "h" }, { "price" : {$lt: 500 }}] }

{"reviews.1.rating" : 1}           // Access array (reviews) element by index
```

## Sorting Examples

```
db.books.find().sort({"price":-1}) // Descending order by price

db.books.find().sort({"cover": -1, "price": 1})
```

# Cursor

- ❑ Method find() returns a cursor with selected documents.
- ❑ Cursor provides the following method to alter the behavior of the cursor.

- `cursor.allowDiskUse()`
- `cursor.allowPartialResults()`
- `cursor.batchSize()`
- `cursor.close()`
- `cursor.isClosed()`
- `cursor.collation()`
- `cursor.comment()`
- `cursor.count()`
- `cursor.explain()`
- `cursor.forEach()`
- `cursor.hasNext()`
- `cursor.hint()`
- `cursor.isExhausted()`
- `cursor.itcount()`
- `cursor.limit()`
- `cursor.map()`
- `cursor.max()`
- `cursor.maxTimeMS()`
- `cursor.min()`
- `cursor.next()`
- `cursor.noCursorTimeout()`
- `cursor.objsLeftInBatch()`
- `cursor.pretty()`
- `cursor.readConcern()`
- `cursor.readPref()`
- `cursor.returnKey()`
- `cursor.showRecordId()`
- `cursor.size()`
- `cursor.skip()`
- `cursor.sort()`
- `cursor.tailable()`
- `cursor.toArray()`

## Cursor Methods

```
db.books.find().count()
db.books.find({price : {$gt : 500}}).count()
db.books.find().sort({price : -1}).limit(2)
db.books.find().skip(2).limit(2)
db.books.find().max( {price : 500}).hint ( {price : 1})  // use index on price
```

## Aggregating Examples

An aggregation pipeline consists of one or more stages that process documents:

- ✓ Each stage performs an operation on the input documents. For example, a stage can filter documents, group documents, and calculate values.
- ✓ The documents that are output from a stage are passed to the next stage.
- ✓ An aggregation pipeline can return results for groups of documents. For example, return the total, average, maximum, and minimum values.

```
db.collection.aggregate (pipeline, options)
```

## Pipeline Stages

| Stage | Description |
|---|---|
| $count | Returns a count of the number of documents at this stage of the aggregation pipeline. |
| $group | Groups input documents by a specified identifier expression and applies the accumulator expression(s), if specified, to each group. Consumes all input documents and outputs one document per each distinct group. The output documents only contain the identifier field and, if specified, accumulated fields. |
| $limit | Passes the first n documents unmodified to the pipeline where n is the specified limit. For each input document, outputs either one document (for the first n documents) or zero documents (after the first n documents). |
| $match | Filters the document stream to allow only matching documents to pass unmodified into the next pipeline stage. $match uses standard MongoDB queries. |
| $project | Reshapes each document in the stream, such as by adding new fields or removing existing fields. |
| $skip | Skips the first n documents where n is the specified skip number and passes the remaining documents unmodified to the pipeline |
| $sort | Reorders the document stream by a specified sort key. Only the order changes; the documents remain unmodified. For each input document, outputs one document. |
| $unwind | Deconstructs an array field from the input documents to output a document for each element. Each output document replaces the array with an element value. For each input document, outputs n documents where n is the number of array elements and can be zero for an empty array. |

## Aggregating Examples

```
demo>db.books.aggregate(
   {$match: {price : {"$gt": 500}}},
   {$group: { _id: "$cover", avgPrice: { $avg: "$price" } }})

[ { _id: 'h', avgPrice: 650 },
  { _id: 'p', avgPrice: 575 } ]
```

```
demo>db.books.aggregate(
        {$group: { _id: "$cover", avgPrice: { $avg: "$price" }}})

[{_id: 'p', avgPrice: 462.5 },
 {_id: 'h', avgPrice: 650 } ]
```

## CRUD Operations

```
db.collection.insertOne()
db.collection.insertMany()
db.collection.updateOne()
db.collection.updateMany()
db.collection.replaceOne()
db.collection.deleteOne()
db.collection.deleteMany()
db.collection.remove()
```

# insertOne()

- ❑ Inserts a single document into a collection.
- ❑ If the collection does not exist, then the insertOne() method creates the collection.
- ❑ If the document does not specify an _id field, then mongodb will add the _id field and assign a unique ObjectId() for the document before inserting.
- ❑ Return document containing:
  - ✓ A boolean acknowledged as true if the operation ran with write concern or false if write concern was disabled
  - ✓ A field insertedId with the _id value of the inserted document

```
db.collection.insertOne(
   <document>,
   {
      writeConcern: <document>
   }
)
```

# insertMany()

- ❑ Inserts multiple documents into a collection.
- ❑ Given an array of documents, insertMany() inserts each document in the array into the collection.
- ❑ If the collection does not exist, then the insertMany() method creates the collection.
- ❑ If the document does not specify an _id field, then mongodb will add the _id field and assign a unique ObjectId() for the document before inserting.
- ❑ Field ordered specifies whether the mongodb instance should perform an ordered or unordered insert. Defaults to true.
- ❑ Return document containing:
    - ✓ A boolean acknowledged as true if the operation ran with write concern or false if write concern was disabled
    - ✓ A field insertedIds with the _id value of the inserted document

```
db.collection.insertMany(
   [ <document 1> , <document 2>, ... ],
   {
      writeConcern: <document>,
      ordered: <boolean>
   }
)
```

## Inserting Documents

```
db.books.insertOne({
  "title": "Outliers",
  "author": "Malcolm Gladwell",
  "price": 550,
  "cover" : "p",
  "rating": {"goodreads": 4.2,"google": 88}
})
```

```
db.books.insertMany([{
  "title": "That will never work",
  "author": "Marc Randolph",
  "price": 600,
  "cover" : "p",
  "rating": {"goodreads": 4.2,"google": 89}
  },
  {
   …
  },
])
```

## updateOne()

- ❑ Updates a single document within the collection based on the filter.
- ❑ Cannot update _id field.
- ❑ The filter specifies the selection criteria for the update.
- ❑ Update specifies the modifications to apply.
- ❑ The parameter upsert, when set to true, creates a new document when no document matches filter. Default is false.
- ❑ It returns a document containing:
  - ✓ matchedCount containing the number of matched documents
  - ✓ modifiedCount containing the number of modified documents
  - ✓ upsertedId containing the _id for the upserted document

```
db.collection.updateOne(
   <filter>,
   <update>,
   {
     upsert: <boolean>
   }
)
```

## updateMany()

- Updates all documents that match the specified filter for a collection.
- The filter specifies the selection criteria for the update.
- Update specifies the modifications to apply.
- The parameter upsert, when set to true, creates a new document when no document matches filter. Default is false.
- It returns a document containing:
  - ✓ matchedCount containing the number of matched documents
  - ✓ modifiedCount containing the number of modified documents
  - ✓ upsertedId containing the _id for the upserted document

```
db.collection.updateMany(
   <filter>,
   <update>,
   {
     upsert: <boolean>
   }
)
```

# Updating One Document

```
demo> db.books.updateOne(
...     { title : "Outliers" },
...     {
...       $set: {price : 575, "rating.goodreads": 4.3}
...     }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

## Updating Many Documents

```
demo> db.books.updateMany(
      {"cover" : "p"},
      {$set : {"rating.goodreads": 4.5}}
)
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
```

## Deleting Documents

```
demo> db.books.deleteOne( { price : 550 } )
{ acknowledged: true, deletedCount: 1 }

demo> db.books.deleteMany({ cover : "h", price : {"$lt" : 400}})
{ acknowledged: true, deletedCount: 2 }
```

## Indexing

- MongoDB provides complete support for indexes on any field in a collection of documents.
- By default, all collections have an index on the _id field, and applications and users may add additional indexes to support important queries and operations.
- The value of the field in the index specification describes the kind of index for that field. For example, a value of 1 specifies an index that orders items in ascending order. A value of -1 specifies an index that orders items in descending order.
- Methods getIndexs(), createIndex() and dropIndex() are related to indexes.

## Indexing

```
db.books.createIndex({ "price" : 1 })
db.books.createIndex({ price : 1 }, {name : 'price_index'})
```

```
demo> db.books.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { price: 1 }, name: 'price_index' }
]
```

```
db.books.dropIndex('price_index')
```

## Drop collection

- ❑ Use drop() method with a collection to drop collection.

```
db.books.drop()
```