



SQL to MongoDB Mapping Chart

On this page

- Terminology and Concepts
- Executables
- Examples
- Further Reading

In addition to the charts that follow, you might want to consider the Frequently Asked Questions section for a selection of common questions about MongoDB.

Terminology and Concepts

The following table presents the various SQL terminology and concepts and the corresponding MongoDB terminology and concepts.

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	primary key
Specify any unique column or column combination as primary key.	In MongoDB, the primary key is automatically set to the <code>_id</code> field.
aggregation (e.g. group by)	aggregation pipeline
	See the SQL to Aggregation Mapping Chart.
SELECT INTO NEW_TABLE	<code>\$out</code>
	See the SQL to Aggregation Mapping Chart.
MERGE INTO TABLE	<code>\$merge</code> (Available starting in MongoDB 4.2)
	See the SQL to Aggregation Mapping Chart.
transactions	transactions

TIP:
For many scenarios, the denormalized data model (embedded documents and arrays) will continue to be optimal for your data and use cases instead of multi-document transactions. That is, for many scenarios, modeling your data appropriately will minimize the need for multi-document transactions.

Executables

The following table presents some database executables and the corresponding MongoDB executables. This table is *not* meant to be exhaustive.

 [Search Documentation](#)

Database Server	<code>mongod</code>	<code>mysqld</code>	<code>oracle</code>	<code>IDS</code>	<code>DB2 Server</code>
Database Client	<code>mongo</code>	<code>mysql</code>	<code>sqlplus</code>	<code>DB-Access</code>	<code>DB2 Client</code>

Examples

The following table presents the various SQL statements and the corresponding MongoDB statements. The examples in the table assume the following conditions:

- The SQL examples assume a table named `people`.
- The MongoDB examples assume a collection named `people` that contain documents of the following prototype:

```
{
  _id: ObjectId("509a8fb2f3f4948bd2f983a0"),
  user_id: "abc123",
  age: 55,
  status: 'A'
}
```

Create and Alter

The following table presents the various SQL statements related to table-level actions and the corresponding MongoDB statements.

SQL Schema Statements	MongoDB Schema Statements
-----------------------	---------------------------

```
CREATE TABLE people (  
  id MEDIUMINT NOT NULL  
    AUTO_INCREMENT,  
  user_id Varchar(30),  
  age Number,  
  status char(1),  
  PRIMARY KEY (id)  
)
```

implicitly created on most insert, save(), or insertMany() operation. The primary key `_id` is automatically added if `_id` field is not specified.

```
db.people.insertOne( {  
  user_id: "abc123",  
  age: 55,  
  status: "A"  
} )
```

However, you can also explicitly create a collection:

```
db.createCollection("people")
```

```
ALTER TABLE people  
ADD join_date DATETIME
```

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, `updateMany()` operations can add fields to existing documents using the `$set` operator.

```
db.people.updateMany(  
  { },  
  { $set: { join_date: new Date() } }  
)
```

```
ALTER TABLE people  
DROP COLUMN join_date
```

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, `updateMany()` operations can remove fields from documents using the `$unset` operator.

```
db.people.updateMany(  
  { },  
  { $unset: { "join_date": "" } }  
)
```

```
CREATE INDEX idx_user_id_asc  
ON people(user_id)
```

```
db.people.createIndex( { user_id: 1 } )
```

```
CREATE INDEX
    idx_user_id_asc_age_desc
ON people(user_id, age DESC)
```

```
db.people.createIndex( { user_id: 1, age: -1 } )
```

```
DROP TABLE people
```

```
db.people.drop()
```

For more information on the methods and operators used, see:

- `db.collection.insertOne()`
 - `db.collection.insertMany()`
 - `db.createCollection()`
- `db.collection.updateMany()`
 - `db.collection.createIndex()`
 - `db.collection.drop()`
- `$set`
 - `$unset`

SEE ALSO:

- Databases and Collections
- Documents
- Indexes
- Data Modeling Concepts.

Insert

The following table presents the various SQL statements related to inserting records into tables and the corresponding MongoDB statements.

SQL INSERT Statements

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

MongoDB insertOne() Statements

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

For more information, see `db.collection.insertOne()`.

- SEE ALSO
- Search Documentation
- [Insert](#)
 - `db.collection.insertMany()`
 - Databases and Collections
 - Documents

Select

The following table presents the various SQL statements related to reading records from tables and the corresponding MongoDB statements.

NOTE:

The `find()` method always includes the `_id` field in the returned documents unless specifically excluded through projection. Some of the SQL queries below may include an `_id` field to reflect this, even if the field is not included in the corresponding `find()` query.

SQL SELECT Statements	MongoDB find() Statements
<code>SELECT *</code> <code>FROM people</code>	<code>db.people.find()</code>
<code>SELECT id,</code> <code>user_id,</code> <code>status</code> <code>FROM people</code>	<code>db.people.find(</code> <code>{ },</code> <code>{ user_id: 1, status: 1 }</code> <code>)</code>
<code>SELECT user_id, status</code> <code>FROM people</code>	<code>db.people.find(</code> <code>{ },</code> <code>{ user_id: 1, status: 1, _id: 0 }</code> <code>)</code>
<code>SELECT *</code> <code>FROM people</code> <code>WHERE status = "A"</code>	<code>db.people.find(</code> <code>{ status: "A" }</code> <code>)</code>



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



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

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)`

EXPLAIN SELECT *
FROM people
WHERE status = "A"

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

For more information on the methods and operators used, see

- `db.collection.find()`
 - `db.collection.distinct()`
 - `db.collection.findOne()`
 - `limit()`
 - `skip()`
 - `explain()`
 - `sort()`
 - `count()`
- `$ne`
 - `$and`
 - `$or`
 - `$gt`
 - `$lt`
 - `$exists`
 - `$lte`
 - `$regex`

SEE ALSO:

- Query Documents
- Query and Projection Operators
- mongo Shell Methods

Update Records

The following table presents the various SQL statements related to deleting records from tables and the corresponding MongoDB statements.

SQL Update Statements	MongoDB updateMany() Statements
<pre>UPDATE people SET status = "C" WHERE age > 25</pre>	<pre>db.people.updateMany({ age: { \$gt: 25 } }, { \$set: { status: "C" } })</pre>
<pre>UPDATE people SET age = age + 3 WHERE status = "A"</pre>	<pre>db.people.updateMany({ status: "A" }, { \$inc: { age: 3 } })</pre>

For more information on the method and operators used in the examples, see:

- `db.collection.updateMany()`
- `$gt`
- `$set`
- `$inc`

SEE ALSO:

- Update Documents
- Update Operators
- `db.collection.updateOne()`
- `db.collection.replaceOne()`

Delete Records

The following table presents the various SQL statements related to deleting records from tables and the corresponding MongoDB statements.

SQL Delete Statements	MongoDB deleteMany() Statements
-----------------------	---------------------------------

SQL Documentation

DELETE FROM people
WHERE status = "D"

db.people.deleteMany({ status: "D" })

DELETE FROM people

db.people.deleteMany({})

For more information, see `db.collection.deleteMany()`.

SEE ALSO:

- Delete Documents
- `db.collection.deleteOne()`

Further Reading

If you are considering migrating your SQL application to MongoDB, download the MongoDB Application Modernization Guide [🔗](#).

The download includes the following resources:

- Presentation on the methodology of data modeling with MongoDB
- White paper covering best practices and considerations for migrating to MongoDB from an RDBMS data model
- Reference MongoDB schema with its RDBMS equivalent
- Application Modernization scorecard