SQL to MongoDB Mapping Chart

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	`$lookup`, embedded documents
primary key Specify any unique column or column combination as primary key.	primary key In MongoDB, the primary key is automatically set to the _id field.
aggregation (e.g. group by)	aggregation pipeline See the SQL to Aggregation Mapping Chart.
SELECT INTO NEW_TABLE	`$out` See the SQL to Aggregation Mapping Chart.
MERGE INTO TABLE	`$merge` (Available starting in MongoDB 4.2) See the SQL to Aggregation Mapping Chart.
UNION ALL	`$unionWith`

SQL Terms/Concepts	MongoDB Terms/Concepts
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.

	MongoDB	MySQL	Oracle	Informix	DB2
Database Server	`mongod`	`mysqld`	`oracle`	`IDS`	`DB2 Server`
Database Client	`mongosh`	`mysql`	`sqlplus`	`DB-Access`	`DB2 Client`

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 first `insertOne()` 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() } })

SQL Schema Statements	MongoDB Schema Statements
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`

TIP

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	MongoDB insertOne() Statements
INSERT INTO people(user_id, age, status) VALUES ("bcd001", 45, "A")	db.people.insertOne({ user_id: "bcd001", age: 45, status: "A" })

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

TIP

See also:

- Insert Documents
- `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
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 } })

SQL SELECT Statements	MongoDB find() Statements
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 WHERE user_id like "bc%"	db.people.find({ user_id: /^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()

SQL SELECT Statements	MongoDB find() Statements
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)
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`

TIP

See also:

- Query Documents
- Query and Projection Operators
- `mongosh` Methods

Update Records

The following table presents the various SQL statements related to updating existing records in tables and the corresponding MongoDB statements.

SQL Update Statements	MongoDB updateMany() Statements
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 } })

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

- `db.collection.updateMany()`

- `$gt`
- `$set`
- `$inc`

TIP

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
DELETE FROM people WHERE status = "D"	db.people.deleteMany({ status: "D" })
DELETE FROM people	db.people.deleteMany({})

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

TIP

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