CRUD COMMANDS MONGO

ELI LEIBA

COMMAND LIST

CREATE A COLLECTION •

- SOME FIND EXAMPLES •
- INSERT DOCUMENTS IN COLLECTION •
- ADD A COLUMN •
- DROP A COLUMN •
- DROP A COLLECTION •
- UPDATE DOCUMENT IN A COLLECTION •
- REMOVE DOCUMENTS FROM COLLECTION •

CRUD –CREATE a COLLECTION

SQL Schema Statements

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

MongoDB Schema Statements

Implicitly created on first `insert()` operation. The primary key `_id` is automatically added if `_id` field is not specified.

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

However, you can also explicitly create a collection:

```
db.createCollection("users")
```

CRUD - FIND

Query Interface

For query operations, MongoDB provides a `db.collection.find()` method. The method accepts both the query criteria and projections and returns a cursor to the matching documents. You can optionally modify the query to impose limits, skips, and sort orders.

The following diagram highlights the components of a MongoDB query operation:

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

The next diagram shows the same query in SQL:

```
SELECT _id, name, address   <-- projection
FROM   users                <-- table
WHERE  age > 18             <-- select criteria
LIMIT  5                    <-- cursor modifier
```

EXAMPLE:

```
db.users.find( { age: { $gt: 18 } }, { name: 1, address: 1 } ).limit(5)
```

CRUD – More find examples

SQL SELECT Statements	MongoDB find() Statements
`SELECT *` `FROM users`	`db.users.find()`

`SELECT id,`
`  user_id,`
`  status`
`FROM users`
	`db.users.find(`
	`  { },`
	`  { user_id: 1, status: 1 }`
	`)`

`SELECT user_id, status`
`FROM users`
	`db.users.find(`
	`  { },`
	`  { user_id: 1, status: 1, _id: 0 }`
	`)`

`SELECT *`
`FROM users`
`WHERE status = "A"`
	`db.users.find(`
	`  { status: "A" }`
	`)`

`SELECT user_id, status`
`FROM users`
`WHERE status = "A"`
	`db.users.find(`
	`  { status: "A" },`
	`  { user_id: 1, status: 1, _id: 0 }`
	`)`

`SELECT *`
`FROM users`
`WHERE status != "A"`
	`db.users.find(`
	`  { status: { $ne: "A" } }`
	`)`

CRUD – More find examples

SQL Schema Statements	MongoDB Schema Statements
`SELECT * FROM users WHERE status = "A" AND age = 50`	`db.users.find( { status: "A", age: 50 } )`
`SELECT * FROM users WHERE status = "A" OR age = 50`	`db.users.find( { $or: [ { status: "A" } , { age: 50 } ] } )`
`SELECT * FROM users WHERE age > 25`	`db.users.find( { age: { $gt: 25 } } )`
`SELECT * FROM users WHERE age < 25`	`db.users.find( { age: { $lt: 25 } } )`
`SELECT * FROM users WHERE age > 25 AND age <= 50`	`db.users.find( { age: { $gt: 25, $lte: 50 } } )`
`SELECT * FROM users WHERE user_id like "%bc%"`	`db.users.find( { user_id: /bc/ } )`
`SELECT * FROM users WHERE user_id like "bc%"`	`db.users.find( { user_id: /^bc/ } )`

CRUD – More find examples

SQL Schema Statements	MongoDB Schema Statements

`SELECT *`
`FROM users`
`WHERE status = "A"`
`ORDER BY user_id DESC`
	`db.users.find( { status: "A" } ).sort( { user_id: -1 } )`

`SELECT COUNT(*)`
`FROM users`
	`db.users.count()`
	or
	`db.users.find().count()`

`SELECT COUNT(user_id)`
`FROM users`
	`db.users.count( { user_id: { $exists: true } } )`
	or
	`db.users.find( { user_id: { $exists: true } } ).count()`

`SELECT COUNT(*)`
`FROM users`
`WHERE age > 30`
	`db.users.count( { age: { $gt: 30 } } )`
	or
	`db.users.find( { age: { $gt: 30 } } ).count()`

`SELECT DISTINCT(status)` `FROM users`	`db.users.distinct( "status" )`

`SELECT *`
`FROM users`
`LIMIT 1`
	`db.users.findOne()`
	or
	`db.users.find().limit(1)`

CRUD – ADD COLUMN

SQL Schema Statements

```
ALTER TABLE users
ADD join_date DATETIME
```

MongoDB Schema Statements

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, `update()` operations can add fields to existing documents using the `$set` operator.

```
db.users.update(
    { },
    { $set: { join_date: new Date() } },
    { multi: true }
)
```

CRUD DROP COLUMN

SQL Schema Statements	MongoDB Schema Statements

`ALTER TABLE users`
`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, `update()` operations can remove fields from documents using the `$unset` operator.
	
	`db.users.update(`
	`{ },`
	`{ $unset: { join_date: "" } },`
	`{ multi: true }`
	`)`

CRUD – DROP COLLECTION

SQL Schema Statements	MongoDB Schema Statements
`DROP TABLE users`	`db.users.drop()`

CRUD - INSERT

Insert

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

SQL INSERT Statements	MongoDB insert() Statements
`INSERT INTO users(user_id, age, status) VALUES ("bcd001", 45, "A")`	`db.users.insert( { user_id: "bcd001", age: 45, status: "A" } )`

CRUD - UPDATE

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 update() Statements

`UPDATE users`
`SET status = "C"`
`WHERE age > 25`
	`db.users.update(`
	`{ age: { $gt: 25 } },`
	`{ $set: { status: "C" } },`
	`{ multi: true }`
	`)`

`UPDATE users`
`SET age = age + 3`
`WHERE status = "A"`
	`db.users.update(`
	`{ status: "A" } ,`
	`{ $inc: { age: 3 } },`
	`{ multi: true }`
	`)`

CRUD - DELETE

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 remove() Statements
`DELETE FROM users` `WHERE status = "D"`	`db.users.remove( { status: "D" } )`
`DELETE FROM users`	`db.users.remove({})`