MongoDB.

SQL to MongoDB Cheat Sheet

Key Concepts

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

SQL Concepts	**MongoDB** Concepts
database	database
table	collection
row	document or BSON document
column	field
index	index
table joins	$lookup, embedded documents
primary key	primary key (_id)
aggregation (e.g. group by)	aggregation pipeline
transactions	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	PostGreSQL	SQL Server
Database server	mongod	mysql	oracle	PostGreSQL	SQL Server
Database client	mongosh	mysql	sqlplus	PGAdmin	SQL Server Management Studio

Create and Alter

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

SQL Schema	**MongoDB** Schema

`CREATE TABLE people (`
`  id MEDIUMINT`
`    NOT NULL`
`    AUTO_INCREMENT,`
`  user_id Varchar(30),`
`  age int,`
`  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"`
	`} )`
	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()`

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" }`
	`)`

Select

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

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 *
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 }] }
	)

```sql
SELECT *
FROM people
WHERE age > 25
```

```javascript
db.people.find(
    { age: { $gt: 25 } }
)
```

```sql
SELECT *
FROM people
WHERE age < 25
```

```javascript
db.people.find(
    { age: { $lt: 25 } }
)
```

```sql
SELECT *
FROM people
WHERE age > 25
  AND age <= 50
```

```javascript
db.people.find(
    { age: { $gt: 25, $lte: 50 } }
)
```

```sql
SELECT *
FROM people
WHERE user_id LIKE "%bc%"
```

```javascript
db.people.find( { user_id: /bc/ } )
```

```sql
SELECT *
FROM people
WHERE user_id LIKE "jim%"
```

```javascript
db.people.find( { user_id: /^jim/ } )
```

```sql
SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id ASC
```

```javascript
db.people.find( { status: "A" } ).sort( {user_id: 1 } )
```

```sql
SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id DESC
```

```javascript
db.people.find( { status: "A" } ).sort( {user_id: -1 } )
```

```sql
SELECT COUNT(*)
FROM people
```

```javascript
db.people.countDocuments()
```

```sql
SELECT COUNT(user_id)
FROM people
```

```javascript
db.people.count( { user_id: { $exists: true } } )
```

SELECT COUNT(*)
FROM people
WHERE age > 30
	db.people.count({ age: { $gt: 30 } })

SELECT DISTINCT(status)
FROM people
	db.people.aggregate([{$group : {_id :"$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()

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 } }
	)

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( { } )`

SQL to Aggregation

The aggregation pipeline allows MongoDB to provide native aggregation capabilities that correspond to many common data aggregation operations in SQL.

The following table provides an overview of common SQL aggregation terms, functions, and concepts and the corresponding MongoDB aggregation operators

SQL Terms, Functions and Concepts	**MongoDB** Aggregation Operators
WHERE	`$match`
GROUP BY	`$group`
HAVING	`$match`
SELECT	`$project`
ORDER BY	`$sort`
LIMIT	`$limit`
SUM()	`$sum`
COUNT()	`$sum, $sortByCount`
join	`$lookup`
SELECT INTO NEW_TABLE	`$out`
MERGE INTO TABLE	`$merge`
UNION ALL	`$unionWith`

Aggregation Examples

The following table presents a quick reference of SQL aggregation statements and the corresponding MongoDB statements. The examples in the table assume the following conditions:

- The SQL examples assume two tables, **`orders`** and **`order_lineitem`** that join by the **`order_lineitem.order_id`** and the **`orders.id`** columns.

- The MongoDB examples assume one collection orders that contain documents of the following prototype:

```
{
  cust_id: "abc123",
  ord_date: ISODate("2012-11-02T17:04:11.102Z"),
  status: 'A',
  price: 50,
  items: [ { sku: "xxx", qty: 25, price: 1 },
           { sku: "yyy", qty: 25, price: 1 } ]
}
```

SQL Example	MongoDB Example	Description

`SELECT COUNT(*) AS count`
`FROM orders`
	`db.orders.aggregate( [`
	`  {`
	`   $group: (`
	`      _id: null,`
	`      count: { $sum: 1 }`
	`   }`
	` }`
	`] )`
		Count all records from orders

`SELECT SUM(price) AS`
`total`
`FROM orders`
	`db.orders.aggregate( [`
	`  {`
	`   $group: (`
	`      _id: null,`
	`      total: { $sum: "$price" }`
	`   }`
	` }`
	`] )`
		Sum the price field from orders.

`SELECT`
`   cust_id,`
`   SUM(price) AS total`
`FROM orders`
`GROUP BY cust_id`
	`db.orders.aggregate( [`
	`  {`
	`    $group: (`
	`       _id: "$cust_id",`
	`       total: { $sum: "$price" }`
	`   }`
	` }`
	`] )`
		For each unique cust_id, sum the price field.

`SELECT`
`   cust_id,`
`   SUM(price) AS total`
`FROM orders`
`GROUP BY cust_id`
`ORDER BY total`
	`db.orders.aggregate( [`
	`   {`
	`     $group: (`
	`        _id: "$cust_id",`
	`        total: { $sum: "$price"}`
	`     }`
	`   },`
	`   { $sort: { total: 1 } }`
	`] )`
		For each unique cust_id, sum the price field, results sorted by sum.

Note: The SQL commands used in this document were run in MySQL.
Note: Some implementations of SQL require delimiters. Delimiters are omitted in this document.

References

SQL to MongoDB Aggregation Chart

SQL to MongoDB Mapping Chart

Resources

MongoDB for SQL Professionals - a free online course on MongoDB University

MongoDB for Students - Students are eligible for free resources including MongoDB certification and Atlas credits through the GitHub Student Developer Pack