SQL vs MongoDB Commands

Database and Collection Management: This category covers operations related to creating or removing databases and collections (tables in SQL). MongoDB automatically creates a database when you first use it, and collections can be created implicitly or explicitly.

CRUD Operations: CRUD stands for Create, Read, Update, and Delete. These are the fundamental operations for working with data. MongoDB's `find()` command is the equivalent of SQL's `SELECT`, and `updateOne()`/`deleteOne()` are for updating and deleting specific documents.

Schema and Field Management: SQL typically requires explicit schema modifications, like adding columns to a table. MongoDB, on the other hand, allows for dynamic schema, where fields can be added directly to documents on the fly, without altering the collection's structure.

Data Retrieval and Aggregation: SQL provides powerful querying capabilities through `SELECT`, `GROUP BY`, and aggregate functions. MongoDB's `aggregate()` method is used for more complex queries and data transformations, including grouping and aggregating data.

Indexing: Both SQL and MongoDB use indexes to optimize query performance. MongoDB supports compound indexes and allows for indexing individual fields.

Join Operations: SQL uses `JOIN` to combine data from multiple tables. MongoDB handles joins through the `$lookup` operator in its aggregation framework, which allows you to combine data from different collections.

Category	Operation	SQL Command	MongoDB Command	Example SQL	Example MongoDB
Database and Collection Management	**Create Database**	CREATE DATABASE	MongoDB creates database implicitly when first used	CREATE DATABASE mydb;	use mydb;
	Create Table (Collection)	CREATE TABLE	db.createCollection()	CREATE TABLE users (id INT, name VARCHAR(100));	db.createCollection('users')
	Drop Table (Collection)	DROP TABLE	db.collection.drop()	DROP TABLE users;	db.users.drop()
CRUD Operations	**Insert Data**	INSERT INTO	db.collection.insertOne() / insertMany()	INSERT INTO users (id, name) VALUES (1, 'Alice');	db.users.insertOne({id: 1, name: 'Alice'})
	Select Data	SELECT	db.collection.find()	SELECT * FROM users WHERE id = 1;	db.users.find({id: 1})

Category	Operation	SQL Command	MongoDB Command	Example SQL	Example MongoDB
	Select All Data	SELECT * FROM	db.collection.find()	SELECT * FROM users;	db.users.find()
	Update Data	UPDATE	db.collection.updateOne() / updateMany()	UPDATE users SET name = 'Bob' WHERE id = 1;	db.users.updateOne({id: 1}, {$set: {name: 'Bob'}})
	Delete Data	DELETE	db.collection.deleteOne() / deleteMany()	DELETE FROM users WHERE id = 1;	db.users.deleteOne({id: 1})
Schema and Field Management	**Add Column (Field)**	ALTER TABLE ADD COLUMN	No direct equivalent; add field on the fly during insert or update	ALTER TABLE users ADD COLUMN age INT;	db.users.updateMany({}, {$set: {age: 30}})
	Dynamically Add Fields to Documents	No direct equivalent	No equivalent command, fields are added directly when inserting or updating documents	N/A	db.users.updateOne({name: "Alice"}, {$set: {email: "alice@example.com"}})
Data Retrieval and Aggregation	**Select Specific Fields**	SELECT column1, column2	db.collection.find({}, {field1: 1, field2: 1})	SELECT name, age FROM users;	db.users.find({}, {name: 1, age: 1})
	Count Records	SELECT COUNT(*)	db.collection.countDocuments()	SELECT COUNT(*) FROM users;	db.users.countDocuments()
	Distinct Values	SELECT DISTINCT	db.collection.distinct()	SELECT DISTINCT name FROM users;	db.users.distinct('name')
	Group Data	GROUP BY	db.collection.aggregate()	SELECT COUNT(*), city FROM users GROUP BY city;	db.users.aggregate([{$group: {_id: "$city", count: {$sum: 1}}}])
Indexing	**Create Index**	CREATE INDEX	db.collection.createIndex()	CREATE INDEX idx_name ON users (name);	db.users.createIndex({name: 1})
Join Operations	**Join Tables (Collections)**	JOIN	$lookup	SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id;	db.orders.aggregate([{$lookup: {from: 'customers', localField: 'customer_id', foreignField: '_id', as: 'customer'}}])