MongoDB vs SQL – Complete Query Reference (Basic → Advanced)

This document is a **single, complete reference** that maps **SQL queries to MongoDB queries**, including **simple, complex, joins, subqueries, aggregations, pagination, window-like logic, and edge cases**.

Think of this as:

> 📘 *"SQL → MongoDB translation bible for backend & full-stack developers"*

1️⃣ TERMINOLOGY MAPPING

SQL	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field
Primary Key	`_id`
Foreign Key	Reference field

2️⃣ SELECT QUERIES

2.1 SELECT ALL

SQL

```sql
SELECT * FROM users;
```

MongoDB

```
db.users.find({})
```

2.2 SELECT SPECIFIC COLUMNS (PROJECTION)

SQL

```
SELECT name, age FROM users;
```

MongoDB

```
db.users.find({}, { name: 1, age: 1, _id: 0 })
```

3️⃣WHERE CLAUSE (FILTERING)

3.1 BASIC CONDITIONS

SQL	MongoDB
`=`	`{ field: value }`
`!=`	`{ $ne }`
`>`	`{ $gt }`
`<`	`{ $lt }`
`>=`	`{ $gte }`
`<=`	`{ $lte }`

Example

SQL

```
SELECT * FROM users WHERE age > 25;
```

MongoDB

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

3.2 AND / OR / NOT

SQL

```
SELECT * FROM users WHERE age > 25 AND salary > 50000;
```

MongoDB

```
db.users.find({
  $and: [
    { age: { $gt: 25 } },
    { salary: { $gt: 50000 } }
  ]
})
```

OR

```
db.users.find({
  $or: [{ age: 25 }, { age: 30 }]
})
```

3.3 IN / NOT IN

SQL

```sql
SELECT * FROM users WHERE age IN (25, 30);
```

MongoDB

```
db.users.find({ age: { $in: [25, 30] } })
```

3.4 LIKE / PATTERN MATCH

SQL

```sql
SELECT * FROM users WHERE name LIKE 'Ro%';
```

MongoDB

```
db.users.find({ name: /^Ro/ })
```

4️⃣ORDER BY, LIMIT, OFFSET

4.1 ORDER BY

SQL

```
SELECT * FROM users ORDER BY salary DESC;
```

MongoDB

```
db.users.find().sort({ salary: -1 })
```

4.2 LIMIT & OFFSET (PAGINATION)

SQL

```
SELECT * FROM users LIMIT 10 OFFSET 20;
```

MongoDB

```
db.users.find().skip(20).limit(10)
```

5️⃣UPDATE QUERIES

5.1 UPDATE ONE

SQL

```
UPDATE users SET salary = 60000 WHERE id = 1;
```

MongoDB

```
db.users.updateOne(
  { _id: 1 },
  { $set: { salary: 60000 } }
)
```

5.2 UPDATE MANY

```
db.users.updateMany(
  { age: { $gt: 30 } },
  { $inc: { salary: 5000 } }
)
```

6️⃣DELETE QUERIES

SQL

```
DELETE FROM users WHERE age < 20;
```

MongoDB

```
db.users.deleteMany({ age: { $lt: 20 } })
```

7️⃣AGGREGATION (GROUP BY, HAVING, FUNCTIONS)

7.1 GROUP BY

SQL

```
SELECT age, COUNT(*) FROM users GROUP BY age;
```

MongoDB

```
db.users.aggregate([
  { $group: { _id: "$age", count: { $sum: 1 } } }
])
```

7.2 AGGREGATE FUNCTIONS

SQL	MongoDB
COUNT	`$sum: 1`
SUM	`$sum`
AVG	`$avg`
MIN	`$min`
MAX	`$max`

```
db.users.aggregate([
  {
    $group: {
      _id: null,
      totalSalary: { $sum: "$salary" },
      avgSalary: { $avg: "$salary" },
      maxSalary: { $max: "$salary" }
    }
  }
])
```

7.3 HAVING

SQL

```
SELECT age, COUNT(*) FROM users GROUP BY age HAVING COUNT(*) > 2;
```

MongoDB

```
db.users.aggregate([
  { $group: { _id: "$age", count: { $sum: 1 } } },
  { $match: { count: { $gt: 2 } } }
])
```

8️⃣JOINS

8.1 INNER JOIN

SQL

```
SELECT * FROM orders o
JOIN users u ON o.userId = u.id;
```

MongoDB

```
db.orders.aggregate([
  {
    $lookup: {
      from: "users",
      localField: "userId",
      foreignField: "_id",
      as: "user"
    }
  },
  { $unwind: "$user" }
])
```

8.2 LEFT JOIN

```
db.orders.aggregate([
  {
    $lookup: {
      from: "users",
      localField: "userId",
      foreignField: "_id",
      as: "user"
    }
  }
])
```

9️⃣SUBQUERIES

SQL

```
SELECT * FROM users WHERE id IN (
  SELECT userId FROM orders WHERE total > 5000
);
```

MongoDB

```
db.orders.aggregate([
  { $match: { total: { $gt: 5000 } } },
  { $group: { _id: "$userId" } },
  {
    $lookup: {
      from: "users",
      localField: "_id",
      foreignField: "_id",
      as: "user"
    }
  }
])
```

🔟INDEXES

SQL

```sql
CREATE INDEX idx_email ON users(email);
```

MongoDB

```
db.users.createIndex({ email: 1 })
```

1️⃣1️⃣ADVANCED PAGINATION (CURSOR BASED)

```
db.users.find({ _id: { $gt: lastSeenId } })
        .sort({ _id: 1 })
        .limit(10)
```

Used for feeds & infinite scroll.

1️⃣2️⃣FINAL MENTAL MODEL

SQL Concept	MongoDB
SELECT	find / aggregate
WHERE	filter / $match
GROUP BY	$group
HAVING	$match after $group
JOIN	$lookup
ORDER BY	sort
LIMIT OFFSET	limit + skip
SUBQUERY	aggregation pipeline

1️⃣3️⃣WINDOW FUNCTIONS (RANK, DENSE_RANK, RUNNING TOTAL)

ROW_NUMBER / RANK Equivalent

SQL

```sql
SELECT id, salary,
       RANK() OVER (ORDER BY salary DESC) AS rank
FROM users;
```

MongoDB

```
db.users.aggregate([
  { $setWindowFields: {
      sortBy: { salary: -1 },
      output: {
        rank: { $rank: {} }
      }
  }}
])
```

RUNNING TOTAL

SQL

```sql
SELECT date, SUM(amount) OVER (ORDER BY date) AS running_total
FROM payments;
```

MongoDB

```
db.payments.aggregate([
  { $setWindowFields: {
      sortBy: { date: 1 },
      output: {
        runningTotal: {
          $sum: "$amount",
          window: { documents: ["unbounded", "current"] }
        }
      }
  }}
])
```

1️⃣4️⃣COMPLEX JOINS (MULTI-COLLECTION)

SQL

```
SELECT u.name, o.total, p.status
FROM users u
JOIN orders o ON u.id = o.userId
JOIN payments p ON o.id = p.orderId;
```

MongoDB

```
db.users.aggregate([
  { $lookup: {
      from: "orders",
      localField: "_id",
      foreignField: "userId",
      as: "orders"
  }},
  { $unwind: "$orders" },
  { $lookup: {
      from: "payments",
      localField: "orders._id",
      foreignField: "orderId",
      as: "payment"
  }},
  { $unwind: "$payment" }
])
```

1️⃣5️⃣ARRAY QUERIES (ADVANCED)

ANY / ALL

SQL

```
SELECT * FROM users WHERE 5 = ANY(scores);
```

MongoDB

```
db.users.find({ scores: 5 })
```

ARRAY FILTER + UPDATE

```
db.users.updateMany(
  { "skills.level": "junior" },
  { $set: { "skills.$[elem].level": "mid" } },
  { arrayFilters: [{ "elem.level": "junior" }] }
)
```

1️⃣6️⃣ TRANSACTIONS (ACID)

SQL

```sql
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;
```

MongoDB

```js
session.startTransaction();
try {
  db.accounts.updateOne({ _id: 1 }, { $inc: { balance: -100 } }, { session })
  db.accounts.updateOne({ _id: 2 }, { $inc: { balance: 100 } }, { session })
  session.commitTransaction()
} catch (e) {
  session.abortTransaction()
}
```

1️⃣7️⃣ CTE (WITH CLAUSE) EQUIVALENT

SQL

```sql
WITH high_orders AS (
  SELECT userId FROM orders WHERE total > 5000
)
SELECT * FROM users WHERE id IN (SELECT userId FROM high_orders);
```

MongoDB

```js
db.orders.aggregate([
  { $match: { total: { $gt: 5000 } } },
  { $group: { _id: "$userId" } },
  { $lookup: {
      from: "users",
      localField: "_id",
      foreignField: "_id",
      as: "user"
  }}
])
```

1️⃣8️⃣QUERY EXPLAIN & PERFORMANCE

SQL

```
EXPLAIN ANALYZE SELECT * FROM users WHERE email = 'a@b.com';
```

MongoDB

```
db.users.find({ email: "a@b.com" }).explain("executionStats")
```

1️⃣9️⃣FINAL SUMMARY TABLE (ADVANCED)

SQL Advanced Feature	MongoDB Equivalent
Window Functions	$setWindowFields
CTE	Aggregation pipeline
Multi-Join	Multiple $lookup
ANY / ALL	Array match
Transactions	session + transaction
Explain	explain()

📌**This is now a COMPLETE BASIC → ADVANCED → EXPERT MongoDB vs SQL reference in one place.**

You can directly use this as: - Personal documentation - Interview prep - Backend architecture guide - SQL → MongoDB migration reference

If you want next: - Same doc in **PDF / Markdown** - **Mongoose equivalents** line-by-line - **Real production schemas & indexes** - **Interview questions mapped to queries**

Just tell me.