MongoDB with Mongoose (ORM) – Complete Query Documentation (Basic → Advanced)

This document is a **full ORM-level reference** showing how **everything you do in SQL or raw MongoDB** is implemented using **Mongoose in Node.js**.

Think of this as:

> 📘 *"SQL → MongoDB → Mongoose ORM translation guide for production Node.js apps"*

1️⃣ BASIC SETUP

```
import mongoose from "mongoose";

await mongoose.connect("mongodb://localhost:27017/app");
```

2️⃣ SCHEMA & MODEL (TABLE → COLLECTION)

SQL

```
CREATE TABLE users (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  age INT,
  salary INT
);
```

Mongoose

```
const userSchema = new mongoose.Schema({
  name: String,
  age: Number,
  salary: Number
});

const User = mongoose.model("User", userSchema);
```

3️⃣ SELECT / FIND QUERIES

SELECT *

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

```
User.find();
```

SELECT specific fields

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

```
User.find({}, { name: 1, age: 1, _id: 0 });
```

4️⃣ WHERE CLAUSE (FILTERING)

Comparisons

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

```
User.find({
  age: { $gt: 25 },
  salary: { $gte: 50000 }
});
```

IN / NOT IN

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

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

5️⃣ SORT, LIMIT, PAGINATION

```sql
SELECT * FROM users ORDER BY salary DESC LIMIT 10 OFFSET 20;
```

```
User.find()
  .sort({ salary: -1 })
  .skip(20)
  .limit(10);
```

Cursor-based pagination

```
User.find({ _id: { $gt: lastId } })
  .sort({ _id: 1 })
  .limit(10);
```

6️⃣ INSERT

```
INSERT INTO users (name, age, salary) VALUES ('A', 25, 40000);
```

```
await User.create({ name: "A", age: 25, salary: 40000 });
```

7️⃣ UPDATE

UPDATE ONE

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

```
User.updateOne({ _id: id }, { $set: { salary: 60000 } });
```

UPDATE MANY

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

8️⃣ DELETE

```
DELETE FROM users WHERE age < 18;
```

```
User.deleteMany({ age: { $lt: 18 } });
```

9️⃣ AGGREGATION (GROUP BY, HAVING)

GROUP BY

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

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

HAVING

```
HAVING COUNT(*) > 2;
```

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

🔟 JOINS (POPULATE vs LOOKUP)

SQL JOIN

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

Mongoose Populate (ORM-level JOIN)

```
Order.find().populate("userId");
```

Aggregation JOIN

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

```
  { $unwind: "$user" }
]);
```

1️⃣1️⃣ SUBQUERIES (IN / EXISTS)

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

```
User.aggregate([
  {
    $lookup: {
      from: "orders",
      let: { uid: "$_id" },
      pipeline: [
        { $match: { $expr: { $and: [
          { $eq: ["$userId", "$$uid"] },
          { $gt: ["$total", 5000] }
        ] } } }
      ],
      as: "orders"
    }
  },
  { $match: { orders: { $ne: [] } } }
]);
```

1️⃣2️⃣ WINDOW FUNCTIONS (MongoDB 5+)

```
SELECT userId, RANK() OVER (ORDER BY salary DESC) FROM users;
```

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

1️⃣3️⃣ TRANSACTIONS (ACID)

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

```
const session = await mongoose.startSession();
session.startTransaction();

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

1️⃣4️⃣ INDEXES

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

```
userSchema.index({ email: 1 });
```

1️⃣5️⃣ QUERY PERFORMANCE

```
User.find({ age: { $gt: 30 } }).explain("executionStats");
```

1️⃣6️⃣ FINAL ORM MENTAL MODEL

SQL	MongoDB	Mongoose
Table	Collection	Model
Row	Document	Document
JOIN	$lookup	populate

SQL	MongoDB	Mongoose
WHERE	filter	find
GROUP BY	aggregate	aggregate

📌**This document covers 95% of real-world backend MongoDB usage using Mongoose ORM.**