SQL vs MongoDB — Kab Kaun Sa?

The decider lesson: the same questions in both languages, connected data both ways, and the three questions that pick the right database.

Introduction

You now speak both languages of data: SQL's tables (the DB series) and MongoDB's documents (this series). So the real question of this whole phase arrives: **when a new app lands on your desk, which one do you reach for?** This lesson runs the same questions through both databases, side by side — the same five students, the same fees — and ends with the three decision questions professionals actually ask. There is no "winner" here; there is only *fit*.

(Roman Urdu: Sawal ye nahi ke kaun behtar hai — sawal ye hai ke is app ke data ki shape kya hai.)

Key points

- **Shape decides first:** data with a fixed, known shape that rarely changes (students, fees, orders) sits naturally in SQL tables. Data whose shape varies per record or evolves weekly (products with different features, app content, event logs) sits naturally in documents.
- **Connections decide second:** if the app's daily questions join many things (student + fees + class + teacher), SQL's JOIN is home ground. If each screen reads one self-contained record (a post with its author snapshot, a product with its sizes), MongoDB's embedded document answers in one fetch.
- **Rules vs freedom is a real trade:** SQL enforces one strict shape for everyone — safety by design, change by ceremony (ALTER TABLE). MongoDB lets each record differ — change is free, and the discipline becomes *your* job (which is why Mongoose schemas exist, MG.5).
- **Both scale; differently:** SQL traditionally grows *up* (a stronger server) and by careful design; MongoDB was built to grow *out* (sharding across many servers). For the apps in this course, both are honestly far more than enough — fit matters before scale.
- **Transactions:** SQL's all-or-nothing transactions (DB.10) are its birthright. MongoDB supports multi-document transactions too — but its design philosophy avoids needing them: data kept together in one document changes together, atomically, by nature.
- **The honest summary:** most real systems you will meet use SQL for the money-and-records core. MongoDB shines at the flexible edges — and many apps run both. Choosing is a per-app decision, not a religion.

Code examples

Example 1 — The same question, both languages

```
// SQL (from DB.1)
SELECT name, marks FROM students WHERE marks >= 90;

// MongoDB (from MG.4)
db.students.find({ marks: { $gte: 90 } }, { name: 1, marks: 1, _id: 0 })
```

What it does: Identical question — *the 90 club* — asked of the same five students in both worlds. SQL names columns to return; MongoDB's projection does the same job. Compare the two answers below: the *facts* are identical; only the packaging differs.

What you will see:

```
+-----------+-------+
| name      | marks |
+-----------+-------+
| Sara Khan |    92 |
| Hira Shah |    95 |
+-----------+-------+
2 rows in set

[ { name: 'Sara Khan', marks: 92 }, { name: 'Hira Shah', marks: 95 } ]
```

A grid of rows versus a list of small documents. Neither answer is "more correct" — this is the whole comparison in miniature: **SQL answers in tables; MongoDB answers in records that resemble the objects your code already uses.** Pick the packaging your app thinks in.

Example 2 — Connected data: JOIN vs the document way

```
// SQL (from DB.5) — reconnect two tables at question time
SELECT students.name, fees.amount, fees.status
FROM students
LEFT JOIN fees ON students.id = fees.student_id;

// MongoDB — the fee lives with its own document; $lookup visits the neighbour
db.fees.find()
```

What it does: September's fees (from DB.5): Ali, Sara and Bilal paid; Usman's is pending; Hira has **no fee record at all**. SQL reconnects the two tables at question time — the LEFT JOIN keeps Hira visible with NULL. MongoDB keeps fees as their own small documents carrying a `studentId` reference, joining only when a question truly crosses the border (the full `$lookup` is MG.8's lesson).

What you will see:

```
+-------------+--------+---------+
| name        | amount | status  |
+-------------+--------+---------+
| Ali Raza    |   5000 | Paid    |
| Sara Khan   |   5000 | Paid    |
| Bilal Ahmed |   4500 | Paid    |
| Hira Shah   |   NULL | NULL    |
| Usman Tariq |   5000 | Pending |
+-------------+--------+---------+
5 rows in set

[
  { _id: ObjectId('650000000000000000000201'), studentId:
ObjectId('650000000000000000000001'), month: 'September', amount: 5000, status: 'Paid' },
  { _id: ObjectId('650000000000000000000202'), studentId:
ObjectId('650000000000000000000002'), month: 'September', amount: 5000, status: 'Paid' },
  { _id: ObjectId('650000000000000000000203'), studentId:
ObjectId('650000000000000000000003'), month: 'September', amount: 4500, status: 'Paid' },
  { _id: ObjectId('650000000000000000000204'), studentId:
ObjectId('650000000000000000000005'), month: 'September', amount: 5000, status: 'Pending' }
]
```

Same truth, two philosophies. The SQL answer is one ready-made report table — Hira's NULL shouting *no record*. The MongoDB answer is the raw material: four honest fee documents, and Hira's absence visible only when you go looking (MG.8 shows how `$lookup` makes her absence speak). If the app prints fee reports all day, SQL's way is home. If the app mostly reads one student at a time, documents win.

Example 3 — Change arrives: add an email field

```
// SQL — one ceremony, every row changes together
ALTER TABLE students ADD COLUMN email VARCHAR(100);

// MongoDB — one document grows, the rest are untouched
db.students.updateOne(
  { name: 'Ali Raza' },
  { $set: { email: 'ali.raza@example.com' } }
)
```

What it does: The school app now collects emails. In SQL the table itself must change first — one `ALTER TABLE` gives *every* row an (empty) email column, then data fills in. In MongoDB, Ali's document simply grows a field the day Ali provides one; the other students gain theirs whenever they provide theirs — no ceremony, no empty column carried by all.

What you will see:

```
Query OK, 0 rows affected

{ acknowledged: true, matchedCount: 1, modifiedCount: 1, upsertedId: null, upsertedCount: 0 }
```

Neither is free: SQL pays the cost once, up front, and gains a guarantee — *every* student row has an email slot, forever. MongoDB pays nothing today, and the cost arrives later as a question — *which documents have email at all?* (`$exists` from MG.4 is the answer tool). Ceremony buys certainty; flexibility buys speed. Mature developers name the price out loud before choosing.

Quick recap

- **Shape first:** fixed known shape → SQL; varying/evolving shape → MongoDB.
- **Connections second:** questions that join everything, every day → SQL; screens that read one self-contained record → MongoDB documents.
- SQL's strictness is a guarantee bought with ceremony; MongoDB's freedom is speed bought with self-discipline (Mongoose, MG.5).
- Both scale, both transact, both are professional tools. Many real apps use **both** — core records in SQL, flexible content in MongoDB.

Common mistakes

- **Choosing by fashion or by the last video watched:** "MongoDB is modern" and "SQL is old" are marketing, not engineering. Run the three decision questions (shape? connections? change rate?) and let the data answer.
- **Modelling MongoDB like SQL:** ten tiny collections and a join for every screen rebuilds SQL slower, without its guarantees. If your design joins constantly, that is the data telling you its shape — listen.
  (Roman Urdu: Agar har screen par join chahiye, to data khud keh raha hai — main SQL ki shape ka hoon.)
- **Modelling SQL like MongoDB:** one giant table with a JSON column for everything throws away the JOINs, constraints and reports that are SQL's whole gift. The JSON column is a spice, not the meal.
- **Forgetting the team factor:** the right database is also the one your team (or future you) can operate — backups, hosting, skills. Atlas and the SQL hosts from Phase 10 both count; a brilliant choice nobody can run is a wrong choice.

Practice

1. Decide, with one-line reasons: (a) a bank's accounts and transfers, (b) a clothing shop's product catalogue where every product type has different features, (c) a school's results and fees system, (d) a chat app's messages. Which parts surprised you?
2. Take Example 1's question and change it: *students from Sahiwal with marks above 80, best first*. Write it in both languages from memory, then check against MG.4 and DB.4.

3. Your app's data will gain a *guardian phone* field next month for some students only. Describe exactly what happens in each database on the day the first guardian phone arrives — commands, costs, and what later queries must remember.

Practice quiz

1. Shape decides first. Which shape points to which database?

- Fixed known shape to MongoDB, varying shape to SQL
- Fixed, known, rarely-changing shape to SQL tables; shape that varies per record or evolves weekly to MongoDB documents
- Shape never matters
- Both handle both shapes equally well

Answer: Fixed, known, rarely-changing shape to SQL tables; shape that varies per record or evolves weekly to MongoDB documents

2. Connections decide second. Which question pattern points to MongoDB?

- Daily questions joining many things
- Each screen reading one self-contained record, like a post with its author snapshot, answered in one fetch
- Questions that need transactions only
- Questions with no data at all

Answer: Each screen reading one self-contained record, like a post with its author snapshot, answered in one fetch

3. What is the honest summary of SQL vs MongoDB in real systems?

- Every real system uses only one, and it is always MongoDB
- Most real systems use SQL for the money-and-records core and MongoDB at the flexible edges; many apps run both, choosing per app, not by religion
- SQL is obsolete
- MongoDB cannot transact at all

Answer: Most real systems use SQL for the money-and-records core and MongoDB at the flexible edges; many apps run both, choosing per app, not by religion

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/mongodb-sql-vs-mongodb · More free tools and notes at https://scopetreasure.com