
✅ Sample MongoDB Database — `schoolDB`

Collection: `students`

```
[
  { name: "Arjun", class: 10, marks: 89, subjects: ["Math", "Science"],
gender: "M", passed: true },
  { name: "Sneha", class: 10, marks: 95, subjects: ["English", "Science"],
gender: "F", passed: true },
  { name: "Rahul", class: 10, marks: 76, subjects: ["Math", "History"],
gender: "M", passed: false },
  { name: "Anjali", class: 10, marks: 88, subjects: ["English", "Math"],
gender: "F", passed: true },
  { name: "Ravi", class: 10, marks: 92, subjects: ["Science", "History"],
gender: "M", passed: true },
  { name: "Sanjuj", class: 10, marks: 69, subjects: ["Math"], gender: "M",
passed: false }
]
```

Collection: `teachers`

```
[
  { name: "Mr. Raj", subject: "Math", salary: 50000 },
  { name: "Ms. Latha", subject: "Science", salary: 52000 },
  { name: "Mr. Vivek", subject: "English", salary: 48000 },
  { name: "Ms. Jaya", subject: "History", salary: 47000 }
]
```

📋 MongoDB Questions List (Arranged by Category)

🧮 Marks-Based Queries

1. Find the **average marks** in class 10.
2. Find the **second highest mark** in class 10.
3. **Increase all marks by 10%** in class 10.
4. **Reduce all teacher salaries by 10%.**
5. Find **students who scored more than the class average** (use `$expr`).
6. Find **students who have passed** and scored **more than 85**.
7. Find the **maximum and minimum marks** in class 10 (`$max`, `$min`).

🧩 Operator-Based Practice

8. Use `$group` to **group students by gender** and get the **average marks** for each gender.
9. Use `$exists` to **find students who have the "passed" field**.

10. Use `$elemMatch` to **find students who have both "Math" and "Science"** as subjects.
11. Use `$expr` to **compare fields inside a document**, e.g., `marks > 70 && passed == true`.
12. Use `$regex` to find **students whose names end with "j"**.
13. Use `$in` to find **students who study either Math or History**.

🧰 Advanced Aggregation & Bulk Operations

14. Use `$facet` to get:
    - Total number of students
    - Average marks
    - List of students who failed
15. Use `bulkWrite()` to:
    - Increase marks of all students by 10
    - Mark all who scored < 75 as `passed: false`
16. Use `$lookup` to join:
    - Students with teachers by matching `subjects` (array in students with subject in teachers)
17. Use `createView` to create a view for:
    - Students with marks greater than 85
18. Use `$project` to return only `name` and `marks` of all students.
19. Add a new field `"status": "active"` to all students.
20. Add a new field `"rank"` based on sorted marks descending (optional challenge).

🧮 Marks-Based Queries

---

1. Average marks in class 10

```
db.students.aggregate([
  { $match: { class: 10 } },
  { $group: { _id: null, avgMarks: { $avg: "$marks" } } }
])
```

---

2. Second highest mark in class 10

```
db.students.aggregate([
  { $match: { class: 10 } },
  { $sort: { marks: -1 } },
  { $skip: 1 },
  { $limit: 1 },
  { $project: { _id: 0, secondHighest: "$marks" } }
])
```

---

3. Increase all marks by 10%

```
db.students.updateMany(
  {},
  [{ $set: { marks: { $multiply: ["$marks", 1.1] } } }]
)
```

---

4. Reduce all teacher salaries by 10%

```
db.teachers.updateMany(
  {},
  [{ $set: { salary: { $multiply: ["$salary", 0.9] } } }]
)
```

---

5. Students who scored above class average (using $expr)

```
const avg = db.students.aggregate([
  { $group: { _id: null, avgMarks: { $avg: "$marks" } } }
]).toArray()[0].avgMarks;

db.students.find({ $expr: { $gt: ["$marks", avg] } })
```

---

6. Students who passed and scored more than 85

```
db.students.find({ passed: true, marks: { $gt: 85 } })
```

---

7. Max and Min marks in class 10

```
db.students.aggregate([
  { $match: { class: 10 } },
  {
    $group: {
      _id: null,
      maxMarks: { $max: "$marks" },
      minMarks: { $min: "$marks" }
    }
  }
])
```

🧩 Operator-Based Practice

---

8. Group by gender and get average marks

```
db.students.aggregate([
  {
    $group: {
      _id: "$gender",
      avgMarks: { $avg: "$marks" }
    }
  }
])
```

---

9. Students where "passed" field exists

```
db.students.find({ passed: { $exists: true } })
```

---

10. Students having both "Math" and "Science" in subjects

```
db.students.find({
  subjects: { $all: ["Math", "Science"] }
})
```

---

11. Using $expr: marks > 70 && passed == true

```
db.students.find({
  $expr: {
    $and: [
      { $gt: ["$marks", 70] },
      { $eq: ["$passed", true] }
    ]
  }
})
```

---

12. Names ending with "j"

```
db.students.find({
  name: { $regex: /j$/i }
})
```

---

13. Students who study either Math or History

```
db.students.find({
  subjects: { $in: ["Math", "History"] }
})
```

🧰 Advanced Aggregation & Bulk Operations

---

14. $facet with multiple results

```
db.students.aggregate([
  {
    $facet: {
      totalCount: [{ $count: "total" }],
      averageMarks: [{ $group: { _id: null, avg: { $avg: "$marks" } } }],
      failedStudents: [{ $match: { passed: false } }]
    }
  }
])
```

---

15. BulkWrite: Increase marks and mark failed if < 75

```
db.students.bulkWrite([
  {
    updateMany: {
      filter: {},
      update: [{ $set: { marks: { $multiply: ["$marks", 1.1] } } }]
    }
  },
  {
    updateMany: {
      filter: { marks: { $lt: 75 } },
      update: { $set: { passed: false } }
    }
  }
])
```

---

16. $lookup: Join students with matching teachers by subject

```
db.students.aggregate([
  { $unwind: "$subjects" },
  {
    $lookup: {
      from: "teachers",
      localField: "subjects",
      foreignField: "subject",
      as: "subjectTeachers"
    }
  }
])
```

---

17. createView: Students with marks > 85

```
db.createView(
  "topScorers",
  "students",
  [
    { $match: { marks: { $gt: 85 } } }
  ]
)
```

---

18. Project name and marks only

```
db.students.find({}, { name: 1, marks: 1, _id: 0 })
```

---

19. Add a field `status: "active"` to all students

```
db.students.updateMany({}, { $set: { status: "active" } })
```

---

20. Add a "rank" based on descending marks (challenge)

```
const rankedStudents = db.students.aggregate([
  { $sort: { marks: -1 } },
  {
    $group: {
      _id: null,
      students: { $push: "$$ROOT" }
    }
  }
]).toArray()[0].students;

rankedStudents.forEach((student, index) => {
  db.students.updateOne({ _id: student._id }, { $set: { rank: index + 1 }
});
});
```

Let me know if you want this saved as a `.js` or `.json` file for MongoDB Compass / CLI, or if you'd like to **add user input through forms or APIs (Node.js)** for this database.