Section 1: Working with Schemas & Data Types

1. Create a database named trainingdb

```
use trainingdb
```

2. Create a collection employees

```
db.createCollection("employees")
```

3. Insert 4 employees with varying skill sets and joining dates

```
db.employees.insertMany(
 [
{name: "Tharun", age: 21, isManager: false, skills: ["ML", "DL"], joiningDate: new Date("2025-05-23"),
profile: {firstName: "Tharun", linkedin: "url"}},
{name: "Atithya", age:23, isManager: true, skills: ["Digital Marketing", "Graphics Designing"],
joiningDate: new Date("2025-04-03"), profile: {firstName: "Aadhi", linkedin: "url_1"}},
{name: "Dhoni", age:41, isManager:true, skills: ["Team Leader", "Batsman"], joiningDate: new
Date("2025-09-03"), profile: {firstName: "MSD", linkedin: "url"}},
{name: "Virat Kholi", age: 37, isManager: false, skills: ["Bowler", "Data Scientist"], joiningDate: new
Date("2025-06-07"), profile: {firstName: "Virat", linkedin: "url"}}
]
)
```

4. Query all employees who Have more than 2 skills, Joined after a specific date.

```
db.employees.find(
  {$expr:
  {$and: [
    {$gte: [{$size: "$skills"}, 2]},
    {$gte: ["$joiningDate", new Date("2025-01-23")]}
]}
}
)
```

5. Add a field rating (float) to one employee.

```
db.employees.updateOne({name: "Tharun"}, {$set: {rating: 8.5}})
```

6. Find all employees with rating field of type double.

```
db.employees.find({rating: {$exists: true, $type: "double"}})
```

7. Exclude the _id field in a query result and show only name and skills.

```
db.employees.find({}, {_id: false, name: true, skills: true})
```

Section 2: One-to-One (Embedded)

1. Create a database schooldb

```
use schooldb
```

2. In the students collection, insert 3 student documents with:

Embedded guardian sub-document (name , phone , relation)

```
db.createCollection("students")
```

```
db.students.insertMany(
[
{name: "Eren", guardian: {name: "Sasha", phone: "+91 93843 94923", relation: "Mother"}},
{name: "Asta", guardian: {name: "Julicus", phone: "+91 95835 34586", relation: "Grandfather"}},
{name: "Ichigo", guardian: {name: "Yhwach", phone: "+91 94932 84593", relation: "Father"}}
]
)
```

3. Query students where the guardian is their "Mother"

```
db.students.find({"guardian.relation": "Mother"})
```

4. Update the guardian's phone number for a specific student

```
db.students.updateOne({name: "Eren"}, {$set: {"guardian.phone": "+82 94953 93943"}})
```

Section 3: One-to-Many (Embedded)

1. In the same schooldb , create a teachers collection

```
use schooldb
```

```
db.createCollection("teachers")
```

2. Insert documents where each teacher has an embedded array of classes they teach (e.g., ["Math", "Physics"])

```
db.teachers.insertMany(
[
{name: "Izen", department: "Psylogical", classes: ["Biology", "Math", "Physics"]},
{name: "Yami", department: "Information Technology", classes: ["DSA", "Foundational AI"]},
]
)
```

3. Query teachers who teach "Physics"

```
db.teachers.find({classes: "Physics"})
```

4. Add a new class "Robotics" to a specific teacher's classes array

```
db.teachers.updateOne({name: "Yami"}, {$push: {classes: "Robotics"}})
```

5. Remove "Math" from one teacher's class list

```
db.teachers.updateOne({name: "Izen"}, {$pull: {classes: "Math"}})
```

Section 4: One-to-Many (Referenced)

1. Create a database academia

```
use academia
```

2. Insert documents into courses with fields:

_id , title, credits

```
db.createCollection("courses")
```

```
db.courses.insertMany(
[
  {_id: ObjectId("64d2a8f8c2a6c3c1a1b2c6d4"), title: "IOT", credits: 4},
  {_id: ObjectId("64d2a8f8c2a6c3c1a1b2c3d5"), title: "AI", credits: 5},
  {_id: ObjectId("64d2a8f8c2a6c3c1a1b2c3d6"), title: "CSE", credits: 3}
]
)
```

3. Insert documents into students with fields:

name , enrolledCourse (store ObjectId reference to a course)

```
db.students.insertMany(
[
  {name: "Luffy", enrolledCourse: ObjectId('64d2a8f8c2a6c3c1a1b2c3d5')},
  {name: "Light Yagami", enrolledCourse: ObjectId('64d2a8f8c2a6c3c1a1b2c6d4')}
]
)
```

4. Query students who are enrolled in a specific course (by ObjectId)

```
db.students.find({enrolledCourse: ObjectId('64d2a8f8c2a6c3c1a1b2c6d4')})
```

5. Query the course details separately using the referenced _id

```
db.students.aggregate(
[
{$lookup: {from: "courses", localField: "enrolledCourse", foreignField: "_id", as: "result"}}
]
)
```

Section 5: $lookup (Join in Aggregation)

1. Use the academia database

```
use academia
```

2. Use $lookup to join students with courses based on enrolledCourse

```
db.students.aggregate(
[
{$lookup: {from: "courses", localField: "enrolledCourse", foreignField: "_id", as: "result"}}
]
)
```

3. Show only student name , and course title in the output using $project

```
db.students.aggregate(
[
{$lookup: {from: "courses", localField: "enrolledCourse", foreignField: "_id", as: "info"}},
{$project: {_id: 0, name: 1, title: "$info.title"}}
]
)
```

4 Add a $match after $lookup to get only students enrolled in "Machine Learning" course

```
[
{$lookup: {from: "courses", localField: "enrolledCourse", foreignField: "_id", as: "info"}},
{$match: {"info.title": {$in: ["Machine Learning"]}}}
]
)
```