MongoDB Schema & Relationships Exercise Set

Section 1: Working with Schemas & Data Types

1 Create a database named trainingdb

use trainingdb

2 Create a collection employees with documents that include:

name (string)

age (number)

isManager (boolean)

skills (array of strings)

joiningDate (date)

profile (sub-document with linkedin and portfolio)

```
db.employees.insertMany([
...  {name: "Nams Ramita",
...   age: 22,
...   isManager: true,
...   skills: ["Python", "Leadership"],
...   joiningDate: new Date("2024-04-12"),
...   profile: {
...    linkedin: "linkedin.com/namsramesh",
...    portfolio: "namsramesh.dev"
...   }},
...  {name: "Mary Saloni",
...   age: 23,
...   isManager: false,
...   skills: ["Java", "SQL"],
...   joiningDate: new Date("2023-08-12"),
...   profile: {
...    linkedin: "linkedin.com/marysaloni",
...    portfolio: "marysaloni.dev"
...   }},
...  {name: "Anon Chan",
...   age: 35,
...   isManager: false,
...   skills: ["Python", "Data Analysis", "Machine Learning"],
...   joiningDate: new Date("2019-03-10"),
```

```
...     profile: {
...       linkedin: "linkedin.com/anonchan",
...       portfolio: null}},
...   {name: "Kurosaki Isagi",
...     age: 30,
...     isManager: true,
...     skills: ["Project Management", "Agile", "Scrum"],
...     joiningDate: new Date("2015-11-05"),
...     profile: {
...       linkedin: "linkedin.com/kurosaki",
...       portfolio: "kurosaki.com"}}])
```

3 Insert 4 employees with varying skill sets and joining dates

```
db.employees.insertMany([
...   {name: "Nams Ramita",
...     age: 22,
...     isManager: true,
...     skills: ["Python", "Leadership"],
...     joiningDate: new Date("2024-04-12"),
...     profile: {
...       linkedin: "linkedin.com/namsramesh",
...       portfolio: "namsramesh.dev"}},
...   {name: "Mary Saloni",
...     age: 23,
...     isManager: false,
...     skills: ["Java", "SQL"],
...     joiningDate: new Date("2023-08-12"),
...     profile: {
...       linkedin: "linkedin.com/marysaloni",
...       portfolio: "marysaloni.dev" }},
...   {name: "Anon Chan",
...     age: 35,
...     isManager: false,
...     skills: ["Python", "Data Analysis", "Machine Learning"],
...     joiningDate: new Date("2019-03-10"),
...     profile: {
...       linkedin: "linkedin.com/anonchan",
```

```
...     portfolio: null }},
...  {name: "Kurosaki Isagi",
...   age: 30,
...   isManager: true,
...   skills: ["Project Management", "Agile", "Scrum"],
...   joiningDate: new Date("2015-11-05"),
...   profile: {
...    linkedin: "linkedin.com/kurosaki",
...    portfolio: "kurosaki.com"}}])
```

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

```
db.employees.find({
 skills: { $size: { $gt: 2 } },
 joiningDate: { $gt: new Date("2020-01-01") }})
```

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

```
db.employees.updateOne(
...  { name: "Nams Ramita" },
...  { $set: { rating: 4.5 }
})
```

6 Find all employees with rating field of type double

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

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

```
db.employees.find(
...  {},
...  { _id: 0, name: 1, skills: 1
})
```

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.students.insertMany([
...  {
...    name: "Zeke Yaegar",
...    guardian: {
...      name: "Eren Yaegar",
...      phone: "92482928392",
...      relation: "Father"
...    }
...  },
...  {name: "Ahmed Sherif",
...    guardian: {
...      name: "Yoruichi",
...      phone: "9274282948",
...      relation: "Mother"}},
...  {
...    name: "Kaiser",
...    guardian: {
...      name: "Isagi Yochi",
...      phone: "9573842839",
...      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: "Kaiser" },
...  { $set: { "guardian.phone": "97347328328" } }
... )
```

Section 3: One-to-Many (Embedded)

1 In the same schooldb , create a teachers collection and Insert documents where each teacher has an embedded array of classes they teach (e.g., ["Math", "Physics"])

```
db.teachers.insertMany([
 {
  name: "Prof. Levi",
  classes: ["Math", "Physics"]
 },
 {
  name: "Prof. John Cena",
  classes: ["Physics", "Chemistry"]
 },
 {
  name: "Ms. David",
  classes: ["Biology"]
 }
])
```

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: "Ms. David" },
 { $push: { classes: "Robotics" } }
)
```

5 Remove "Math" from one teacher’s class list

```
db.teachers.updateOne(
 { name: "Prof. Levi" },
 { $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.courses.insertMany([
  { _id: 1, title: "Computer Science", credits: 4 },
  { _id: 2, title: "Mathematics", credits: 3 },
  { _id: 3, title: "Machine Learning", credits: 4 }
])
```

3 Insert documents into students with fields:

name

enrolledCourse (store ObjectId reference to a course)

```
db.students.insertMany([
  { name: "Student A", enrolledCourse: 1 },
  { name: "Student B", enrolledCourse: 2 },
  { name: "Student C", enrolledCourse: 3 },
  { name: "Student D", enrolledCourse: 1 }
])
```

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

```
db.students.find({
  enrolledCourse: 1  // Computer Science
})
```

5 Query the course details separately using the referenced _id

```
db.courses.findOne({ _id: 1 })
```

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: "courseDetails"}})]
```

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

```
db.students.aggregate([
 {  $project: {
    _id: 0,
    name: 1,
    "courseDetails.title": 1}})]
```

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

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