

**INSTRUCTIONS:**

**PRODUCTIVITY TOOLS ARE NOT ALLOWED. STATE ALL YOUR SOURCES WHEN ALLOWED.**

**SHOW SCREENSHOTS OF YOUR QUERIES AND RESULT WHEN APPLICABLE. IF IN DOUBT, CREATE A SCREENSHOT.**

**[VIEW FULL ASSIGNMENT HERE](#)**

**1. During the class, there was a discussion of modelling a dataset as a Matrix and a record as Vector. Describe your current BA64 SQL database mathematically**

In my relational model (SQL), every table in the Homework4/BA64 database can be represented as a matrix, and each record as a row vector:

**- Tables as Matrices:**

- For example, the students table with 9 attributes: (n columns, n features, n attributes)

➔  $A = \{\text{studentID, firstName, lastName, gender, DateOfBirth, major, department, HeightInCm, WeightInPounds}\}$

- If there are m students (m rows), the table is an  $m \times 9$  matrix:  $(m \times n)$

➔  $M_{\text{students}} \in D(\text{ID}) \times D(\text{fName}) \times \dots \times D(\text{weight})$

- Similarly:
  - classes: an  $m \times 3$  matrix.
  - enrollments: an  $m \times 3$  matrix.
  - Grades: an  $m \times 4$  matrix.

**- Records as Vectors**

- Each student record is a 9-dimensional row vector, e.g.:

➔  $R = (1001, \text{"Anh"}, \text{"Nguyen"}, \text{"Male"}, 2002-05-12, \text{"ComputerScience"}, \text{"SchoolofTechnology"}, 172.0, 150.0)$

**- Uniform Rectangular Structure**

- All rows in the students matrix have exactly 9 attributes.
- Each column is typed (e.g., HeightInCm is DECIMAL, DateOfBirth is DATE).
- Missing values are represented by NULL, but the slot in the matrix remains, so the rectangular form is preserved.

**- Inter-table Relationships**

- Although each table individually is rectangular and uniform, they are connected through keys (e.g., studentID in enrollments links to the students table).
- Mathematically, we can see the database as a **set of matrices with foreign-key mappings** that preserve integrity.

## **2. MongoDB relaxes the rectangular shape that an SQL table guarantees and allows a database collection to have an irregular shape.**

### **a. Will your mathematical model work with an irregular data shape like what is possible in MongoDB?**

=> No. The rigid mathematical model of SQL, based on matrices and vectors, does not work effectively and breaks down when applied directly to the irregular shape of MongoDB data.

Why, let me explain more below:

### **b. What are the limitations of your current model in Question 1 under the relaxed constraints of MongoDB?**

- First: *Conflict of Dimensionality*

- In SQL, each table must be a matrix of size  $m \times n$ , where  $n$  (the number of attributes/columns) is fixed for all  $m$  records.
- This means that every row vector has exactly  $n$  dimensions.
- MongoDB, however, as emphasized in class, allows documents in the same collection to have different numbers of attributes.

- Example:

- Document 1 (new student): { studentID: 1, firstName: "Becky", lastName: "Nguyen" } → a 3-dimensional vector.

Document 2 (old student): { studentID: 2 } → a 1-dimensional vector.

- A set of vectors with different dimensionalities cannot be placed into a single rectangular matrix. This immediately breaks the most fundamental SQL assumption.

- Second: *Conflict of Attribute Semantics*

Even when two documents have the same number of attributes, those attributes may represent entirely different concepts.

- Example:

- Document A: { studentID: 3, major: "Finance" }
  - Document B: { studentID: 4, club: "Chess Club" }

Both could be described as 2-dimensional vectors, but the second dimension has different semantics (“major” vs. “club”).

If we attempt to force these into the same matrix column, the meaning of that column collapses, and the matrix loses its analytical value.

- Third: *NULL* vs. *Attribute Absence*

- In SQL, when a value is missing, it is represented by **NULL**. The column still exists, preserving the rectangular structure.
- In MongoDB, if a document lacks an attribute, that field simply does not exist in the document at all.
- This is not just a missing value—it is a missing dimension. This structural absence cannot be captured by the SQL matrix model.

**c. Can you find a way to overcome this? [EXTERNAL SOURCE IS ALLOWED HERE, STATE YOUR SOURCE]**

We can overcome the limitations of the rigid matrix/vector model by abandoning the flat, rectangular structure and adopting a more flexible mathematical model designed for semi-structured data. A graph-based (or tree-based) representation provides a natural solution.

Proposed Model: Representing Each Document as a Graph or Tree

In this model, each MongoDB document is represented as a rooted, directed graph (or equivalently a tree), where:

- Nodes represent objects or values.
- Edges represent attributes (keys). Each edge is labeled with the key name.

Example:

Consider the following document:

```
{  
  "studentID": 1,  
  "name": "Minh",  
  "contact": { "email": "minh@email.com" },  
  "courses": ["DB101", "CS102"]  
}
```

It can be represented as a labeled graph:

- Root node → edges labeled "studentID", "name", "contact", "courses".

- The "contact" edge points to an internal node, which in turn has an edge "email" pointing to a leaf value.
- The "courses" edge points to an internal node with two child edges (0, 1), each leading to array elements "DB101" and "CS102".

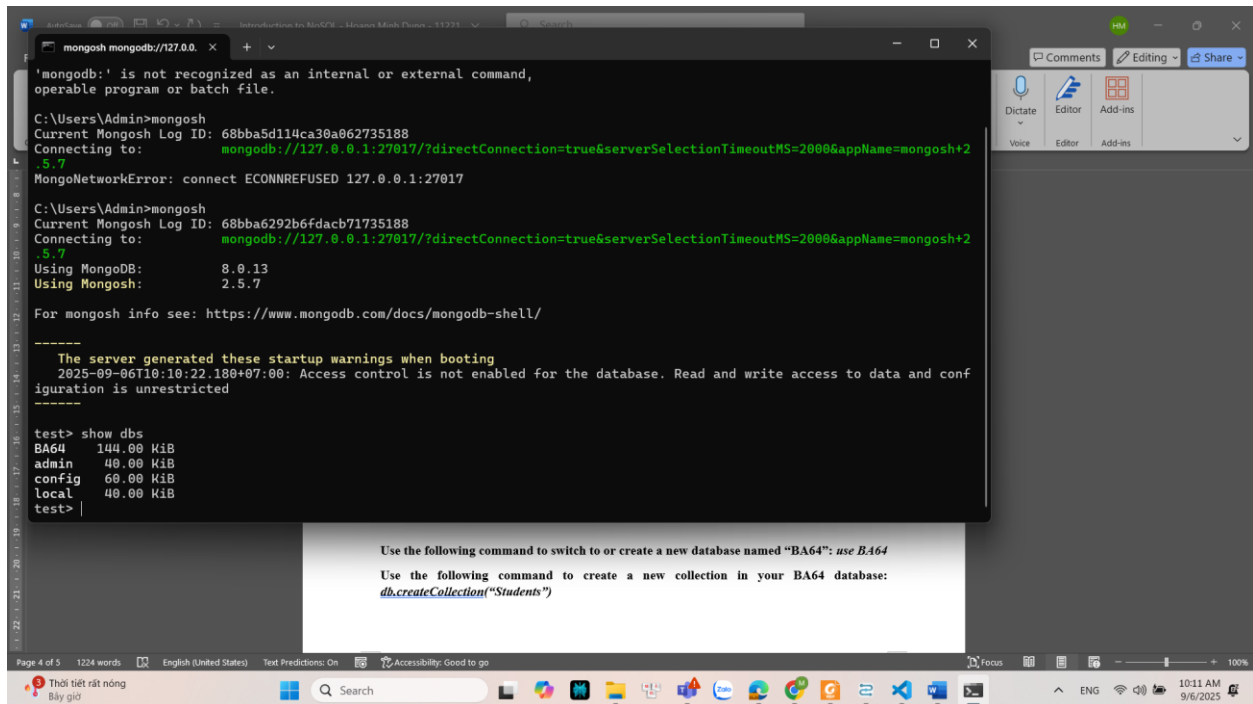
#### How This Model Addresses SQL Matrix Limitations

- Irregular structure (dimensionality):
  - If a document lacks the "contact" field, its graph simply does not include that edge.
  - There is no need for “empty cells” or a sparse matrix.
- Preserving key–value semantics:
  - The edge label is the key, and the node it points to is the value.
  - This ensures that semantic meaning is preserved, even if documents differ.
- Nested objects and arrays:
  - Internal nodes capture sub-documents naturally.
  - Arrays are modeled as nodes with multiple edges, ordered or indexed, pointing to their elements.
  - This provides a faithful mathematical representation of MongoDB’s hierarchical structure.

This graph-based view of semi-structured data is not new. A well-known formalization is the **Object-Exchange Model (OEM)**, introduced by Abiteboul et al. (1997):

- 3. Connect to your MongoDB instance using the MongoDB shell using the following connection string: `mongodb://127.0.0.1:27017`.**

**Run the following command to enumerate the databases that exist on your local machine: `show dbs`**



```
'mongodb:' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Admin>mongosh
Current Mongosh Log ID: 68bba5d114ca38a862735188
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.5.7
MongoNetworkError: connect ECONNREFUSED 127.0.0.1:27017

C:\Users\Admin>mongosh
Current Mongosh Log ID: 68bba6292b6fdacb71735188
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.5.7
Using MongoDB:      8.0.13
Using Mongosh:      2.5.7

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

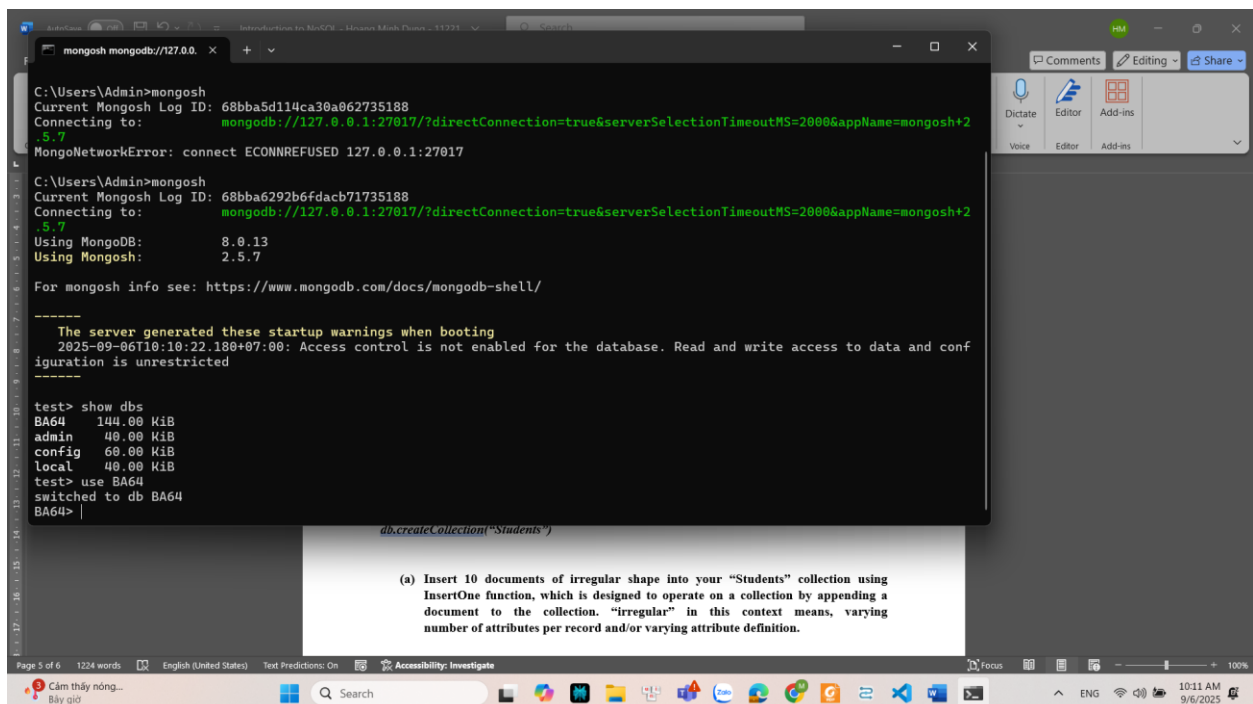
-----
The server generated these startup warnings when booting
2025-09-06T10:10:22.180+07:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
-----

test> show dbs
BA64      144.00 KiB
admin     40.00 KiB
config    60.00 KiB
local     40.00 KiB
test> |
```

Use the following command to switch to or create a new database named “BA64”: *use BA64*

Use the following command to create a new collection in your BA64 database:  
*db.createCollection(“Students”)*

Use the following command to switch to or create a new database named “BA64”: *use BA64*



```
C:\Users\Admin>mongosh
Current Mongosh Log ID: 68bba5d114ca38a862735188
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.5.7
MongoNetworkError: connect ECONNREFUSED 127.0.0.1:27017

C:\Users\Admin>mongosh
Current Mongosh Log ID: 68bba6292b6fdacb71735188
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.5.7
Using MongoDB:      8.0.13
Using Mongosh:      2.5.7

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

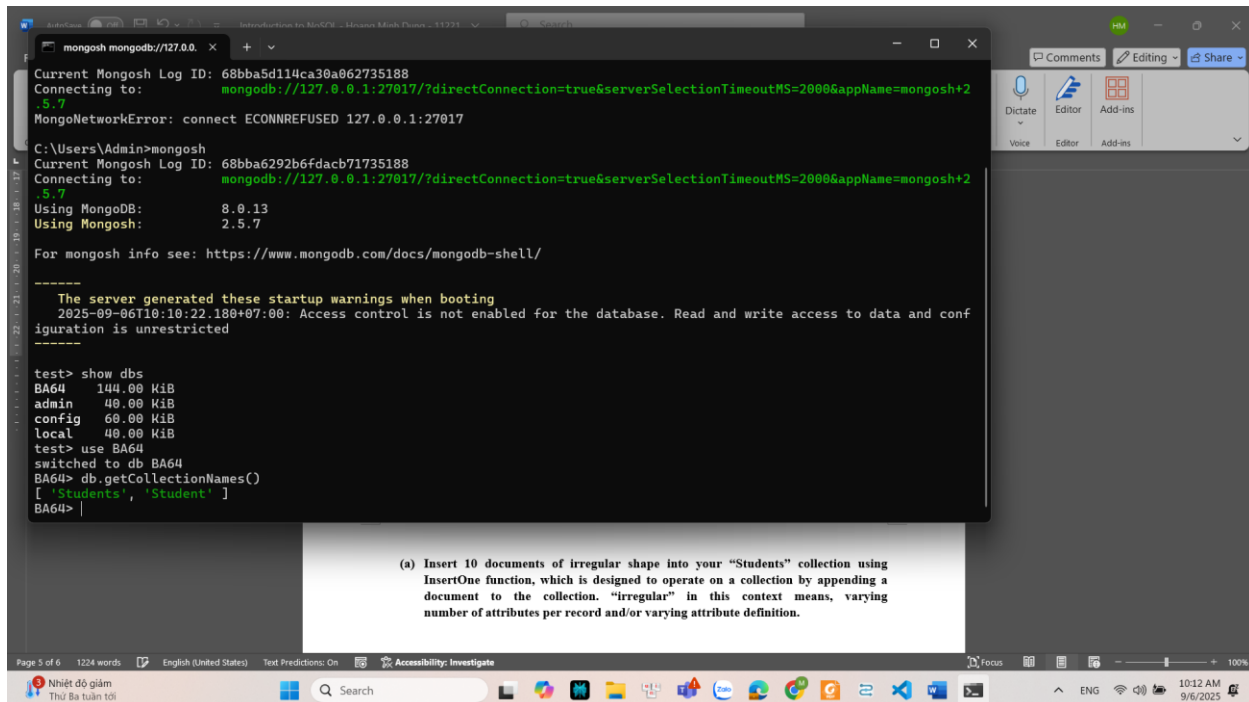
-----
The server generated these startup warnings when booting
2025-09-06T10:10:22.180+07:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
-----

test> show dbs
BA64      144.00 KiB
admin     40.00 KiB
config    60.00 KiB
local     40.00 KiB
test> use BA64
switched to db BA64
BA64> |
```

*db.createCollection(“Students”)*

(a) Insert 10 documents of irregular shape into your “Students” collection using InsertOne function, which is designed to operate on a collection by appending a document to the collection. “irregular” in this context means, varying number of attributes per record and/or varying attribute definition.

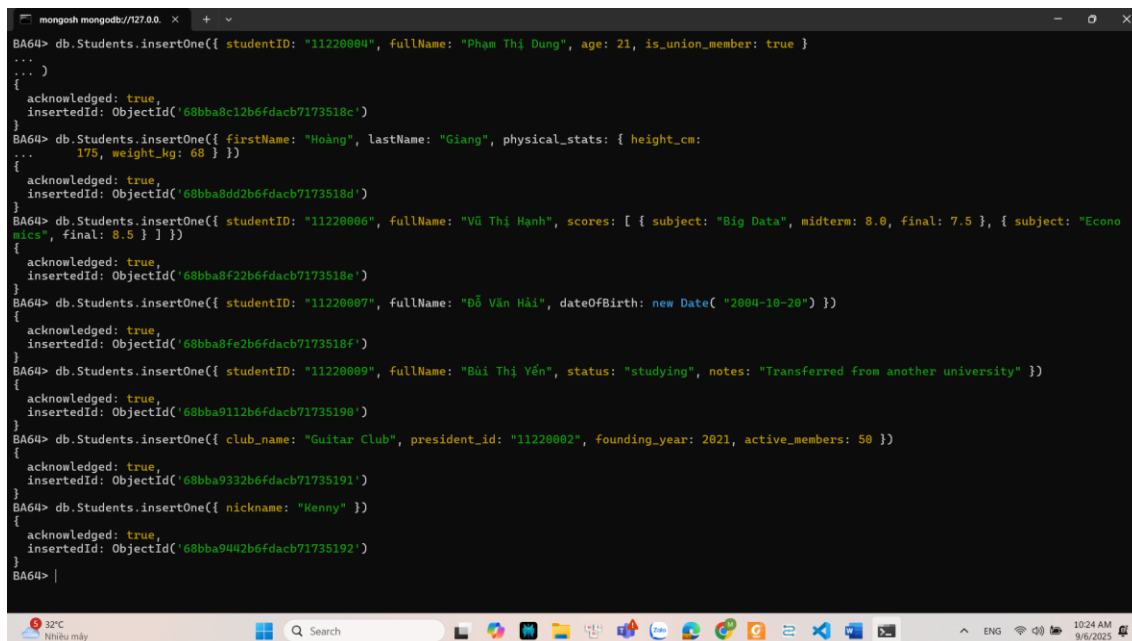
Use the following command to create a new collection in your BA64 database:  
*db.createCollection(“Students”)*



The screenshot shows a MongoDB shell terminal window. The top part displays connection logs for a MongoDB instance at 127.0.0.1:27017. It shows the current MongoDB Log ID, the connection string, and the version of MongoDB (8.0.13) and mongosh (2.5.7). Below the logs, there is a warning message about access control not being enabled for the database. The terminal then shows the output of the 'show dbs' command, listing databases and their sizes. Finally, it shows the output of the 'use BA64' command, indicating that the user has switched to the 'BA64' database. Below the terminal window, there is a text box with the following instruction: (a) Insert 10 documents of irregular shape into your "Students" collection using InsertOne function, which is designed to operate on a collection by appending a document to the collection. "irregular" in this context means, varying number of attributes per record and/or varying attribute definition.

➔ I have already create it

- (a) Insert 10 documents of irregular shape into your “Students” collection using InsertOne function, which is designed to operate on a collection by appending a document to the collection. “irregular” in this context means, varying number of attributes per record and/or varying attribute definition.



The screenshot shows a MongoDB shell terminal window with the following commands and output:

```
BA64> db.Students.insertOne({ studentID: "11220004", fullName: "Phạm Thị Dung", age: 21, is_union_member: true })
{
  acknowledged: true,
  insertedId: ObjectId('68bba8c12b6fdac7173518c')
}
BA64> db.Students.insertOne({ firstName: "Hoàng", lastName: "Giang", physical_stats: { height_cm: 175, weight_kg: 68 } })
{
  acknowledged: true,
  insertedId: ObjectId('68bba8dd2b6fdac7173518d')
}
BA64> db.Students.insertOne({ studentID: "11220006", fullName: "Vũ Thị Hạnh", scores: [ { subject: "Big Data", midterm: 8.0, final: 7.5 }, { subject: "Economics", final: 8.5 } ] })
{
  acknowledged: true,
  insertedId: ObjectId('68bba8f22b6fdac7173518e')
}
BA64> db.Students.insertOne({ studentID: "11220007", fullName: "Đỗ Văn Hải", dateOfBirth: new Date("2004-10-20") })
{
  acknowledged: true,
  insertedId: ObjectId('68bba8fe2b6fdac7173518f')
}
BA64> db.Students.insertOne({ studentID: "11220009", fullName: "Bùi Thị Yến", status: "studying", notes: "Transferred from another university" })
{
  acknowledged: true,
  insertedId: ObjectId('68bba9112b6fdac71735190')
}
BA64> db.Students.insertOne({ club_name: "Guitar Club", president_id: "11220002", founding_year: 2021, active_members: 50 })
{
  acknowledged: true,
  insertedId: ObjectId('68bba9332b6fdac71735191')
}
BA64> db.Students.insertOne({ nickname: "Kenny" })
{
  acknowledged: true,
  insertedId: ObjectId('68bba9442b6fdac71735192')
}
BA64>
```

## db.Students.find()

```
mongosh mongodb://127.0.0.1:27020/
{
  "_id": ObjectId("68b8e442cf6af7677373518e"),
  "StudentId": 3,
  "FirstName": "Dung",
  "LastName": "Hoang",
  "Gender": "Male",
  "HeightInCm": 188,
  "WeightInKg": 80
},
{
  "_id": ObjectId("68b8e442cf6af7677373518f"),
  "StudentId": 4,
  "FirstName": "Minh",
  "LastName": "Hoang",
  "WeightInKg": 80
},
{
  "_id": ObjectId("68bba82e2b6fdacb71735189"),
  "studentID": "11228881",
  "fullName": "Nguyễn Văn An",
  "class": "64A-TKT"
},
{
  "_id": ObjectId("68bba8402b6fdacb7173518a"),
  "studentID": "11228882",
  "fullName": "Trần Thị Bình",
  "contact": { email: "binh.tran@neu.edu.vn", phone: "0912345678" }
},
{
  "_id": ObjectId("68bba8572b6fdacb7173518b"),
  "studentID": "11228883",
  "fullName": "Lê Văn Cường",
  "major": "Computer Science",
  "registered_courses": [ "CS101", "BA202", "EC101" ]
},
{
  "_id": ObjectId("68bba8c12b6fdacb7173518c"),
  "studentID": "11228884",
  "fullName": "Phạm Thị Dung",
  "age": 21,
  "is_union_member": true
},
{
  "_id": ObjectId("68bba8dd2b6fdacb7173518d"),
  "FirstName": "Hoang",
  "lastName": "Giang",
  "physical_stats": { height_cm: 175, weight_kg: 68 }
},
{
  "_id": ObjectId("68bba8f22b6fdacb7173518e"),
  ...
}
```

- (b) Insert at least 500 records into your Students collection using InsertMany function which accepts an ARRAY([]) of BSON documents. You are allowed to generate up to 10 batches containing at least 50 records. Each batch should define an attribute that is not present in other batches.

```
mongosh mongodb://127.0.0.1:27020/
{
  "_id": ObjectId("68bba8dd2b6fdacb7173518d"),
  "FirstName": "Hoang",
  "lastName": "Giang",
  "physical_stats": { height_cm: 175, weight_kg: 68 }
},
{
  "_id": ObjectId("68bba8f22b6fdacb7173518e"),
  "studentID": "11228886",
  "fullName": "Vũ Thị Hạnh",
  "scores": [
    { subject: "Big Data", midterm: 8, final: 7.5 },
    { subject: "Economics", final: 8.5 }
  ]
},
{
  "_id": ObjectId("68bba8fe2b6fdacb7173518f"),
  "studentID": "11228887",
  "fullName": "Ôn Văn Hải",
  "dateOfBirth": ISODate("2004-10-20T00:00:00.000Z")
},
{
  "_id": ObjectId("68bba9112b6fdacb71735190"),
  "studentID": "11228889",
  "fullName": "Bùi Thị Yên",
  "status": "studying",
  "notes": "Transferred from another university"
},
{
  "_id": ObjectId("68bba9332b6fdacb71735191"),
  "club_name": "Guitar Club",
  "president_id": "11228892",
  "founding_year": 2021,
  "active_members": 50
},
{
  "_id": ObjectId("68bba9442b6fdacb71735192"),
  "nickname": "Kenny"
}
]
BA64> load("D:/OneDrive - National Economics University/Documents/Học liệu/Senior - Fourth year/Semester 1/Big Data Analytics/w8/3b.js")
Batch 1 (50 docs with 'legacy_id') inserted.
Batch 2 (50 docs with 'financial_id_status') inserted.
Batch 3 (50 docs with 'library_card_number') inserted.
Batch 4 (50 docs with 'sports_team') inserted.
Batch 5 (50 docs with 'dorm_info') inserted.
Batch 6 (50 docs with 'parking_permit_code') inserted.
Batch 7 (50 docs with 'volunteer_hours') inserted.
Batch 8 (50 docs with 'alumni_mentor_id') inserted.
Batch 9 (50 docs with 'thesis_submission_date') inserted.
Batch 10 (50 docs with 'exchange_program_country') inserted.
--- Total 500 documents inserted successfully! ---
true
BA64> |
```

### Inserting 500 Records in Batches:

The task requires inserting at least 500 records into a collection, divided into batches, with each batch containing a unique attribute. Manually creating and inserting 500 records would be impractical, time-consuming, and error-prone.

- ➔ To ensure accuracy, efficiency, and reproducibility, I adopted a programmatic approach by writing a script that automates the entire process.
- **Description of the 3b.js Script:** The full logic is implemented in the 3b.js file (review it through the link assignment github). The script performs the following tasks:
  - **Automated data generation:** Uses for loops in JavaScript to create 500 student documents.
  - **Division into 10 batches:** Splits the dataset into 10 batches, each containing 50 documents.
  - **Unique attribute per batch:** Each batch is assigned a unique attribute that does not appear in the other nine batches (e.g., Batch 1 → `legacy_id`, Batch 2 → `financial_aid_status`, etc.).
  - **Data insertion:** Uses the `db.Students.insertMany()` function to insert each batch into the collection.
- **Execution and Evidence:** The script was executed in the Mongo Shell using the command:

`load("D:/OneDrive - National Economics University/Documents/Học liệu/Senior - Fourth year/Semester 1/Big Data Analytics/w5/3b.js")`. Here I learn an additional function `load()` which can execute a file.

- (c) Using the lecture, write a query to find the subset of your data with the unique attributes **DEFINED** over the entire set of students. Do this for all unique attributes introduced during the batching process.

*For easily read, I've inserted `pretty()` in the end each query*

**Batch 1:** `db.Students.find({ legacy_id: { $exists: true } }).pretty()`



```
--- Total 500 documents inserted successfully! ---
true
BA64> db.Students.find({ legacy_id: { $exists: true } }).pretty()
[
  {
    _id: ObjectId('68bbabfc2b6fdacb71735193'),
    studentID: 'LGCY1001',
    fullName: 'Legacy Student 1',
    legacy_id: 'old_sys_1001'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb71735194'),
    studentID: 'LGCY1002',
    fullName: 'Legacy Student 2',
    legacy_id: 'old_sys_1002'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb71735195'),
    studentID: 'LGCY1003',
    fullName: 'Legacy Student 3',
    legacy_id: 'old_sys_1003'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb71735196'),
    studentID: 'LGCY1004',
    fullName: 'Legacy Student 4',
    legacy_id: 'old_sys_1004'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb71735197'),
    studentID: 'LGCY1005',
    fullName: 'Legacy Student 5',
    legacy_id: 'old_sys_1005'
  }
]
```

Batch 2: `db.Students.find({ financial_aid_status: { $exists: true } }).pretty()`

```
BA64> db.Students.find({ financial_aid_status: { $exists: true } }).pretty()
[
  {
    _id: ObjectId('68bbabfc2b6fdacb717351c5'),
    studentID: 'FIN2001',
    fullName: 'Financial Student 1',
    financial_aid_status: 'pending'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb717351c6'),
    studentID: 'FIN2002',
    fullName: 'Financial Student 2',
    financial_aid_status: 'approved'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb717351c7'),
    studentID: 'FIN2003',
    fullName: 'Financial Student 3',
    financial_aid_status: 'pending'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb717351c8'),
    studentID: 'FIN2004',
    fullName: 'Financial Student 4',
    financial_aid_status: 'approved'
  }
]
```

**Batch 3: db.Students.find({ library\_card\_number: { \$exists: true } }).pretty()**

```
]
Type "it" for more
BA64> db.Students.find({ library_card_number: { $exists: true } }).pretty()
[
  {
    _id: ObjectId('68bbabfc2b6fdacb717351f7'),
    studentID: 'LIB3001',
    fullName: 'Library User 1',
    library_card_number: 'CARD-3001'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb717351f8'),
    studentID: 'LIB3002',
    fullName: 'Library User 2',
    library_card_number: 'CARD-3002'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb717351f9'),
    studentID: 'LIB3003',
    fullName: 'Library User 3',
    library_card_number: 'CARD-3003'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb717351fa'),
    studentID: 'LIB3004',
    fullName: 'Library User 4',
    library_card_number: 'CARD-3004'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb717351fb'),
    studentID: 'LIB3005',
    fullName: 'Library User 5',
    library_card_number: 'CARD-3005'
  }
]
```

**Batch 4: db.Students.find({ sports\_team: { \$exists: true } }).pretty()**

```
mongosh mongodb://127.0.0.1:27020/
BA64> db.Students.find({ sports_team: { $exists: true } }).pretty()
[
  {
    _id: ObjectId('68bbabfc2b6fdacb71735229'),
    studentID: 'SPT4001',
    fullName: 'Athlete Student 1',
    sports_team: 'Soccer'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb7173522a'),
    studentID: 'SPT4002',
    fullName: 'Athlete Student 2',
    sports_team: 'Soccer'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb7173522b'),
    studentID: 'SPT4003',
    fullName: 'Athlete Student 3',
    sports_team: 'Basketball'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb7173522c'),
    studentID: 'SPT4004',
    fullName: 'Athlete Student 4',
    sports_team: 'Soccer'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb7173522d'),
    studentID: 'SPT4005',
    fullName: 'Athlete Student 5',
    sports_team: 'Basketball'
  }
]
```

**Batch 5: db.Students.find({ dorm\_info: { \$exists: true } }).pretty()**

```
BA64> db.Students.find({ dorm_info: { $exists: true } }).pretty()
[
  {
    _id: ObjectId('68bbabfc2b6fdacb7173525b'),
    studentID: 'DRM5001',
    fullName: 'Dorm Resident 1',
    dorm_info: { building: 'D2', room: 101 }
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb7173525c'),
    studentID: 'DRM5002',
    fullName: 'Dorm Resident 2',
    dorm_info: { building: 'D3', room: 102 }
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb7173525d'),
    studentID: 'DRM5003',
    fullName: 'Dorm Resident 3',
    dorm_info: { building: 'D4', room: 103 }
  }
]
```

**Batch 6: db.Students.find({ parking\_permit\_code: { \$exists: true } }).pretty()**

```
Type "it" for more
BA64> db.Students.find({ parking_permit_code: { $exists: true } }).pretty()
[
  {
    _id: ObjectId('68bbabfc2b6fdacb7173528d'),
    studentID: 'PRK6001',
    fullName: 'Commuter Student 1',
    parking_permit_code: 'P-2025-6001'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb7173528e'),
    studentID: 'PRK6002',
    fullName: 'Commuter Student 2',
    parking_permit_code: 'P-2025-6002'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb7173528f'),
    studentID: 'PRK6003',
    fullName: 'Commuter Student 3',
    parking_permit_code: 'P-2025-6003'
  }
]
```

**Batch 7: db.Students.find({ volunteer\_hours: { \$exists: true } }).pretty()**

```
Type "it" for more
BA64> db.Students.find({ volunteer_hours: { $exists: true } }).pretty()
[
  {
    _id: ObjectId('68bbabfc2b6fdacb717352bf'),
    studentID: 'VOL7001',
    fullName: 'Volunteer Student 1',
    volunteer_hours: 2
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb717352c0'),
    studentID: 'VOL7002',
    fullName: 'Volunteer Student 2',
    volunteer_hours: 4
  }
]
```

**Batch 8:** `db.Students.find({ alumni_mentor_id: { $exists: true } }).pretty()`

```
BA64> db.Students.find({ alumni_mentor_id: { $exists: true } }).pretty()
[
  {
    _id: ObjectId('68bbabfc2b6fdacb717352f1'),
    studentID: 'MTR8001',
    fullName: 'Mentee Student 1',
    alumni_mentor_id: 'ALUMNI-81'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb717352f2'),
    studentID: 'MTR8002',
    fullName: 'Mentee Student 2',
    alumni_mentor_id: 'ALUMNI-82'
  },
]
```

**Batch 9:** `db.Students.find({ thesis_submission_date: { $exists: true } }).pretty()`

```
BA64> db.Students.find({ thesis_submission_date: { $exists: true } }).pretty()
[
  {
    _id: ObjectId('68bbabfc2b6fdacb71735323'),
    studentID: 'THS9001',
    fullName: 'Thesis Student 1',
    thesis_submission_date: ISODate('2025-09-06T03:35:24.414Z')
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb71735324'),
    studentID: 'THS9002',
    fullName: 'Thesis Student 2',
    thesis_submission_date: ISODate('2025-09-06T03:35:24.414Z')
  },
]
```

**Batch 10:** `db.Students.find({ exchange_program_country: { $exists: true } }).pretty()`

```
BA64> db.Students.find({ exchange_program_country: { $exists: true } }).pretty()
[
  {
    _id: ObjectId('68bbabfc2b6fdacb71735355'),
    studentID: 'EXC10001',
    fullName: 'Exchange Student 1',
    exchange_program_country: 'Japan'
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb71735356'),
    studentID: 'EXC10002',
    fullName: 'Exchange Student 2',
    exchange_program_country: 'Japan'
  },
]
```

- (d) Using the lecture, write a query to find subset of your data with the unique attributes NOT DEFINED over the entire set of students, for all unique attributes introduced during the batching process.

**General Query Structure for Any Attribute:** `db.Students.find({ attribute_name: { $exists: false } })`

`db.Students.find({ legacy_id: { $exists: false } }).pretty()`

`db.Students.find({ financial_aid_status: { $exists: false } }).pretty()`

`db.Students.find({ library_card_number: { $exists: false } }).pretty()`

`db.Students.find({ sports_team: { $exists: false } }).pretty()`

`db.Students.find({ dorm_info: { $exists: false } }).pretty()`

`db.Students.find({ parking_permit_code: { $exists: false } }).pretty()`

`db.Students.find({ volunteer_hours: { $exists: false } }).pretty()`

`db.Students.find({ alumni_mentor_id: { $exists: false } }).pretty()`

`db.Students.find({ thesis_submission_date: { $exists: false } }).pretty()`

`db.Students.find({ exchange_program_country: { $exists: false } }).pretty()`

```
BA64> db.Students.find({ legacy_id: { $exists: false } }).pretty()
[
  {
    _id: ObjectId('68b8e61dcf6af76773735189'),
    StudentId: 1,
    FirstName: 'Becky',
    LastName: 'Nguyen',
    WeightInKg: 0
  },
  {
    _id: ObjectId('68b8e9e6cf6af7677373518d'),
    StudentId: 2,
    FirstName: 'Tony',
    LastName: 'Jackson',
    Gender: 'Male',
    WeightInKg: 0
  },
  {
    _id: ObjectId('68b8ef42cf6af7677373518e'),
    StudentId: 3,
    FirstName: 'Dung',
    LastName: 'Hoang',
    Gender: 'Male',
    HeightInCm: 180,
    WeightInKg: 0
  }
]
```

### Logical Methodology for Determining the Result Count:

Instead of running 10 separate queries, the number of results can be determined logically as follows:

1. From now, the total number of documents currently in the **Students** collection is:
  - 10 (from Question 3a) + 500 (from Question 3b) = **510 documents**.
2. For any unique attribute created in Question 3b (e.g., legacy\_id), it appears only in the 50 documents of its corresponding batch.
3. Therefore, the number of documents that **do not** contain this attribute can be calculated as:
  - **Documents without the attribute = (Total documents) – (Documents with the attribute) => 460 = 510 – 50**

- (e) Create a subset of your data using the find function over the data space of Students such that it constrains the resulting data space based on TWO attributes of your choice. For example: `db.Students.find({LastName: {$exists: true}, StudentId: {$gt: 1}})`

```
BA64> db.Students.find({
...   "dorm_info.building": "D2",
...   studentID: { $regex: /^DRM5/ }
... }).pretty()
[
  {
    _id: ObjectId('68bbabfc2b6fdacb7173525b'),
    studentID: 'DRM5001',
    fullName: 'Dorm Resident 1',
    dorm_info: { building: 'D2', room: 101 }
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb7173525f'),
    studentID: 'DRM5005',
    fullName: 'Dorm Resident 5',
    dorm_info: { building: 'D2', room: 105 }
  },
  {
    _id: ObjectId('68bbabfc2b6fdacb71735263'),
    studentID: 'DRM5009',
    fullName: 'Dorm Resident 9',
    dorm_info: { building: 'D2', room: 109 }
  },
]
```

- ➔ I used the find() function with two attributes as constraints:
- **dorm\_info.building = "D2"** → restricts results to students living in building D2.
- **studentID with regex /^DRM5/** → selects only students whose ID starts with "DRM5".

This query returns a **subset of students** who both live in building D2 **and** have a studentID beginning with "DRM5". The result is a focused dataset instead of the entire collection.

**(f) Explain why (e) is important in your own words. Write in Vietnamese**

Việc thực hiện được truy vấn ở câu (e) – tức là khả năng lọc dữ liệu dựa trên nhiều thuộc tính đồng thời – là một trong những năng lực cốt lõi và quan trọng bậc nhất của bất kỳ hệ quản trị cơ sở dữ liệu nào, bao gồm cả MongoDB. Tầm quan trọng của nó thể hiện ở ba khía cạnh chính:

Trong thực tế, các câu hỏi chúng ta cần trả lời từ dữ liệu hiếm khi nào chỉ dựa trên một điều kiện duy nhất. Chúng ta không chỉ muốn tìm "tất cả sinh viên", mà thường muốn tìm "tất cả sinh viên khoa Công nghệ thông tin và đang ở ký túc xá tòa D2". Khả năng kết hợp nhiều điều kiện cho phép chúng ta "khoan" sâu vào tập dữ liệu khổng lồ để lấy ra chính xác những thông tin cần thiết, loại bỏ đi những dữ liệu nhiễu không liên quan. Đây là nền tảng để biến dữ liệu thô thành thông tin có giá trị.

**4. From the discussion about the differences between RDBMS and NoSQL, what is your understanding (in your own words) of the main differences between these two database paradigms?**

Based on the discussions and exercises, here is my understanding of the main differences between RDBMS (SQL) and NoSQL databases:

I believe the differences are not only technological but also philosophical, reflecting two fundamentally different approaches to storing, managing, and querying data. I summarize them in four key aspects:

**1. Schema & Data Structure**

- RDBMS (SQL): Like a filing cabinet with pre-labeled drawers. It requires a strict schema. Before inserting any data, the structure of the table, column names, and data types must be defined. Data is highly structured and uniform, ensuring that every row has the same attributes.
- NoSQL (e.g., MongoDB): Like a flexible storage box. It uses a flexible schema. Documents in the same collection can have different structures: one may contain three fields, another ten fields, including nested objects or arrays. This flexibility allows the data model to evolve easily with the application.

**2. Scalability**

- RDBMS (SQL): Typically scales vertically. To handle more workload, the existing server must be upgraded (CPU, RAM, storage). This is costly and limited by hardware. Horizontal scaling (distributing data across multiple servers) is complex due to relational integrity.
- NoSQL (MongoDB): Designed for horizontal scaling from the start. When workload increases, additional low-cost commodity servers can be added to the

cluster. This makes NoSQL highly effective for handling massive datasets and heavy traffic at scale.

### **3. Data Model & Relationships**

- RDBMS (SQL): Data is normalized into multiple related tables to avoid redundancy. These tables are connected through foreign keys, and retrieving complete information often requires expensive JOIN operations.
- NoSQL (MongoDB): Optimized for reads by embedding related data in a single document. For example, a blog post and its comments can be stored together in one document. This eliminates the need for JOINS and improves query performance.

### **4. Consistency**

- RDBMS (SQL): Strictly follows the ACID properties (Atomicity, Consistency, Isolation, Durability). Transactions are all-or-nothing, ensuring strong consistency and reliability, which is crucial for systems such as banking and finance.
- NoSQL (MongoDB): Often follows the BASE model (Basically Available, Soft state, Eventual consistency), prioritizing availability over immediate consistency. Data may take a short time to synchronize across the cluster. Modern NoSQL systems, including MongoDB, now also provide strong ACID transactions, but their original design philosophy emphasizes flexibility and availability.