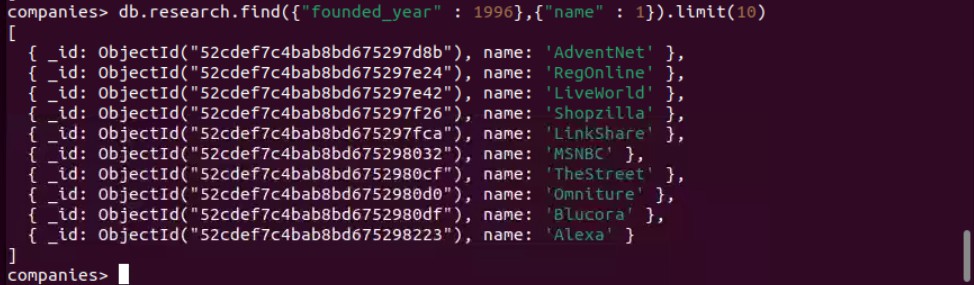
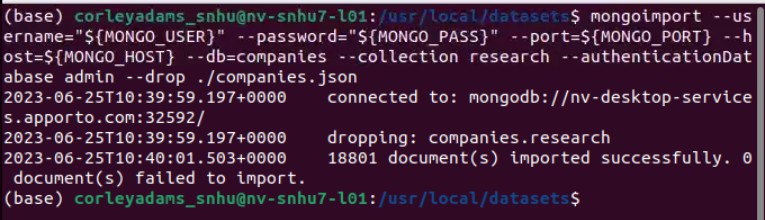
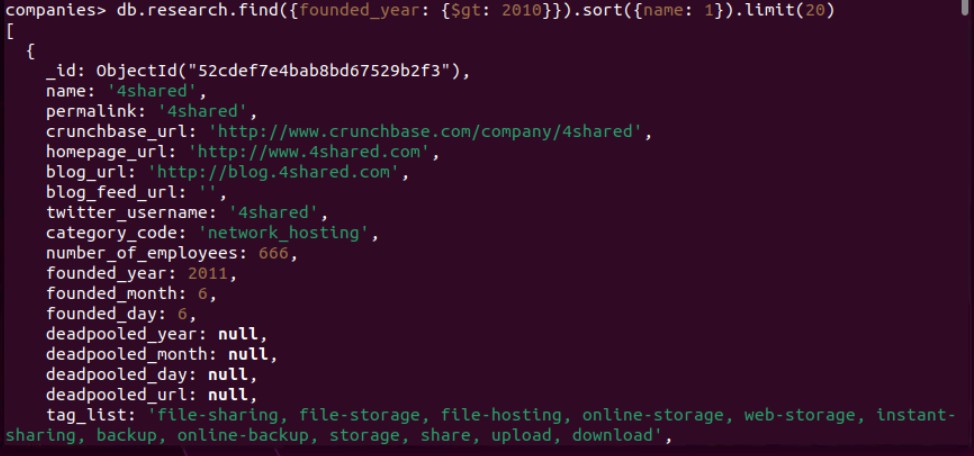
Ques%on 1



Ques%on 2

db.research.ﬁnd({founded\_year: {$gt: 2010}}).sort({name: 1}).limit(20)



db.research.ﬁnd({"oﬃces.state\_code": {"$in": ["CA", "TX"]}}, {"name": 1, "number\_of\_employees": 1, "\_id": 0}).sort({"number\_of\_employees": -1}).limit(20)



Ques%on 3 db.research.aggregate([

{$unwind: "$oﬃces"},

{$match: {"oﬃces.country\_code": "USA"}},

{$group: {\_id: "$oﬃces.state\_code", totalOﬃces: {$sum: 1}}}

])

Explana%on

1. Unwind the offices array, which will create a separate document for each office in each company.
2. Match documents where the offices.country\_code field is equal to "USA".
3. Group the documents by the offices.state\_code field and calculate the total number of offices in each state.



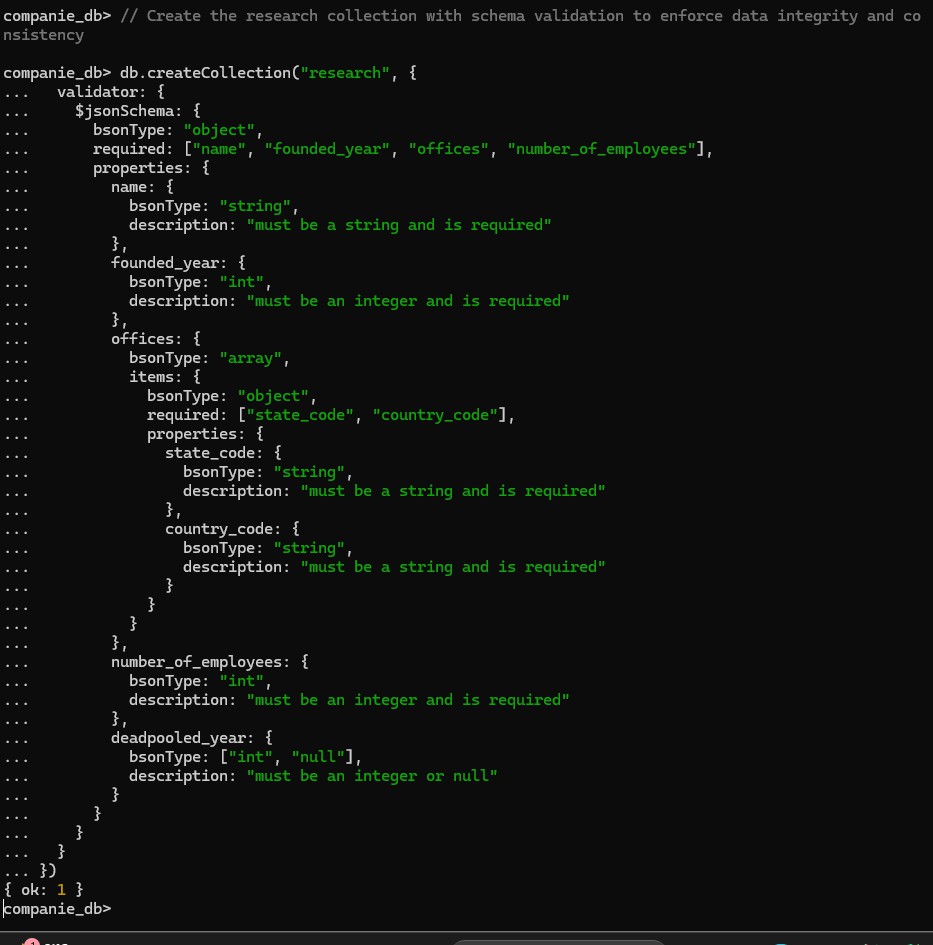
**Advanced enhancements**

## Improve Documentation:

Added detailed comments above each database query to specify what each query will be used for purpose of each query, the logic behind it, and the expected results. This helps developers and users understand the intent and functionality of the code.

## Standardize Naming Conventions:

Ensure that names of fields and collections in use conform to a consistent naming convention in the operations within the database. Such standardization reduces ambiguity and errors, hence making the code easy to maintain and read.



## Error Handling

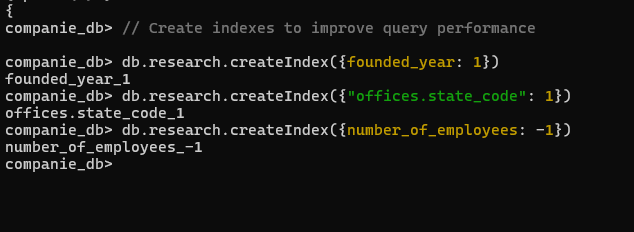
Try-catch blocks have been implemented around all database operations to handle potential errors gracefully. If an error does arise during database interactions, it is caught and an appropriate error message is printed. This prevents a crash and helps during debugging of the application.

**Database Schema Validation**:Added schema validation to the research collection to enforce data integrity and consistency. This schema defines the required fields and their data types, ensuring that only valid data is inserted into the collection.



# Part 1: Create Indexes

**Introduce Indexes**: Created indexes on frequently queried fields such as founded\_year, offices.state\_code, and number\_of\_employees. Indexes enhance query performance by allowing the database to locate and retrieve data more quickly, thus reducing query execution time.



**Optimize Aggregations**:Optimized the aggregation pipeline by using the $project stage to limit the number of fields processed. This reduces the amount of data the database needs to handle, improving the efficiency and speed of aggregation operations.



# Part 2: Example Queries

companie\_

companie\_db> {

db.research.find({t-::L,11,,1, w: {S,it: ,Yl }}).sort({1:11v: 1}).limit(,>O·).forEach(printjson)

} Ce) {

**print(e)**

**companie\_db>**

- **}):-i.imit(**

**companie db>**

**db.research. ind(**

{

{

**).sort({**

Ce)

**print(e)**

]} } '

**. orEach(printjson)**

**name:**

,

**number\_o \_emp1oyees:**

**name:**

**number\_o \_emp1oyees:**

**name:**

**number\_o \_employees:**

**name:**

,

**number\_o \_employees:**

**name:**

,

**number\_o \_employees:**

**name:**

**number\_o \_employees:**

**name:**

**number\_o \_employees:**

**name:**

,

**number\_o \_employees:**

**compa.nie\_db>**

companie\_db>

,

companie\_db>

{

db.research.aggregate([

{ Su ·1 *:* 1 1 I: }

{' 1'1,-\ 1\_ C l: {

{Sq { i I:

{S- vet: {c,t\_.-,t,

]).forEach(printjson)

} Ce) {

print(e)

}

{

}

{

totalOffices: c:.'

state:

totalOffices: ~, state:

}

{

}

totalOffices: state:

}}

, totalOffices:

}},

totalOffices:

1, i I: -·

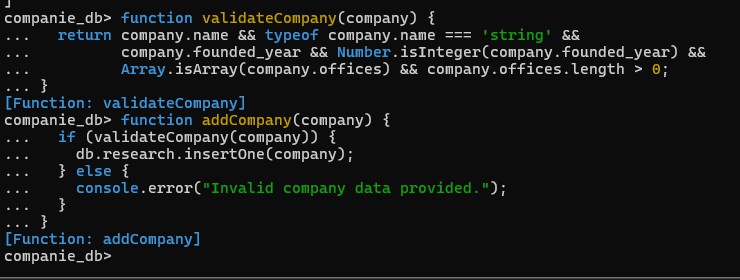
{fc.1,111: 1}}},

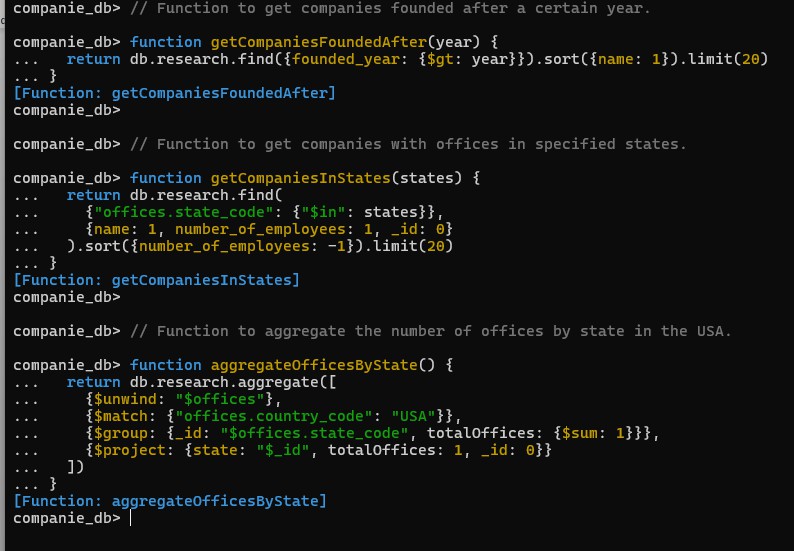
companie\_db>

# Part 3: Modular Functions

## Secure CRUD Operations:

Implemented validation functions to ensure data integrity and prevent injection attacks. The secure insertion logic checks the validity of the data before performing database operations, ensuring only valid data is inserted.





# Part 4: Example Usage of Modular Functions

Refactored complex queries into modular and reusable functions. This improves the maintainability and readability of the code, making it easier to manage and update. Each function handles a specific query, which can be reused across different parts of the application.



**Part 6: Validation Function for a Year**

Included robust input validation to ensure that user inputs are correct and within expected ranges. This prevents potential issues and ensures that the data being processed is valid.

