Document In-Memory Persistent Database

Mahdy Mousa Hamad

Why Build a Database Management System?

- Educational Purposes (Research)
- Paid Service (Business Value)
- Doing Something New
- OpenSource

Document Database Requirements

- Authorization and Authentication

- Flexibility in storing records

- Efficient Queries

- Horizontal Scalability

- Data Sharding

- Persistent Data on-disk

… and more….

Project Scope

- Authentication and Authorization - ACL

- Efficient Queries

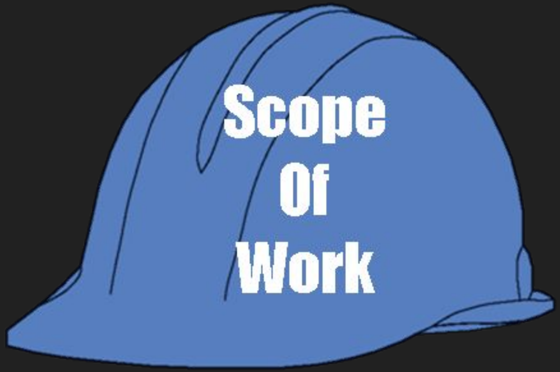
- Persistent Data on-disk

- In-Memory Data Management and Indexing

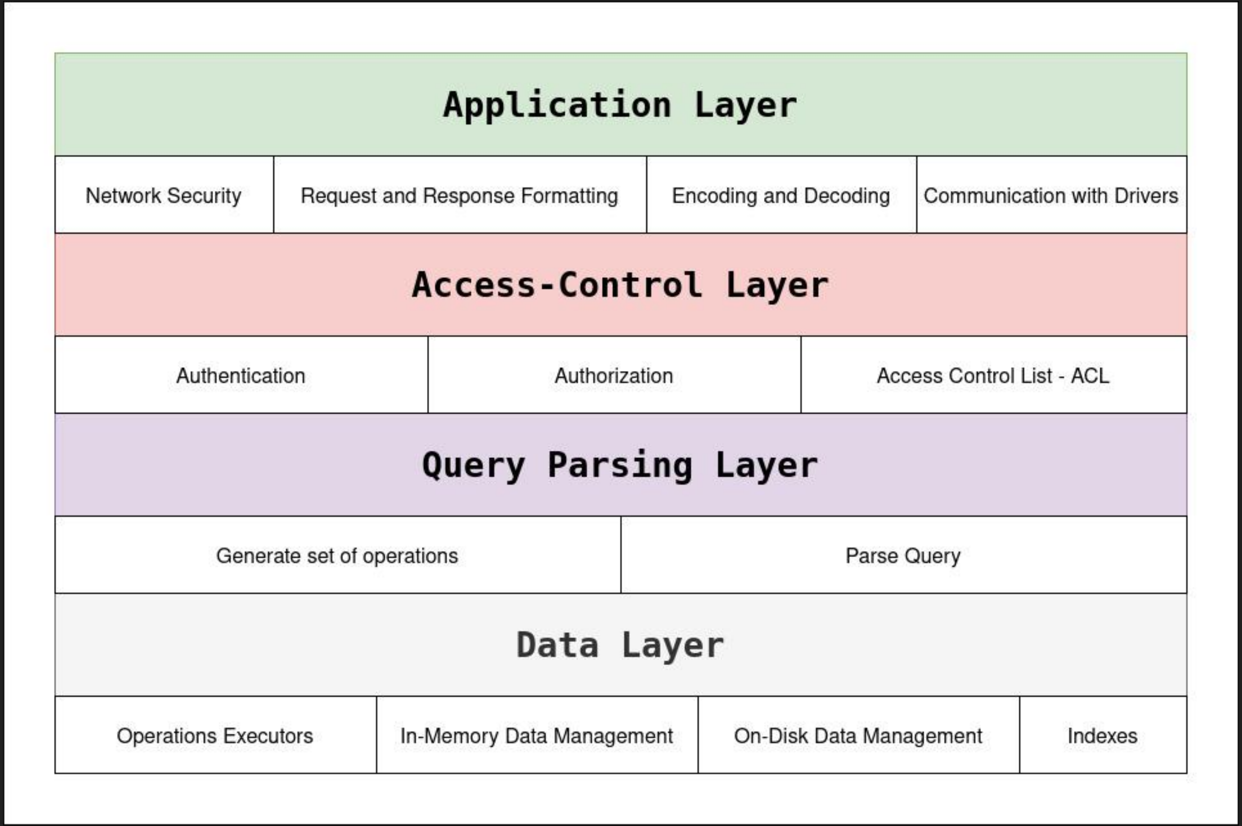
- Simple On-Disk Indexing

- Exposed Network Layer for outside communication

- Simple Shell Client



Database Architecture - Overview



Application Layer

Network Security | Request and Response Formatting | Encoding and Decoding | Communication with Drivers

Access-Control Layer

Authentication | Authorization | Access Control List - ACL

Query Parsing Layer

Generate set of operations | Parse Query

Data Layer

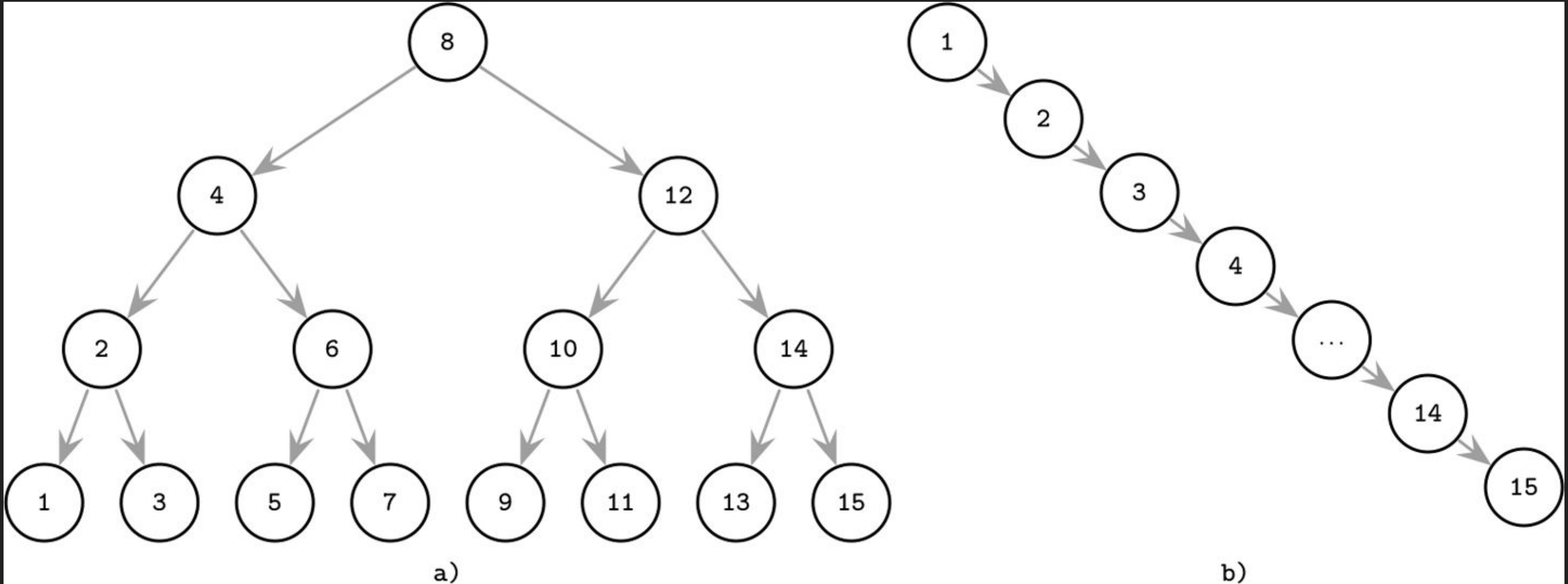
Operations Executors | In-Memory Data Management | On-Disk Data Management | Indexes

How Data is Queried Efficiently?

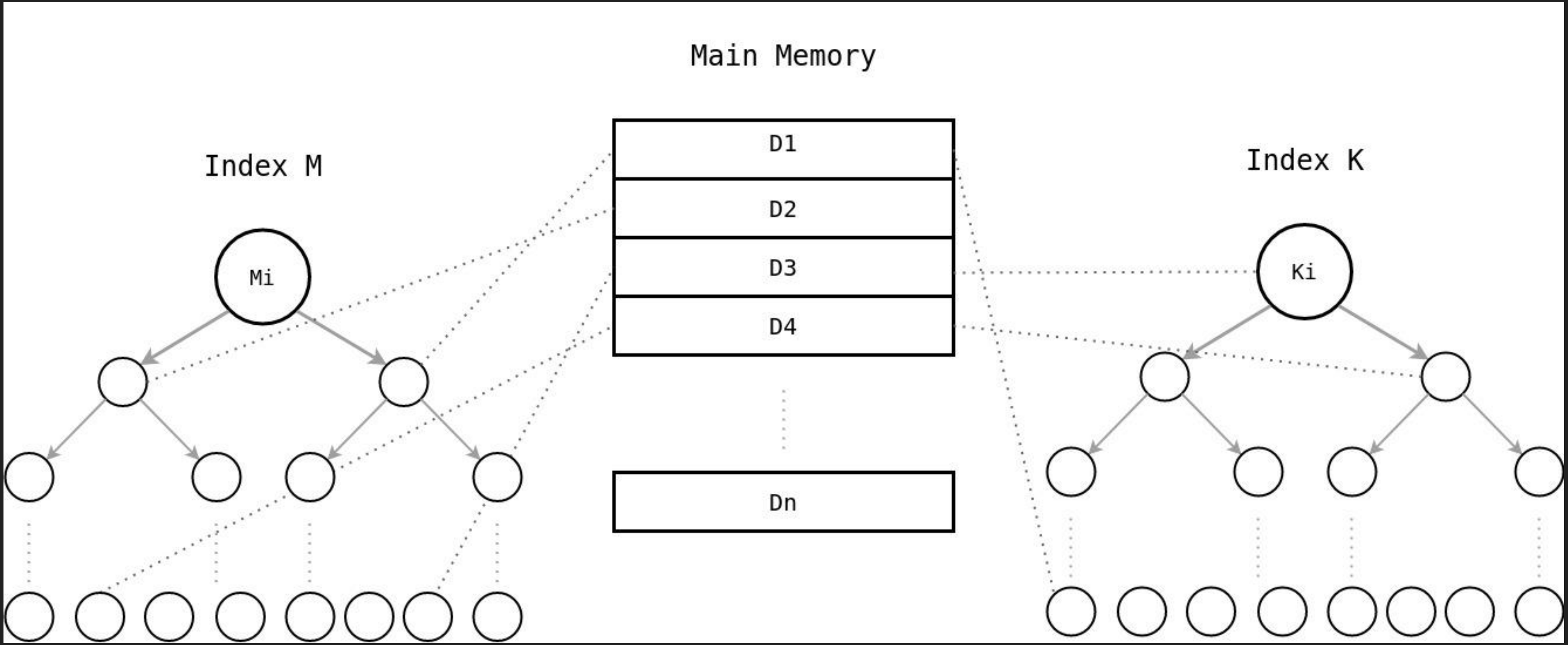
This is done using ***indexing***. In database systems, indexing is a way of storing data (usually in a tree structure) in a way that increase the performance of retrieving data that is associated with a key.

Indexes can be implemented on-disk and in-memory. We will focus on indexes stored in-memory. So data will be loaded into memory as indexes using BSTs (Binary Search Trees).

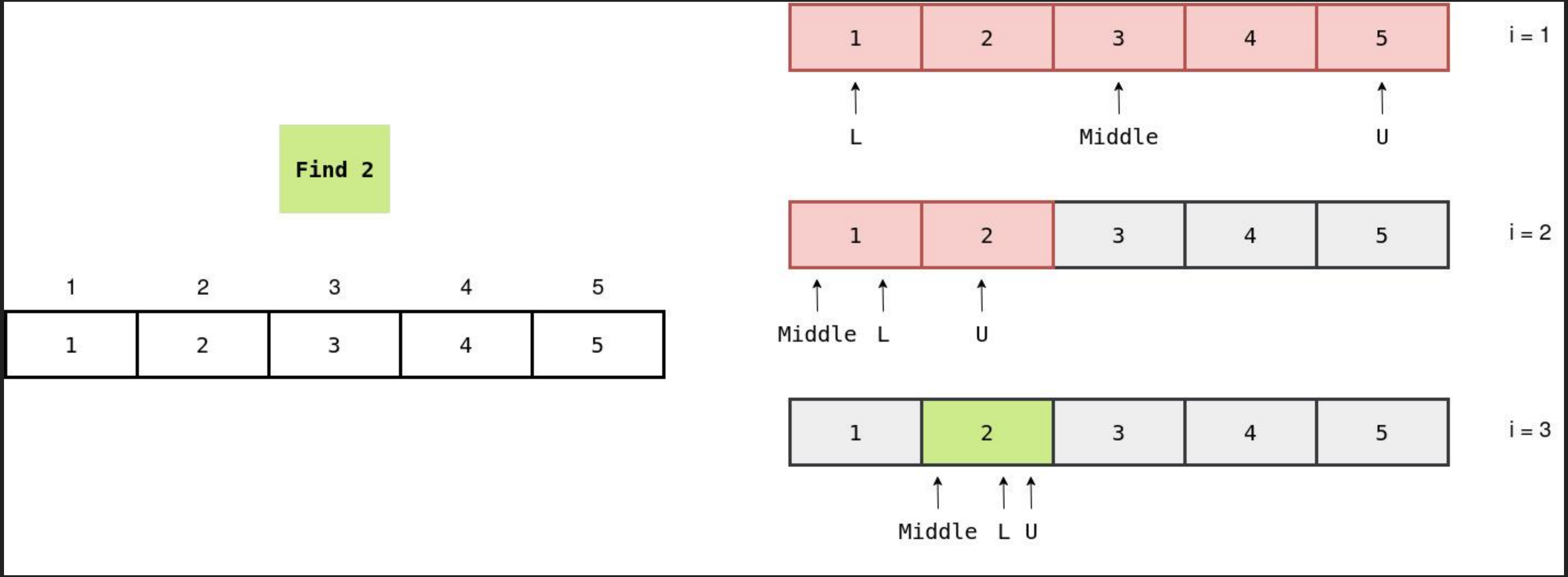
Binary Search Tree - Indexing



In-Memory Indexing



On-Disk Indexing Simple Indexing



Find 2

1	2	3	4	5
1	2	3	4	5

i = 1

1	2	3	4	5

L Middle U

i = 2

1	2	3	4	5

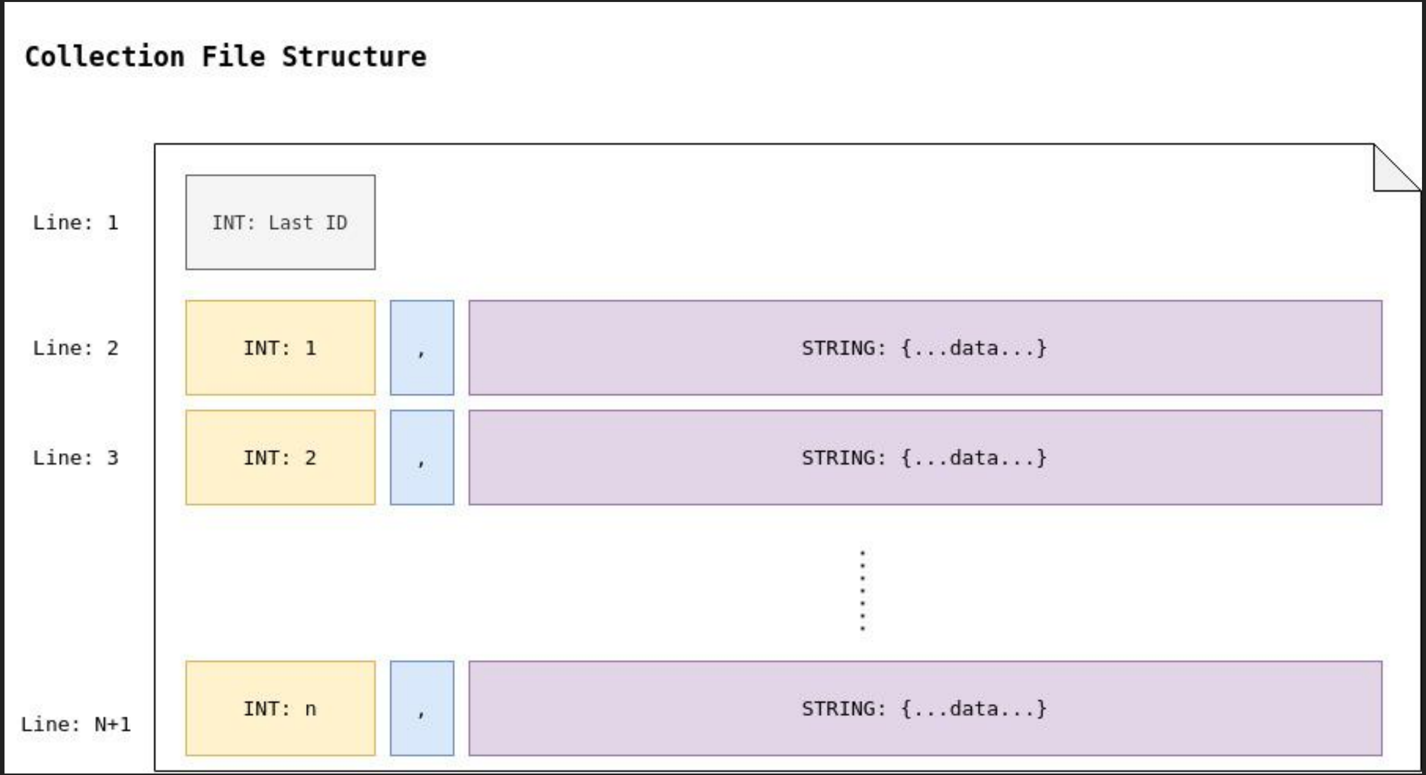
Middle L U

i = 3

1	2	3	4	5

Middle L U

On-Disk Indexing Simple Indexing



Collection File Structure

Line: 1 INT: Last ID

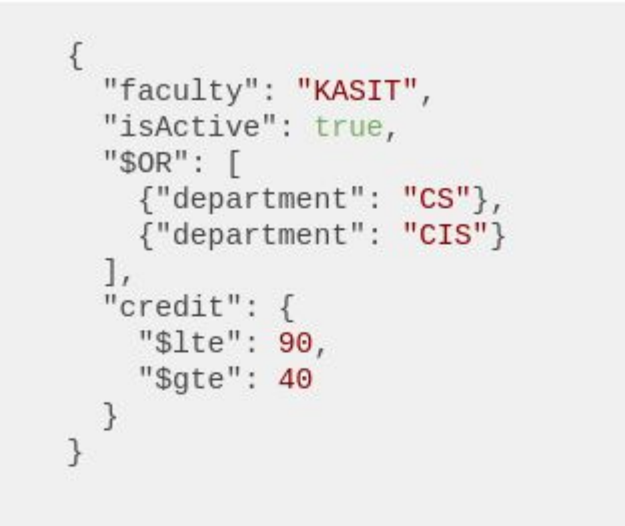
Line: 2 INT: 1 , STRING: {...data...}

Line: 3 INT: 2 , STRING: {...data...}

⋮

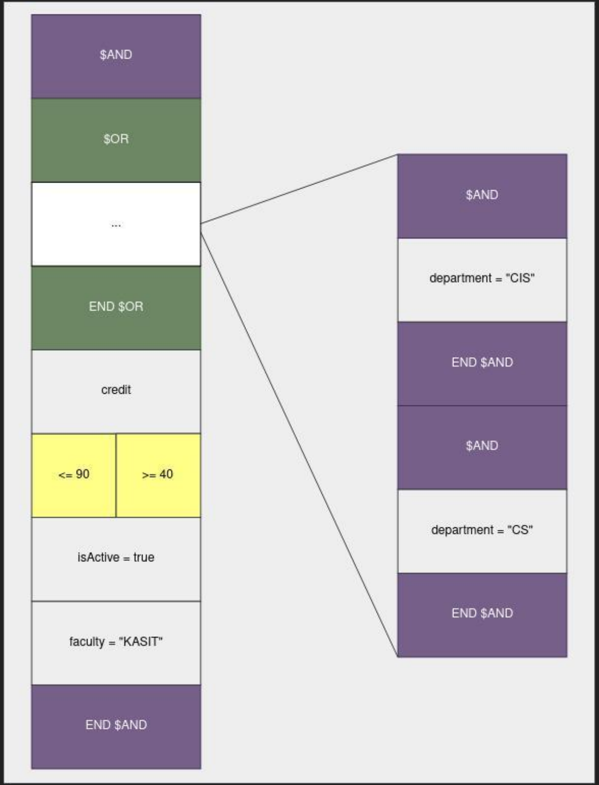
Line: N+1 INT: n , STRING: {...data...}

Parsing Query - Operations Builder



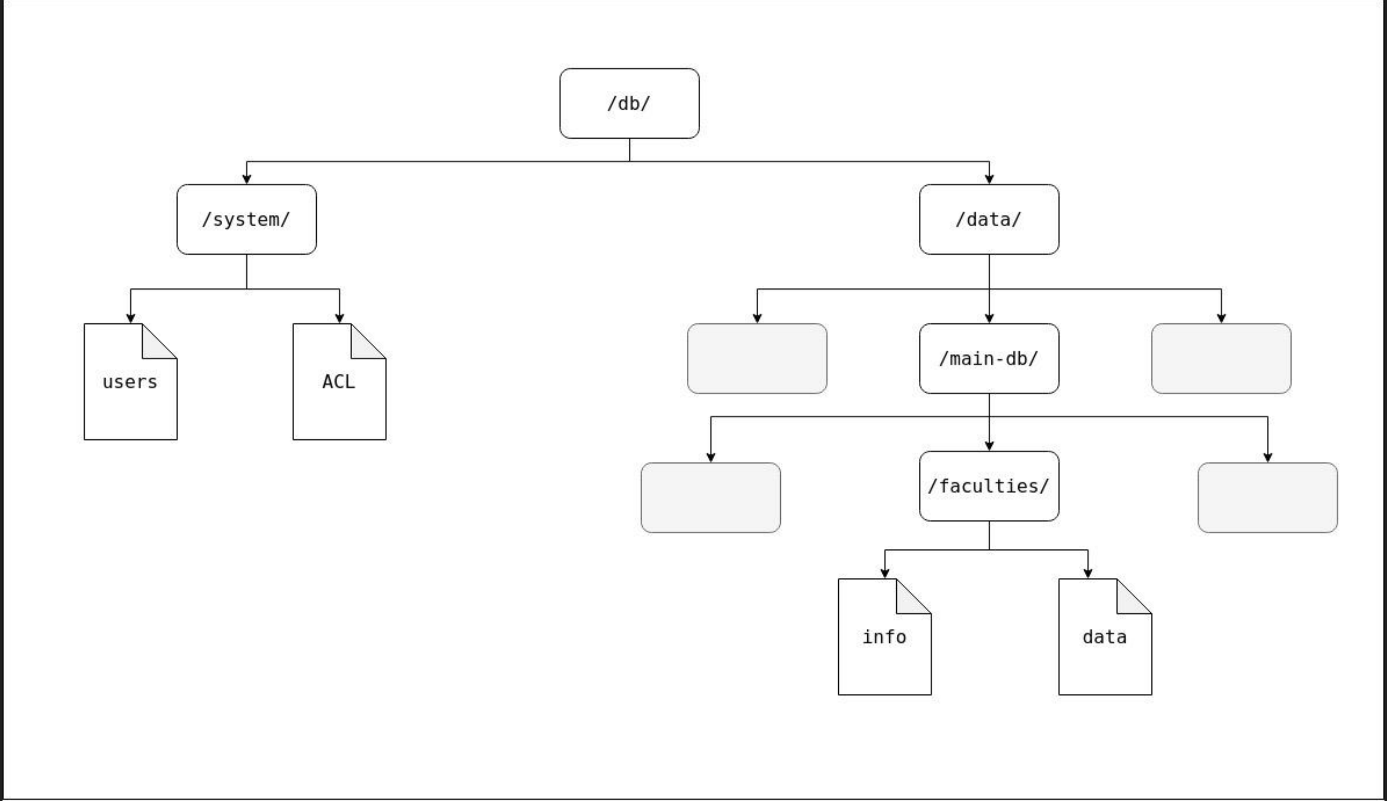
```
{
  "faculty": "KASIT",
  "isActive": true,
  "$OR": [
    {"department": "CS"},
    {"department": "CIS"}
  ],
  "credit": {
    "$lte": 90,
    "$gte": 40
  }
}
```

Raw JSON Query



Operations Stack

Database Storage - File Structure



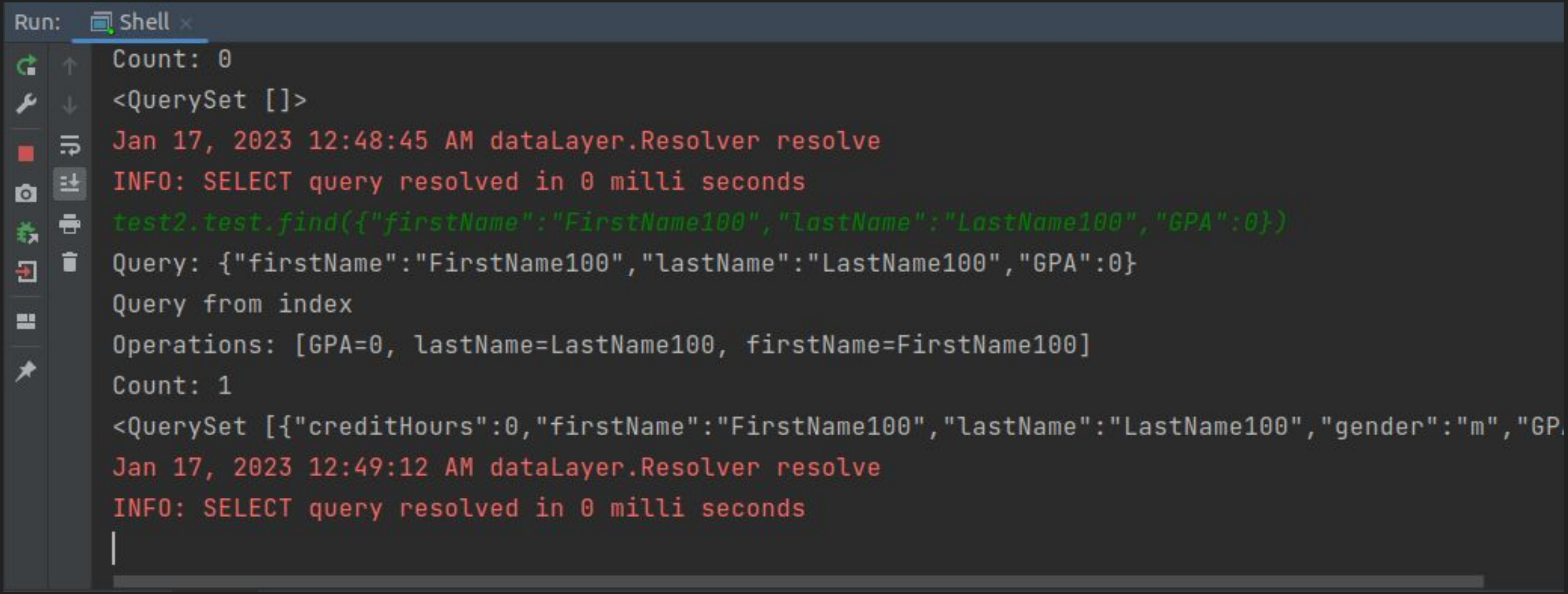
Implementation - Tools and Technologies

Database implementation using pure Java
+ Disk Management
+ In-Memory Indexing
+ Shell Client
+ Multi-Threading Programming
+ Socket Programming - Networking

Source Code Hosting
+ Version Control
+ Open-source the database

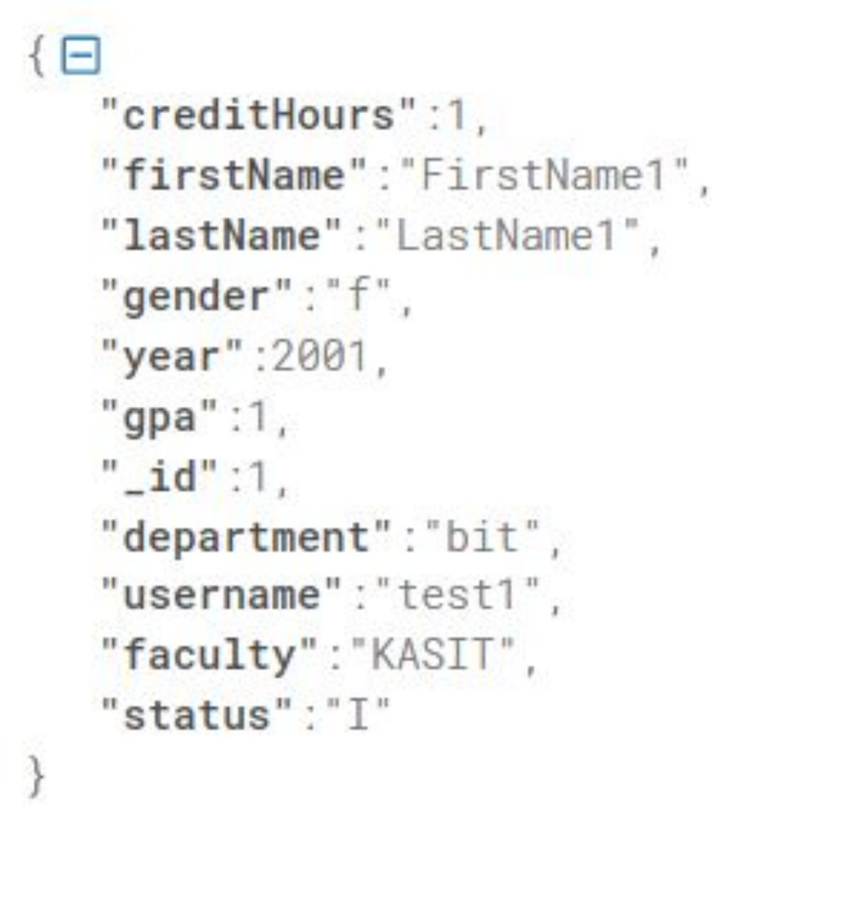
DB Demo

Students Data with ***+ 1 Million*** Records



```
Count: 0
<QuerySet []>
Jan 17, 2023 12:48:45 AM dataLayer.Resolver resolve
INFO: SELECT query resolved in 0 milli seconds
test2.test.find({"firstName":"FirstName100","lastName":"LastName100","GPA":0})
Query: {"firstName":"FirstName100","lastName":"LastName100","GPA":0}
Query from index
Operations: [GPA=0, lastName=LastName100, firstName=FirstName100]
Count: 1
<QuerySet [{"creditHours":0,"firstName":"FirstName100","lastName":"LastName100","gender":"m","GP
Jan 17, 2023 12:49:12 AM dataLayer.Resolver resolve
INFO: SELECT query resolved in 0 milli seconds
```

Sample Document



```
{⊟
    "creditHours":1,
    "firstName":"FirstName1",
    "lastName":"LastName1",
    "gender":"f",
    "year":2001,
    "gpa":1,
    "_id":1,
    "department":"bit",
    "username":"test1",
    "faculty":"KASIT",
    "status":"I"
}
```

Let Us Query Some Data

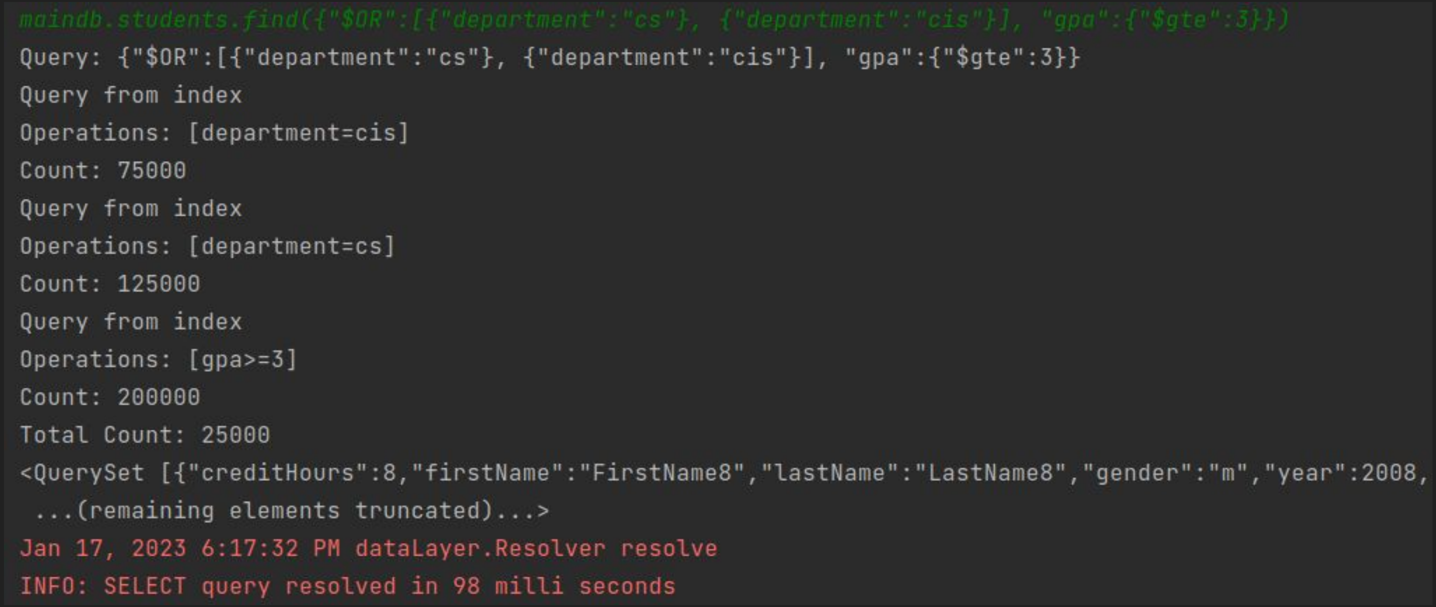
1- Get all students with ***GPA greater than 3*** <u>and</u> ***in the CS or in CIS department***.

2- Get all students that ***joined on 2010*** <u>and</u> ***gpa equals 4*** (without index)

3- Get all students that ***joined on 2010*** <u>and</u> ***gpa equals 4*** (using index)

Let Us Query Some Data

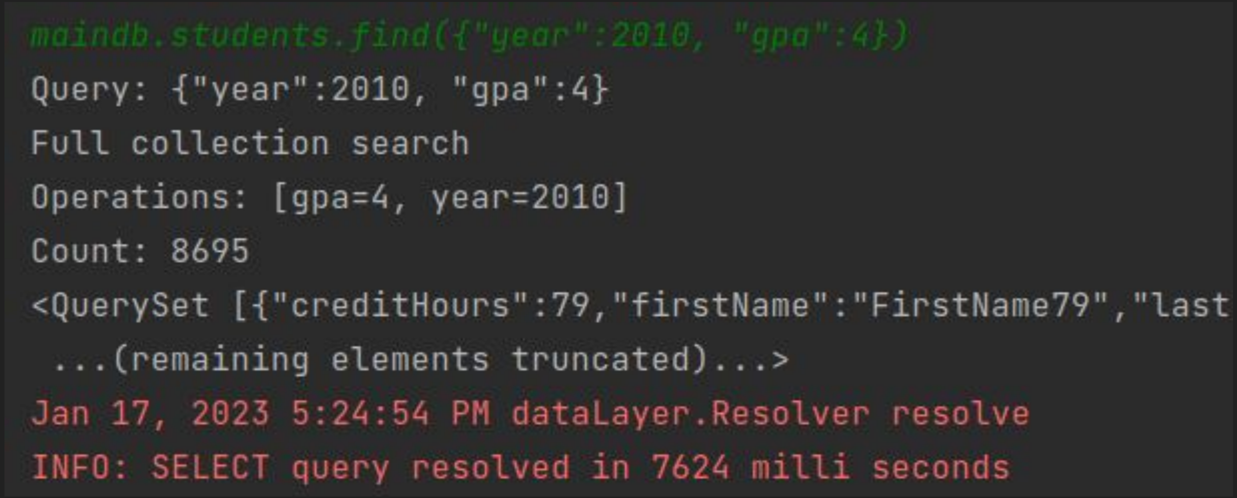
Get all students with ***GPA greater than 3*** and ***in the CS or CIS department*** (using 2 indexes: **department index** and **gpa index**)



```
maindb.students.find({"$OR":[{"department":"cs"}, {"department":"cis"}], "gpa":{"$gte":3}})
Query: {"$OR":[{"department":"cs"}, {"department":"cis"}], "gpa":{"$gte":3}}
Query from index
Operations: [department=cis]
Count: 75000
Query from index
Operations: [department=cs]
Count: 125000
Query from index
Operations: [gpa>=3]
Count: 200000
Total Count: 25000
<QuerySet [{"creditHours":8,"firstName":"FirstName8","lastName":"LastName8","gender":"m","year":2008,
 ...(remaining elements truncated)...>
Jan 17, 2023 6:17:32 PM dataLayer.Resolver resolve
INFO: SELECT query resolved in 98 milli seconds
```

Let Us Query Some Data

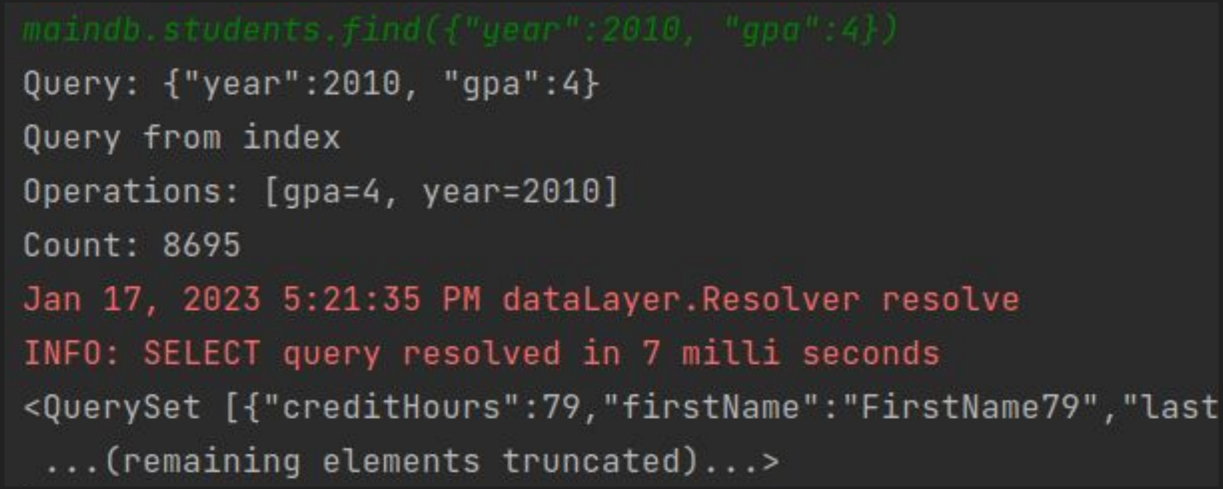
Get all students that ***joined on 2010*** and ***gpa equals 4*** (without index)



```
maindb.students.find({"year":2010, "gpa":4})
Query: {"year":2010, "gpa":4}
Full collection search
Operations: [gpa=4, year=2010]
Count: 8695
<QuerySet [{"creditHours":79,"firstName":"FirstName79","last
 ...(remaining elements truncated)...>
Jan 17, 2023 5:24:54 PM dataLayer.Resolver resolve
INFO: SELECT query resolved in 7624 milli seconds
```

Let Us Query Some Data

Get all students that ***joined on 2010*** and ***gpa equals 4*** (using index)



```
maindb.students.find({"year":2010, "gpa":4})
Query: {"year":2010, "gpa":4}
Query from index
Operations: [gpa=4, year=2010]
Count: 8695
Jan 17, 2023 5:21:35 PM dataLayer.Resolver resolve
INFO: SELECT query resolved in 7 milli seconds
<QuerySet [{"creditHours":79,"firstName":"FirstName79","last
 ...(remaining elements truncated)...>
```

Future Work and Enhancements

This database is fare from perfect, it needs some fundamental enhancements:

- Handling Concurrency Connections

- Horizontal Scaling

- Data Sharding

- On-Disk Indexing

- Building Native Drivers for Programming Languages

Questions?

Thank You