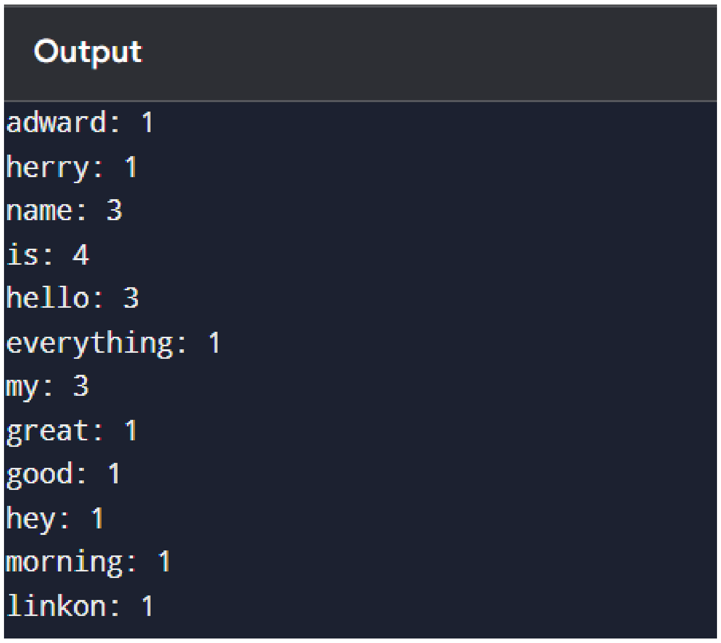
PRACTICAL 1

1) To understand the overall programming architecture using Map Reduce API.

```
import java.util.*;
import java.util.stream.Collectors;

public class MapReduce {
    public static List<KeyValue> map(String document) {
        List<KeyValue> keyValueList = new ArrayList<>();
        String[] words = document.split("\\s+");
        for (String word : words) {
            word = word.replaceAll("[^a-zA-Z]", "").toLowerCase();
            if (!word.isEmpty()) {
                keyValueList.add(new KeyValue(word, 1));
            }
        }
        return keyValueList;
    }

    public static int reduce(String key, List<Integer> values) {
        return values.stream().mapToInt(Integer::intValue).sum();
    }

    public static void main(String[] args) {
        String[] documents = {
                "Hello my name is Adward",
                "Hello my name is Herry",
                "Hello my name is Linkon",
                "Hey, good morning",
                "Everything is great"
        };

        List<KeyValue> intermediate = new ArrayList<>();
        for (String doc : documents) {
            intermediate.addAll(map(doc));
        }

        Map<String, List<Integer>> groupedByKey = intermediate.stream()
                .collect(Collectors.groupingBy(
                        KeyValue::getKey,
                        Collectors.mapping(KeyValue::getValue, Collectors.toList())));

        Map<String, Integer> wordCounts = new HashMap<>();
        for (Map.Entry<String, List<Integer>> entry : groupedByKey.entrySet()) {
            wordCounts.put(entry.getKey(), reduce(entry.getKey(), entry.getValue()));
        }
        wordCounts.forEach((word, count) -> System.out.println(word + ": " + count));
    }
}

class KeyValue {
    private String key;
    private int value;

    public KeyValue(String key, int value) {
        this.key = key;
        this.value = value;
    }

    public String getKey() {
        return key;
    }

    public int getValue() {
        return value;
    }
}
```

Output:



```
adward: 1
herry: 1
name: 3
is: 4
hello: 3
everything: 1
my: 3
great: 1
good: 1
hey: 1
morning: 1
linkon: 1
```

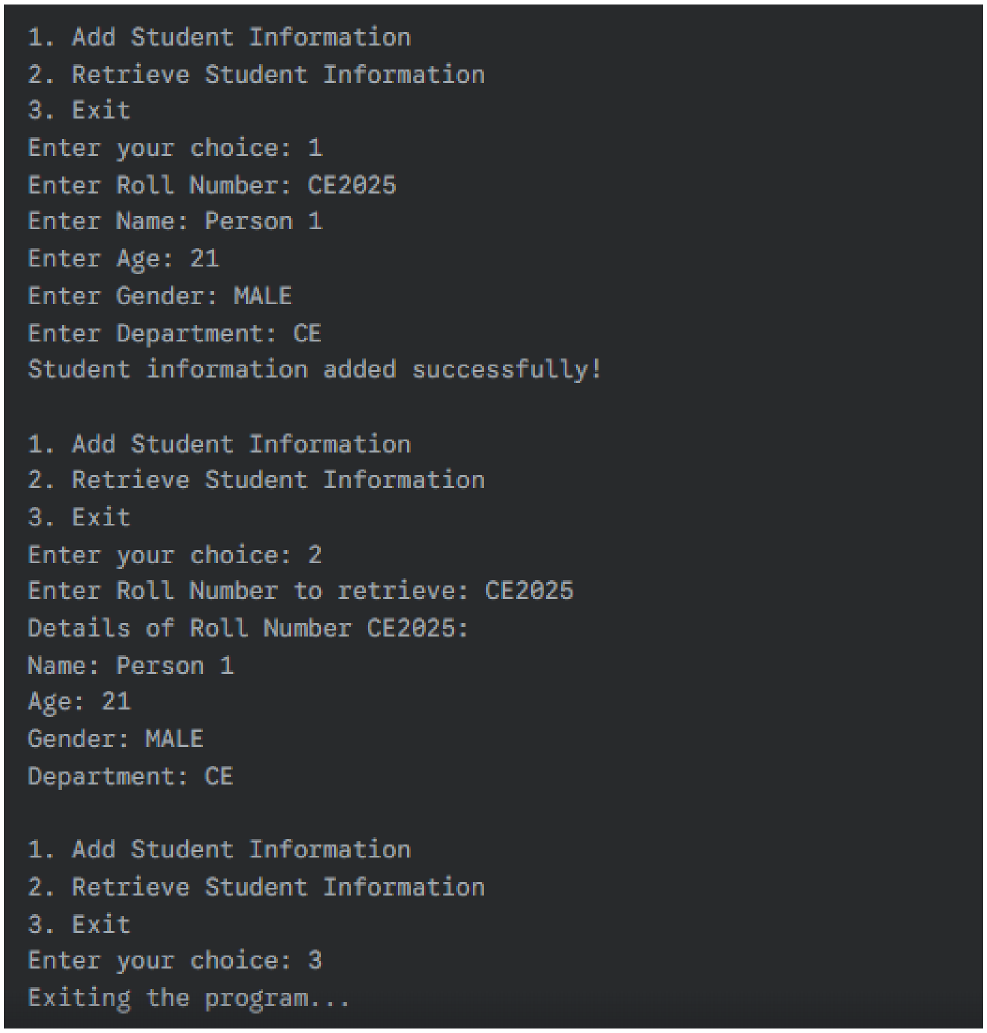
PRACTICAL 2

2) Store the basic information about students such as roll no, name, date of birth, and address of student using various collection types such as List, Set and Map.

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

class Student {
    private String name;
    private int age;
    private String gender;
    private String department;

    public Student(String name, int age, String gender, String department) {
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.department = department;
    }

    @Override
    public String toString() {
        return "Name: " + name +
                "\nAge: " + age +
                "\nGender: " + gender +
                "\nDepartment: " + department;
    }
}

public class StudentInfo {
    private static Map<String, Student> students = new HashMap<>();

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            displayMenu();
            int choice = Integer.parseInt(scanner.nextLine());
            switch (choice) {
                case 1:
                    addStudent(scanner);
                    break;
                case 2:
                    retrieveStudent(scanner);
                    break;
                case 3:
```

```
                System.out.println("Exiting the program...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice! Please try again.\n");
        }
    }
}

private static void displayMenu() {
    System.out.println("1. Add Student Information");
    System.out.println("2. Retrieve Student Information");
    System.out.println("3. Exit");
    System.out.print("Enter your choice: ");
}

private static void addStudent(Scanner scanner) {
    System.out.print("Enter Roll Number: ");
    String rollNumber = scanner.nextLine();
    System.out.print("Enter Name: ");
    String name = scanner.nextLine();
    System.out.print("Enter Age: ");
    int age = Integer.parseInt(scanner.nextLine());
    System.out.print("Enter Gender: ");
    String gender = scanner.nextLine();
    System.out.print("Enter Department: ");
    String department = scanner.nextLine();

    Student student = new Student(name, age, gender, department);
    students.put(rollNumber, student);
    System.out.println("Student information added successfully!\n");
}

private static void retrieveStudent(Scanner scanner) {
    System.out.print("Enter Roll Number to retrieve: ");
    String rollNumber = scanner.nextLine();
    Student student = students.get(rollNumber);
    if (student != null) {
        System.out.println("Details of Roll Number " + rollNumber + ":");
        System.out.println(student);
    } else {
        System.out.println("Student not found!\n");
    }
}
}
```

Output:

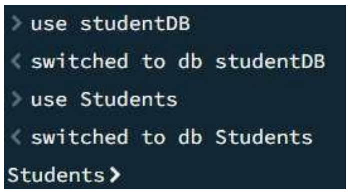


```
1. Add Student Information
2. Retrieve Student Information
3. Exit
Enter your choice: 1
Enter Roll Number: CE2025
Enter Name: Person 1
Enter Age: 21
Enter Gender: MALE
Enter Department: CE
Student information added successfully!

1. Add Student Information
2. Retrieve Student Information
3. Exit
Enter your choice: 2
Enter Roll Number to retrieve: CE2025
Details of Roll Number CE2025:
Name: Person 1
Age: 21
Gender: MALE
Department: CE

1. Add Student Information
2. Retrieve Student Information
3. Exit
Enter your choice: 3
Exiting the program...
```

PRACTICAL 3

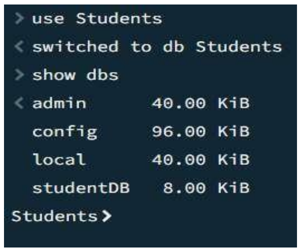
3) Basic CRUD operations in MongoDB.

i) Create studentDB database:



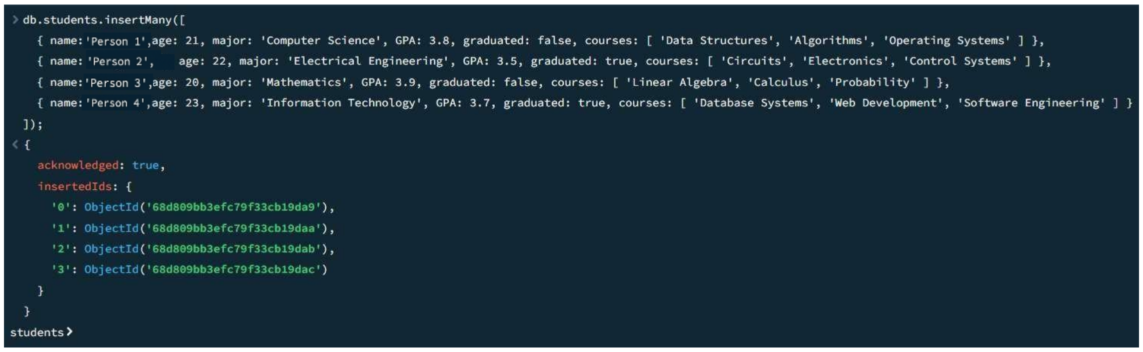
```
> use studentDB
< switched to db studentDB
> use Students
< switched to db Students
Students>
```

ii) Show databases:



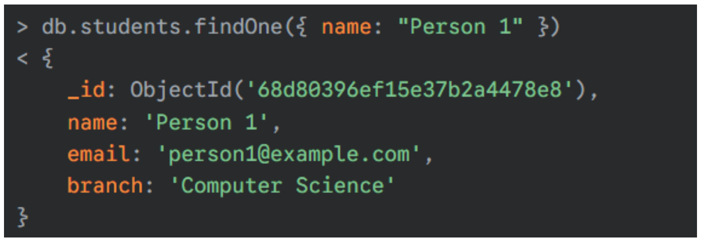
```
> use Students
< switched to db Students
> show dbs
< admin       40.00 KiB
  config      96.00 KiB
  local       40.00 KiB
  studentDB    8.00 KiB
Students>
```

iii) Insertmany() in studentDB:



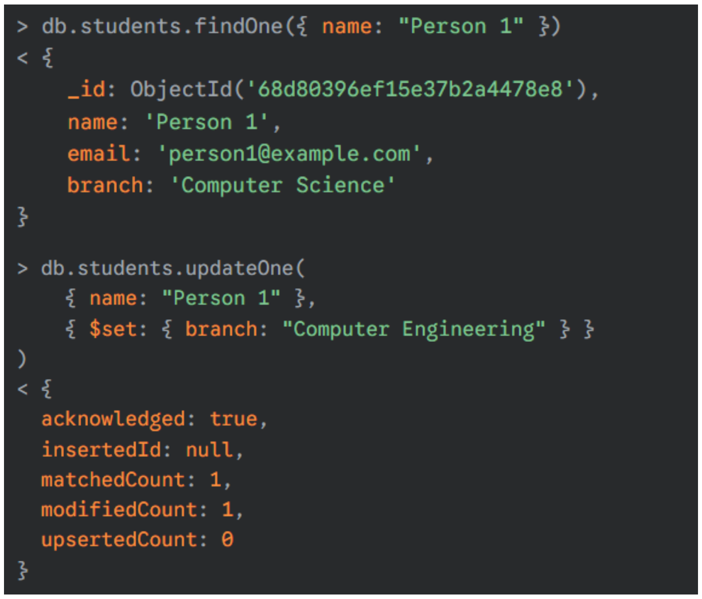
```
> db.students.insertMany([
    { name:'Person 1',age: 21, major: 'Computer Science', GPA: 3.8, graduated: false, courses: [ 'Data Structures', 'Algorithms', 'Operating Systems' ] },
    { name:'Person 2',     age: 22, major: 'Electrical Engineering', GPA: 3.5, graduated: true, courses: [ 'Circuits', 'Electronics', 'Control Systems' ] },
    { name:'Person 3',age: 20, major: 'Mathematics', GPA: 3.9, graduated: false, courses: [ 'Linear Algebra', 'Calculus', 'Probability' ] },
    { name:'Person 4',age: 23, major: 'Information Technology', GPA: 3.7, graduated: true, courses: [ 'Database Systems', 'Web Development', 'Software Engineering' ] }
  ]);
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('68d809bb3efc79f33cb19da9'),
      '1': ObjectId('68d809bb3efc79f33cb19daa'),
      '2': ObjectId('68d809bb3efc79f33cb19dab'),
      '3': ObjectId('68d809bb3efc79f33cb19dac')
    }
  }
students>
```

iv) findOne():



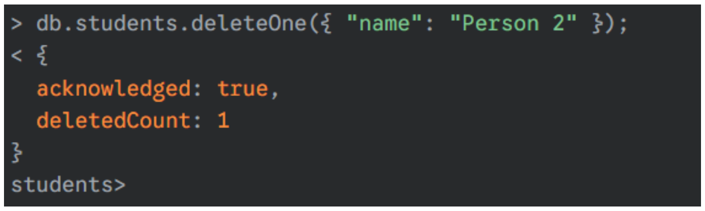
```
> db.students.findOne({ name: "Person 1" })
< {
    _id: ObjectId('68d80396ef15e37b2a4478e8'),
    name: 'Person 1',
    email: 'person1@example.com',
    branch: 'Computer Science'
  }
```

v) **UpdateOne():**



```
> db.students.findOne({ name: "Person 1" })
< {
    _id: ObjectId('68d80396ef15e37b2a4478e8'),
    name: 'Person 1',
    email: 'person1@example.com',
    branch: 'Computer Science'
}

> db.students.updateOne(
    { name: "Person 1" },
    { $set: { branch: "Computer Engineering" } }
)
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

vi) **deleteOne():**

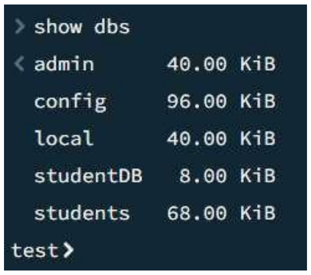


```
> db.students.deleteOne({ "name": "Person 2" });
< {
  acknowledged: true,
  deletedCount: 1
}
students>
```

PRACTICAL 4

4) Retrieve various types of documents from students collection.

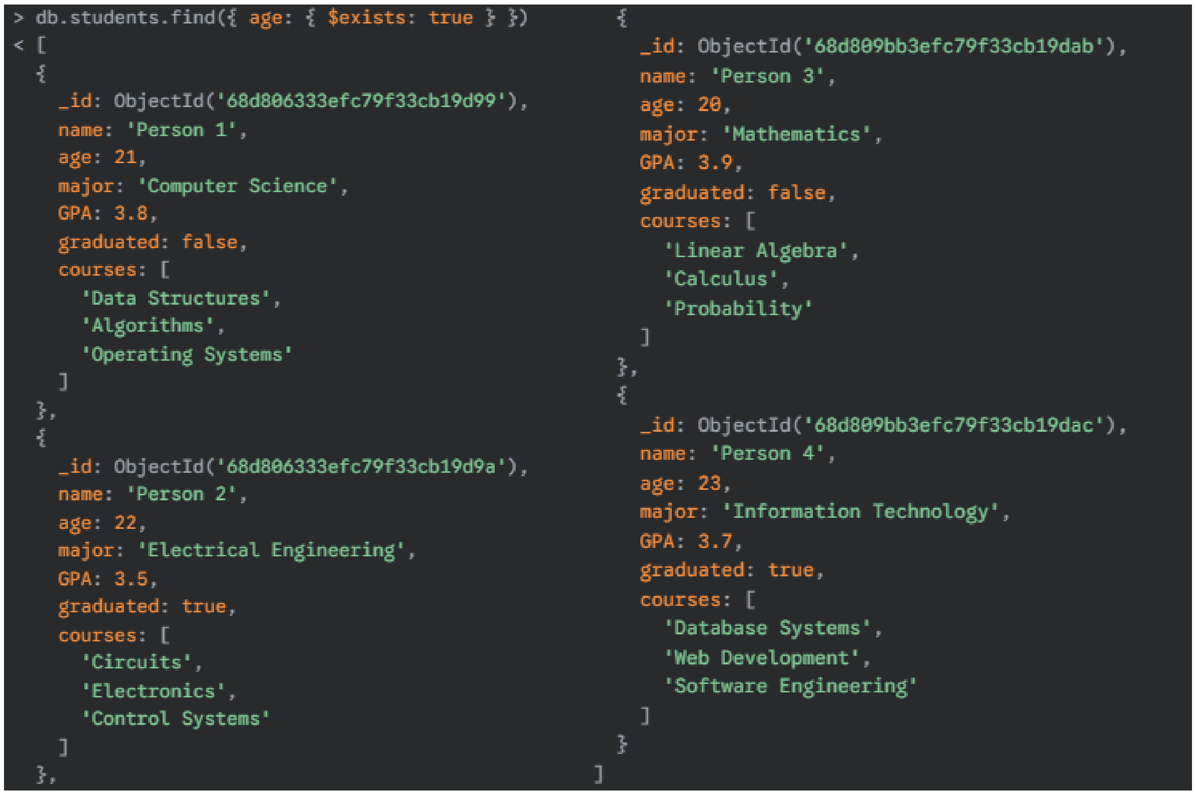
Show Database and use Database:



```
> show dbs
< admin       40.00 KiB
  config      96.00 KiB
  local       40.00 KiB
  studentDB    8.00 KiB
  students    68.00 KiB
test>
```

i) **Simple Documents:**

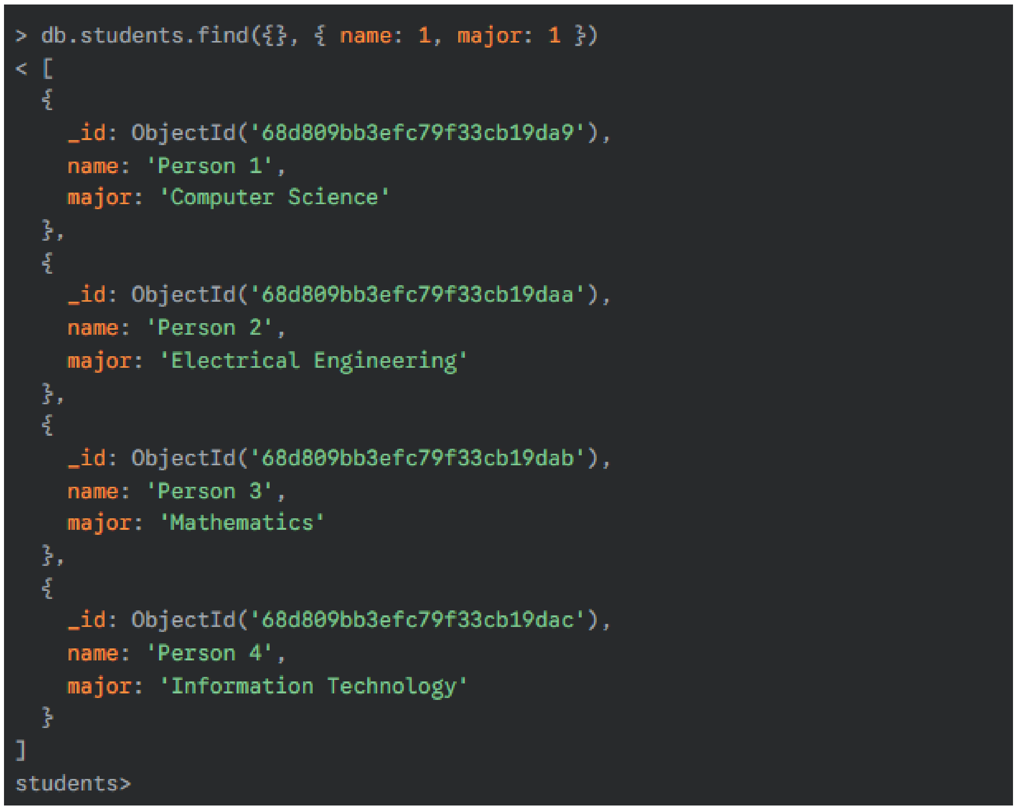
- Documents with straightforward fields and values.



```
> db.students.find({ age: { $exists: true } })
< [
  {
    _id: ObjectId('68d806333efc79f33cb19d99'),
    name: 'Person 1',
    age: 21,
    major: 'Computer Science',
    GPA: 3.8,
    graduated: false,
    courses: [
      'Data Structures',
      'Algorithms',
      'Operating Systems'
    ]
  },
  {
    _id: ObjectId('68d806333efc79f33cb19d9a'),
    name: 'Person 2',
    age: 22,
    major: 'Electrical Engineering',
    GPA: 3.5,
    graduated: true,
    courses: [
      'Circuits',
      'Electronics',
      'Control Systems'
    ]
  },
  {
    _id: ObjectId('68d809bb3efc79f33cb19dab'),
    name: 'Person 3',
    age: 20,
    major: 'Mathematics',
    GPA: 3.9,
    graduated: false,
    courses: [
      'Linear Algebra',
      'Calculus',
      'Probability'
    ]
  },
  {
    _id: ObjectId('68d809bb3efc79f33cb19dac'),
    name: 'Person 4',
    age: 23,
    major: 'Information Technology',
    GPA: 3.7,
    graduated: true,
    courses: [
      'Database Systems',
      'Web Development',
      'Software Engineering'
    ]
  }
]
```

ii) **Documents with Specific Fields:**

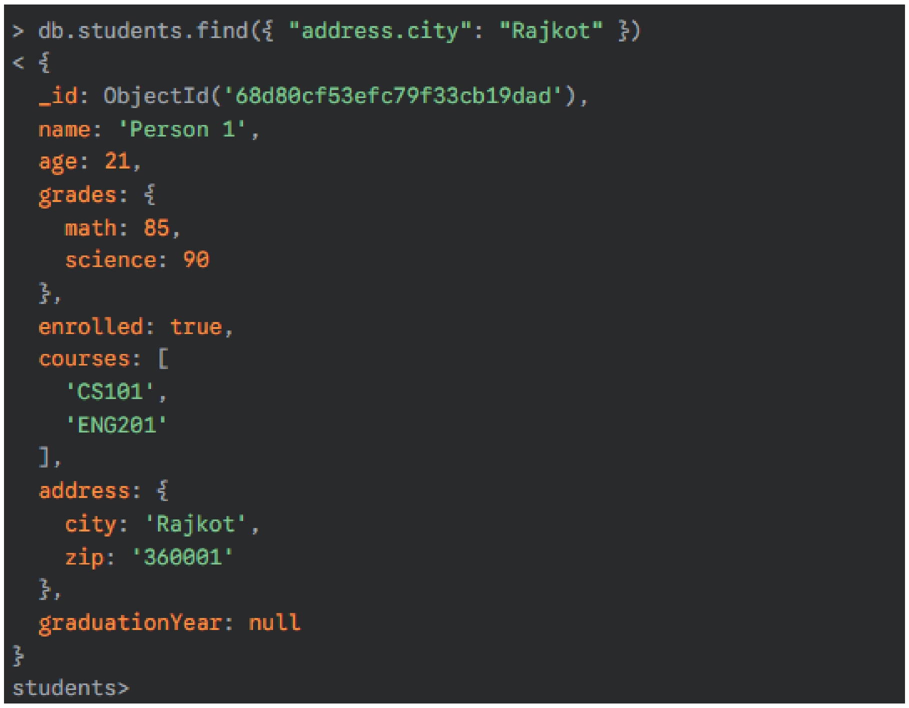
- Retrieving documents that contain specific fields or exclude certain fields.



```
> db.students.find({}, { name: 1, major: 1 })
< [
  {
    _id: ObjectId('68d809bb3efc79f33cb19da9'),
    name: 'Person 1',
    major: 'Computer Science'
  },
  {
    _id: ObjectId('68d809bb3efc79f33cb19daa'),
    name: 'Person 2',
    major: 'Electrical Engineering'
  },
  {
    _id: ObjectId('68d809bb3efc79f33cb19dab'),
    name: 'Person 3',
    major: 'Mathematics'
  },
  {
    _id: ObjectId('68d809bb3efc79f33cb19dac'),
    name: 'Person 4',
    major: 'Information Technology'
  }
]
students>
```

iii) **Nested Documents:**

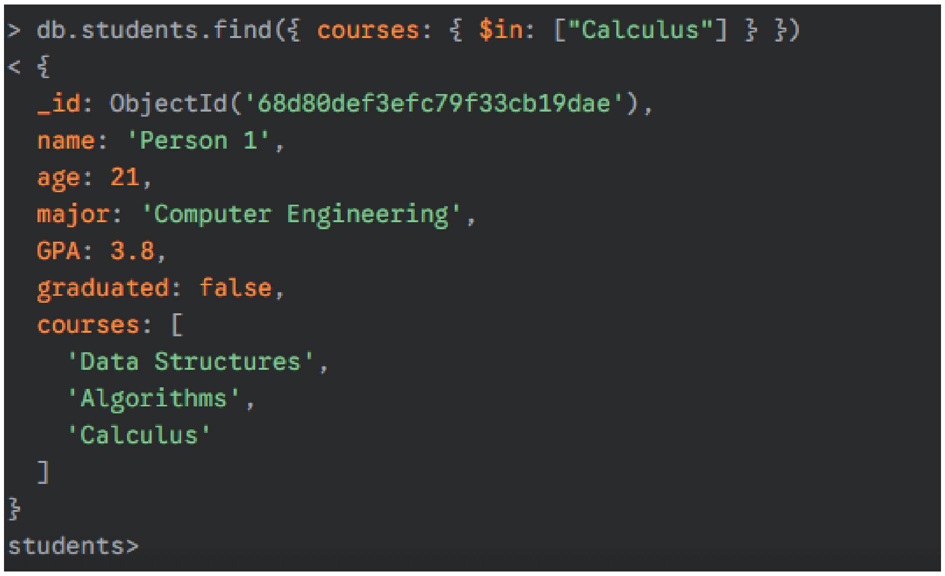
- Documents containing nested structures (documents within documents).



```
> db.students.find({ "address.city": "Rajkot" })
< {
  _id: ObjectId('68d80cf53efc79f33cb19dad'),
  name: 'Person 1',
  age: 21,
  grades: {
    math: 85,
    science: 90
  },
  enrolled: true,
  courses: [
    'CS101',
    'ENG201'
  ],
  address: {
    city: 'Rajkot',
    zip: '360001'
  },
  graduationYear: null
}
students>
```

iv) **Documents with Arrays:**

- Documents that include arrays (lists of values).

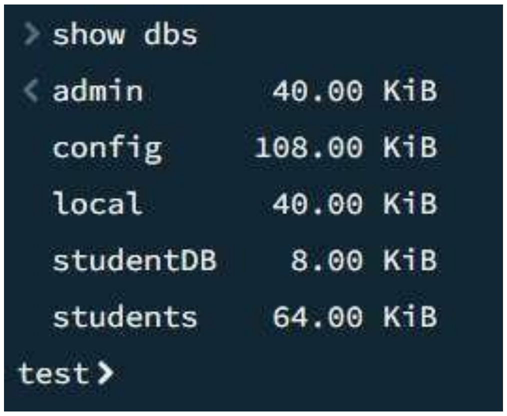


```
> db.students.find({ courses: { $in: ["Calculus"] } })
< {
  _id: ObjectId('68d80def3efc79f33cb19dae'),
  name: 'Person 1',
  age: 21,
  major: 'Computer Engineering',
  GPA: 3.8,
  graduated: false,
  courses: [
    'Data Structures',
    'Algorithms',
    'Calculus'
  ]
}
students>
```

PRACTICAL 5

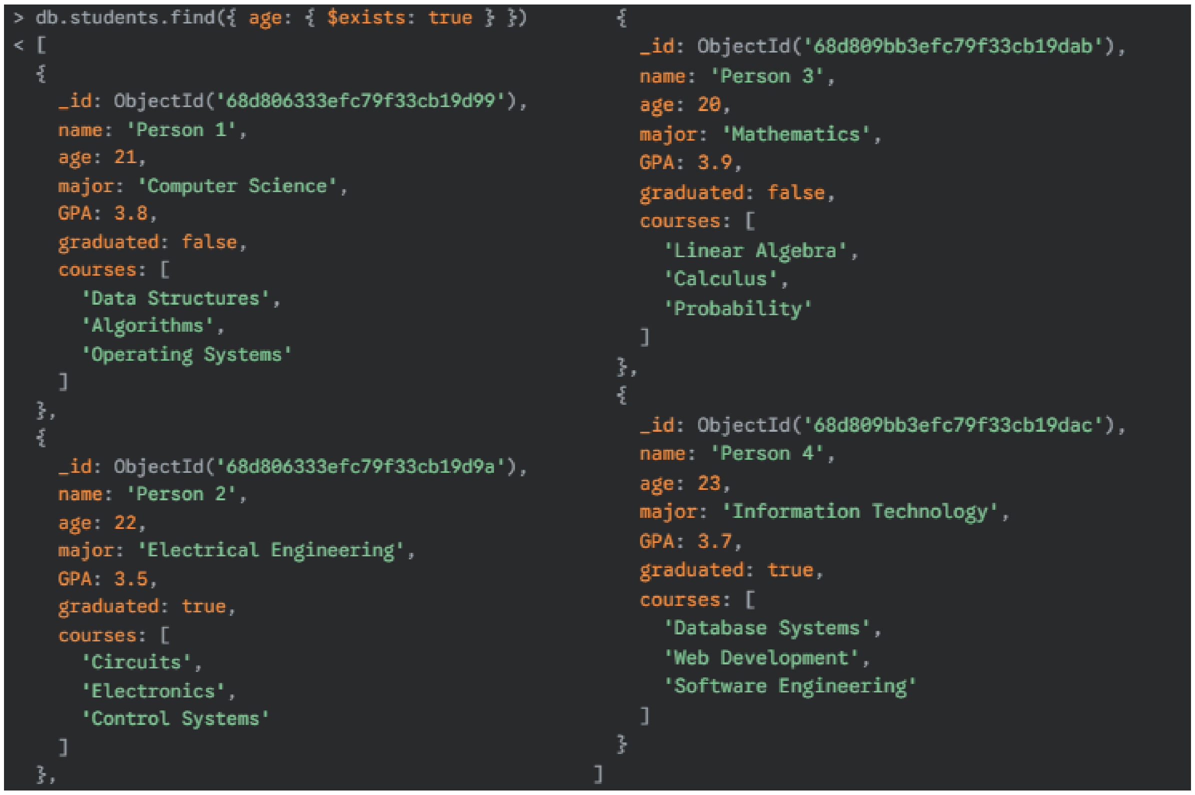
5) To find documents from Students collection.

i) Show Database and use Database:



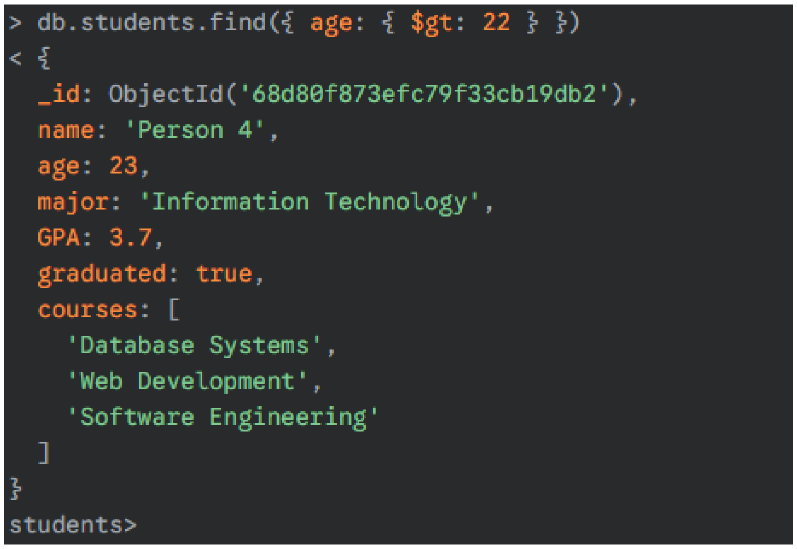
```
> show dbs
< admin        40.00 KiB
  config      108.00 KiB
  local        40.00 KiB
  studentDB     8.00 KiB
  students     64.00 KiB
test>
```

ii) Find() all Data:



```
> db.students.find({ age: { $exists: true } })
< [
  {
    _id: ObjectId('68d806333efc79f33cb19d99'),
    name: 'Person 1',
    age: 21,
    major: 'Computer Science',
    GPA: 3.8,
    graduated: false,
    courses: [
      'Data Structures',
      'Algorithms',
      'Operating Systems'
    ]
  },
  {
    _id: ObjectId('68d806333efc79f33cb19d9a'),
    name: 'Person 2',
    age: 22,
    major: 'Electrical Engineering',
    GPA: 3.5,
    graduated: true,
    courses: [
      'Circuits',
      'Electronics',
      'Control Systems'
    ]
  },
  {
    _id: ObjectId('68d809bb3efc79f33cb19dab'),
    name: 'Person 3',
    age: 20,
    major: 'Mathematics',
    GPA: 3.9,
    graduated: false,
    courses: [
      'Linear Algebra',
      'Calculus',
      'Probability'
    ]
  },
  {
    _id: ObjectId('68d809bb3efc79f33cb19dac'),
    name: 'Person 4',
    age: 23,
    major: 'Information Technology',
    GPA: 3.7,
    graduated: true,
    courses: [
      'Database Systems',
      'Web Development',
      'Software Engineering'
    ]
  }
]
```

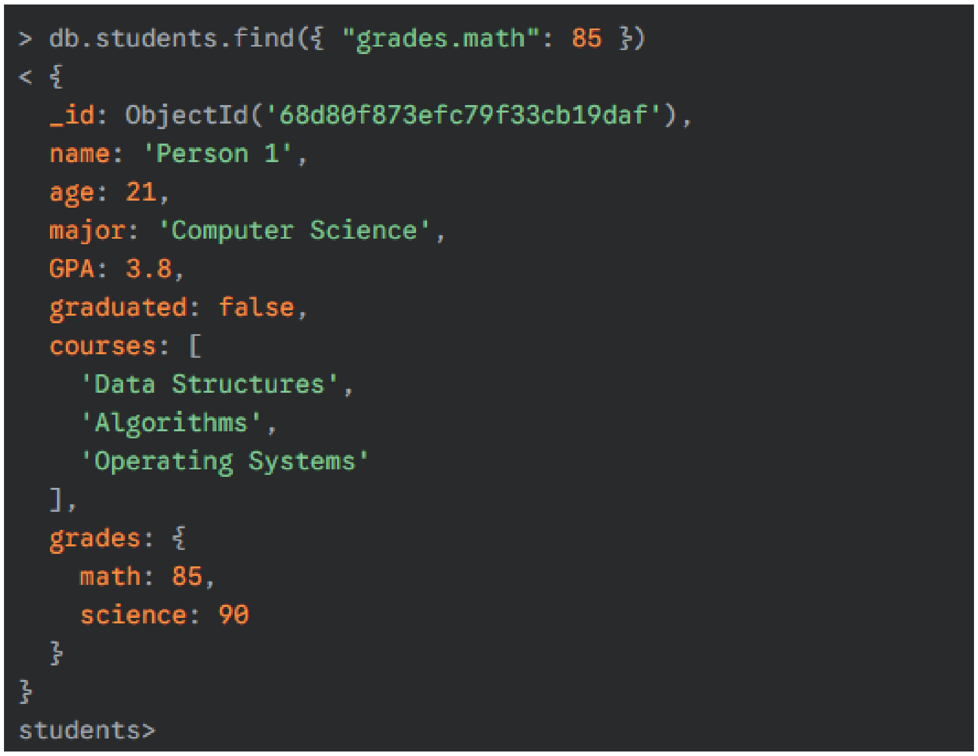
iii) Find Students with Age Greater Than 22:



```
> db.students.find({ age: { $gt: 22 } })
< {
  _id: ObjectId('68d80f873efc79f33cb19db2'),
  name: 'Person 4',
  age: 23,
  major: 'Information Technology',
  GPA: 3.7,
  graduated: true,
  courses: [
    'Database Systems',
    'Web Development',
    'Software Engineering'
  ]
}
students>
```

iv) Find Students with a Specific Grade in Math:

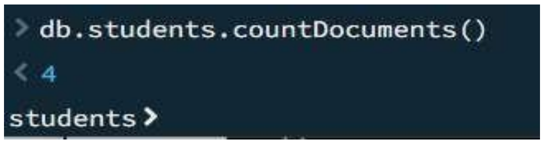
- This query retrieves students who have a grade of 85 in math.



```
> db.students.find({ "grades.math": 85 })
< {
  _id: ObjectId('68d80f873efc79f33cb19daf'),
  name: 'Person 1',
  age: 21,
  major: 'Computer Science',
  GPA: 3.8,
  graduated: false,
  courses: [
    'Data Structures',
    'Algorithms',
    'Operating Systems'
  ],
  grades: {
    math: 85,
    science: 90
  }
}
students>
```

v) Count the Number of Students:

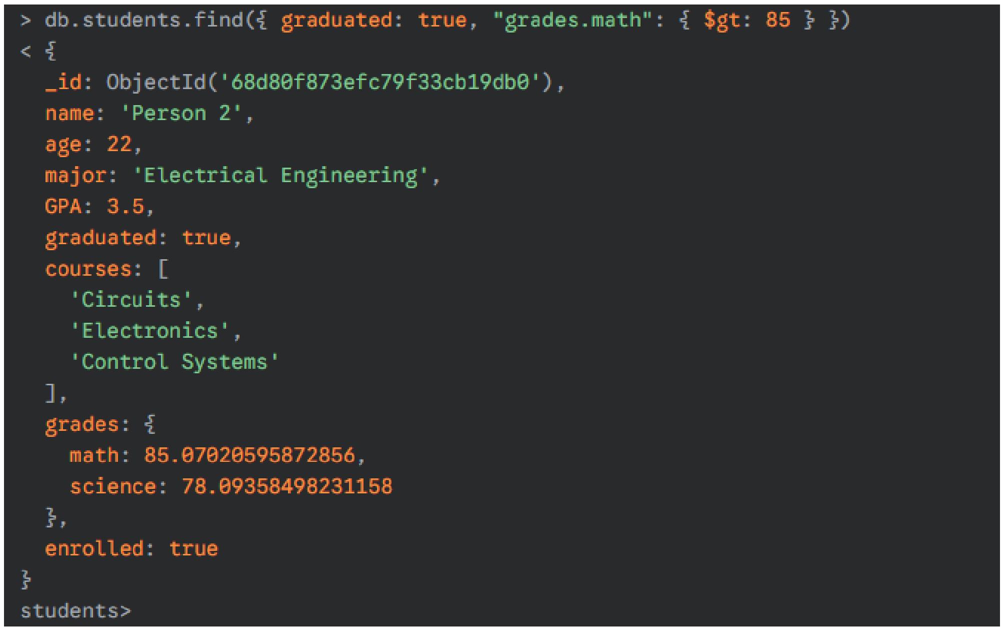
- This query counts the total number of documents (students) in the collection.



```
> db.students.countDocuments()
< 4
students>
```

vi) Find Students with Multiple Conditions:

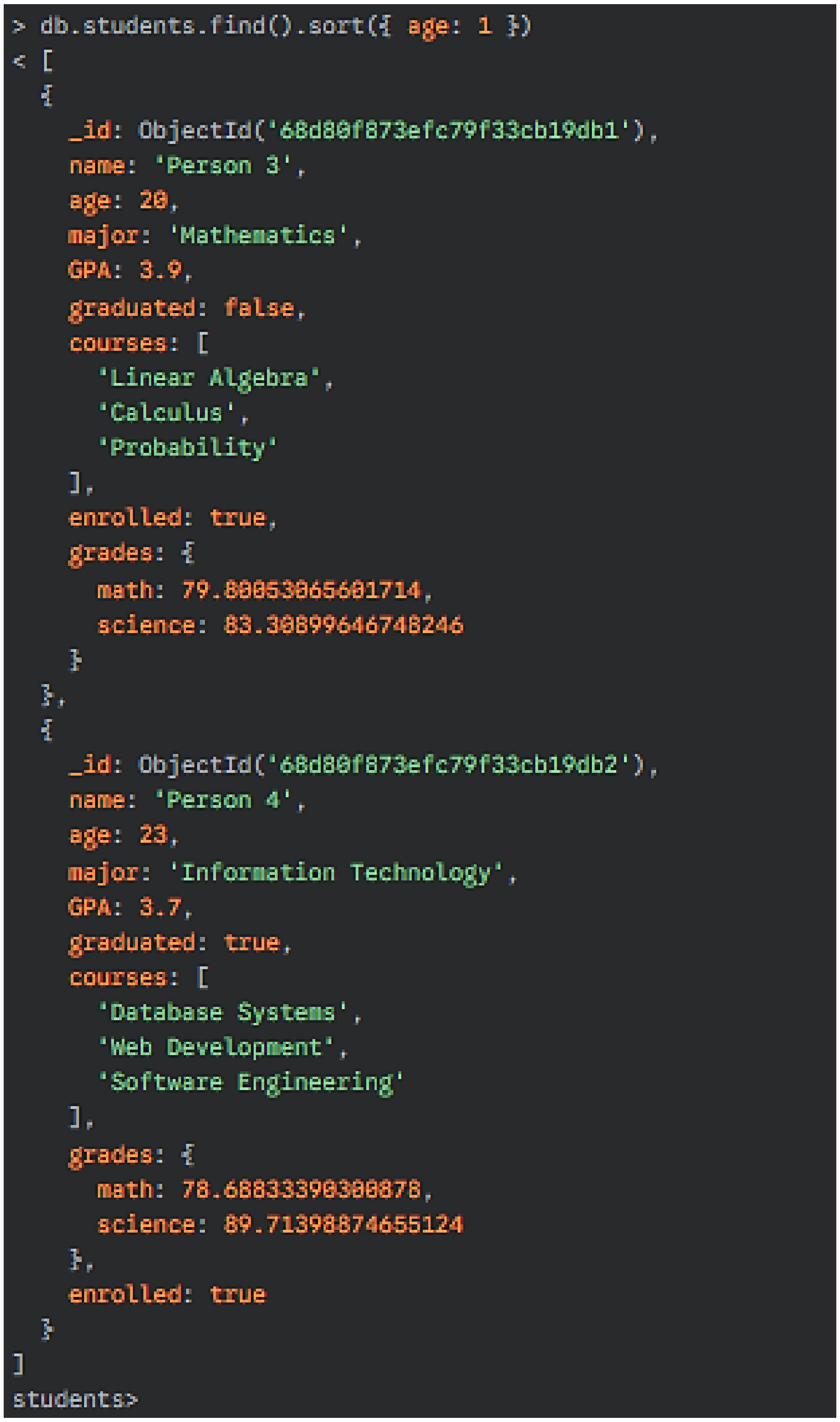
- This query retrieves students who are enrolled and have a science grade > 80.



```
> db.students.find({ graduated: true, "grades.math": { $gt: 85 } })
< {
  _id: ObjectId('68d80f873efc79f33cb19db0'),
  name: 'Person 2',
  age: 22,
  major: 'Electrical Engineering',
  GPA: 3.5,
  graduated: true,
  courses: [
    'Circuits',
    'Electronics',
    'Control Systems'
  ],
  grades: {
    math: 85.07020595872856,
    science: 78.09358498231158
  },
  enrolled: true
}
students>
```

vii) Sort Students by Age:

- This query retrieves all students and sorts them by age in ascending order.



```
> db.students.find().sort({ age: 1 })
< [
  {
    _id: ObjectId('68d80f873efc79f33cb19db1'),
    name: 'Person 3',
    age: 20,
    major: 'Mathematics',
    GPA: 3.9,
    graduated: false,
    courses: [
      'Linear Algebra',
      'Calculus',
      'Probability'
    ],
    enrolled: true,
    grades: {
      math: 79.88853865601714,
      science: 83.30899646748246
    }
  },
  {
    _id: ObjectId('68d80f873efc79f33cb19db2'),
    name: 'Person 4',
    age: 23,
    major: 'Information Technology',
    GPA: 3.7,
    graduated: true,
    courses: [
      'Database Systems',
      'Web Development',
      'Software Engineering'
    ],
    grades: {
      math: 78.68833390300878,
      science: 89.71398874655124
    },
    enrolled: true
  }
]
students>
```

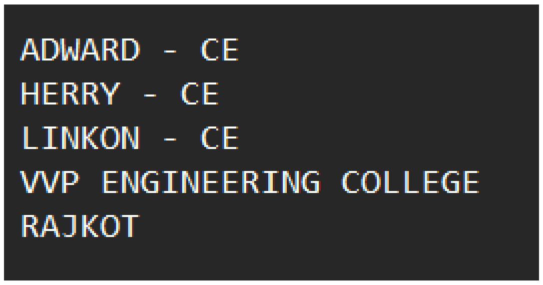
PRACTICAL 6

6) Develop Map Reduce Work Application.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class WordCount {
  public static void main(String[] args) {
    if (args.length < 1) {
      System.out.println("Please provide the file path as an argument.");
      return;
    }

    String filePath = args[0];
    Map<String, Integer> wordCountMap = new HashMap<>();

    try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
      String line;
      while ((line = reader.readLine()) != null) {
        String[] words = line.split("\\s+");
        for (String word : words) {
          word = word.toLowerCase().replaceAll("[^a-zA-Z]", "");
          if (word.isEmpty()) continue;
          wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);
        }
      }
    } catch (IOException e) {
      System.err.println("Error reading the file: " + e.getMessage());
    }

    for (Map.Entry<String, Integer> entry : wordCountMap.entrySet()) {
      System.out.println(entry.getKey() + ": " + entry.getValue());
    }
  }
}
```

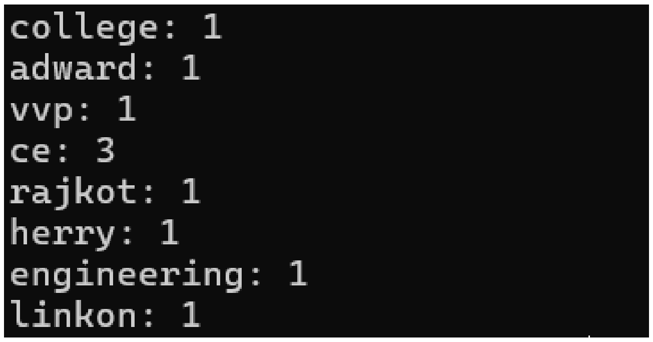
Output:

WordCount.txt:



```
ADWARD - CE
HERRY - CE
LINKON - CE
VVP ENGINEERING COLLEGE
RAJKOT
```

Output Image:

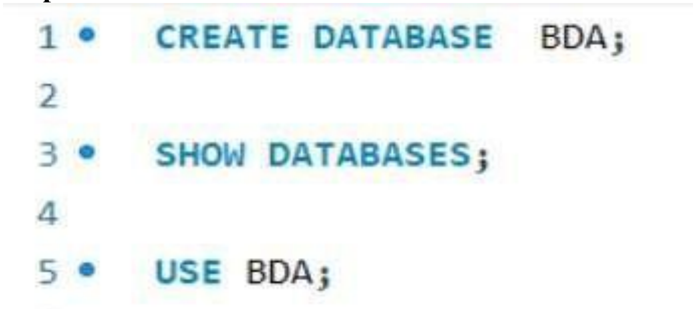


```
college: 1
adward: 1
vvp: 1
ce: 3
rajkot: 1
herry: 1
engineering: 1
linkon: 1
```

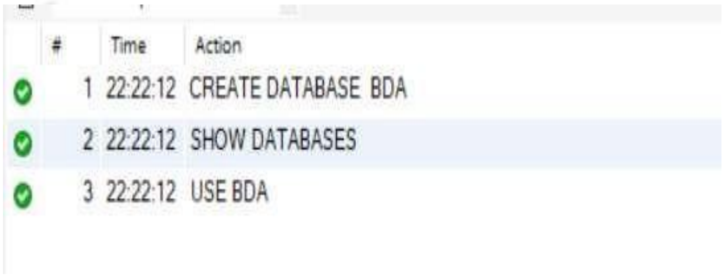
PRACTICAL 7

7) Creating the HDFS tables and loading them in Hive and learn joining of tables in Hive.

Step 1: Create Database & USE Database:



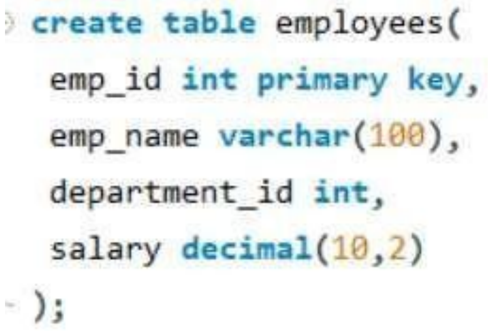
```
CREATE DATABASE  BDA;

SHOW DATABASES;

USE BDA;
```



```
   #  Time      Action
   1  22:22:12  CREATE DATABASE  BDA
   2  22:22:12  SHOW DATABASES
   3  22:22:12  USE BDA
```

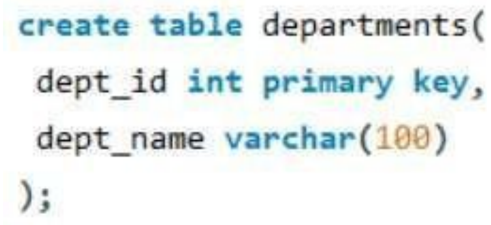
Step 2: Create Tables:

Create employees table:

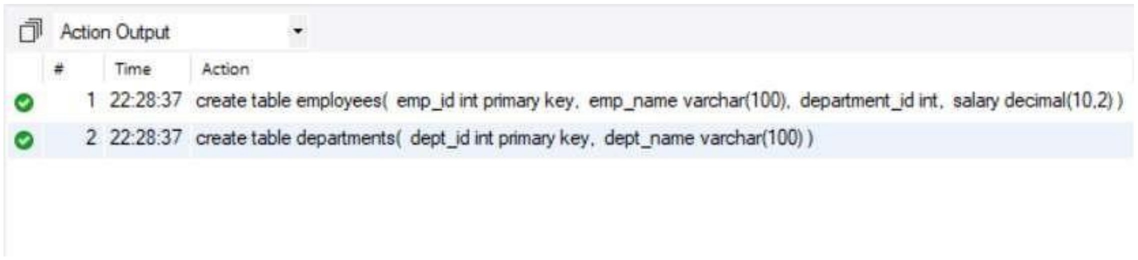


```
create table employees(
  emp_id int primary key,
  emp_name varchar(100),
  department_id int,
  salary decimal(10,2)
);
```

Create departments table:



```
create table departments(
  dept_id int primary key,
  dept_name varchar(100)
);
```



Action Output

#	Time	Action
1	22:28:37	create table employees(emp_id int primary key, emp_name varchar(100), department_id int, salary decimal(10,2))
2	22:28:37	create table departments(dept_id int primary key, dept_name varchar(100))

Step 3: Insert Data into Tables:

i) Insert data into employees table:

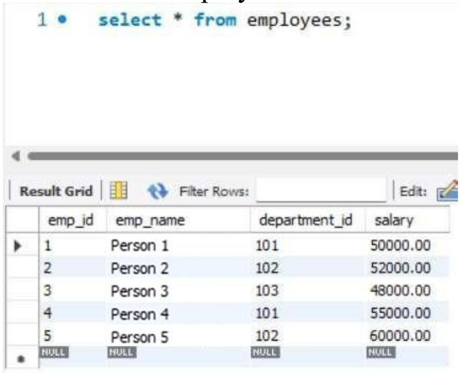
```
INSERT INTO employees (emp_id,
  emp_name, department_id, salary)
  VALUES
  (1, 'Person 1', 101, 50000.00),
  (2, 'Person 2', 102, 52000.00),
  (3, 'Person 3', 103, 48000.00),
  (4, 'Person 4', 101, 55000.00),
  (5, 'Person 5', 102, 60000.00);
```

ii) Insert data into departments table:

```
INSERT INTO departments (dept_id, dept_name) VALUES
(101, 'HR'),
(102, 'Engineering'),
(103, 'Marketing'),
(104, 'Finance');
```

Step 4:

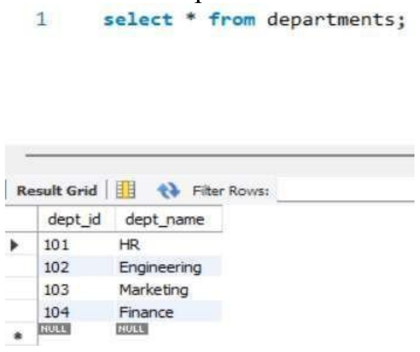
i) select * from employees:



```
select * from employees;
```

emp_id	emp_name	department_id	salary
1	Person 1	101	50000.00
2	Person 2	102	52000.00
3	Person 3	103	48000.00
4	Person 4	101	55000.00
5	Person 5	102	60000.00
NULL	NULL	NULL	NULL

ii) select * from departments:



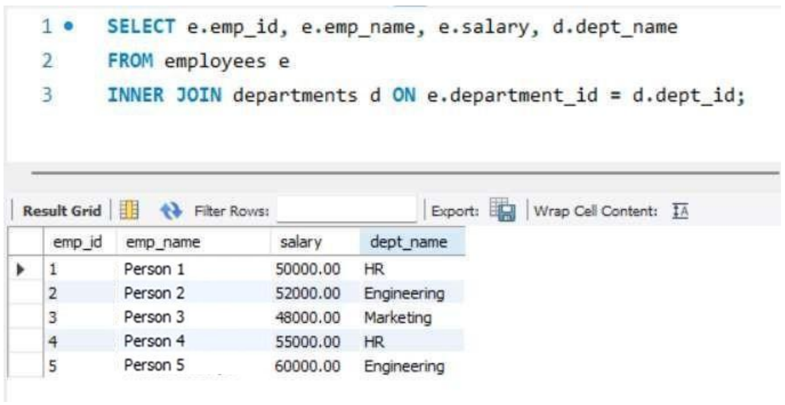
```
select * from departments;
```

dept_id	dept_name
101	HR
102	Engineering
103	Marketing
104	Finance
NULL	NULL

Step 5: Performing Joins:

i) Inner Join:

- Retrieves records where there is a match between employees and departments.



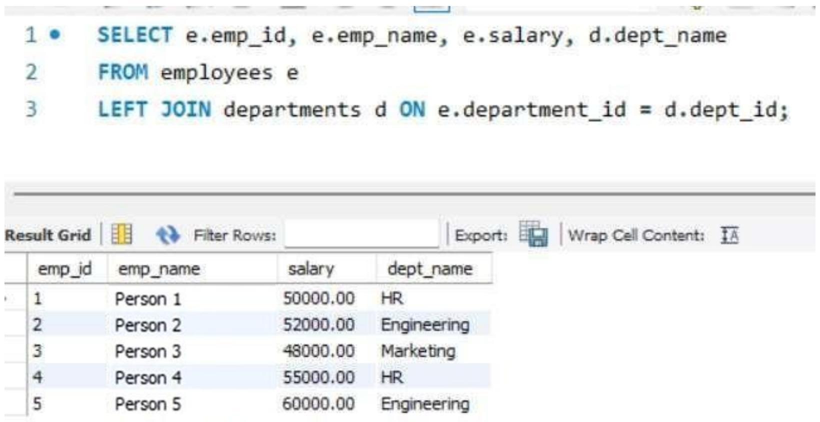
```
SELECT e.emp_id, e.emp_name, e.salary, d.dept_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.dept_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

emp_id	emp_name	salary	dept_name
1	Person 1	50000.00	HR
2	Person 2	52000.00	Engineering
3	Person 3	48000.00	Marketing
4	Person 4	55000.00	HR
5	Person 5	60000.00	Engineering

ii) Left Join:

- Retrieves all employees, even those without a department.



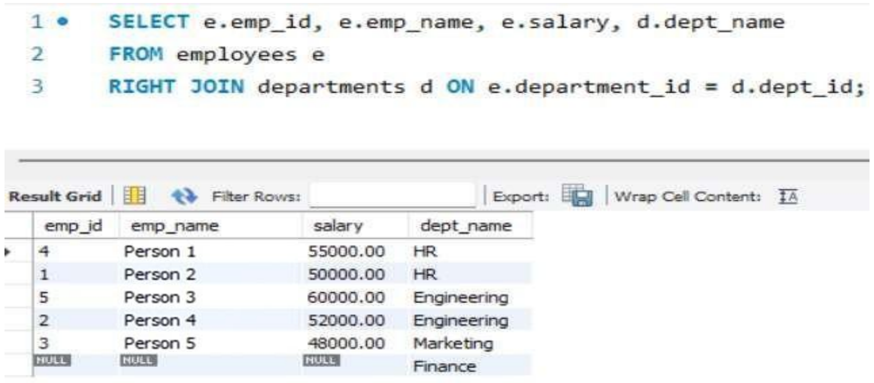
```
SELECT e.emp_id, e.emp_name, e.salary, d.dept_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.dept_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

emp_id	emp_name	salary	dept_name
1	Person 1	50000.00	HR
2	Person 2	52000.00	Engineering
3	Person 3	48000.00	Marketing
4	Person 4	55000.00	HR
5	Person 5	60000.00	Engineering

iii) Right Join:

- Retrieves all departments, even if they have no employees.



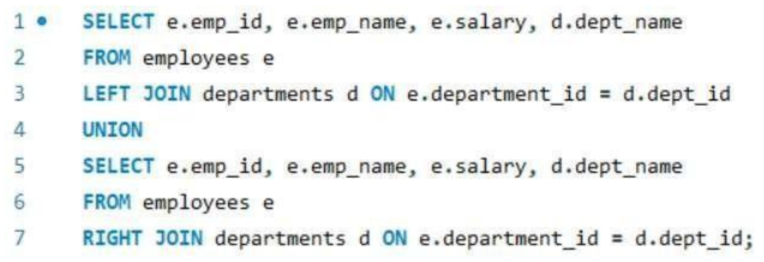
```
SELECT e.emp_id, e.emp_name, e.salary, d.dept_name
FROM employees e
RIGHT JOIN departments d ON e.department_id = d.dept_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

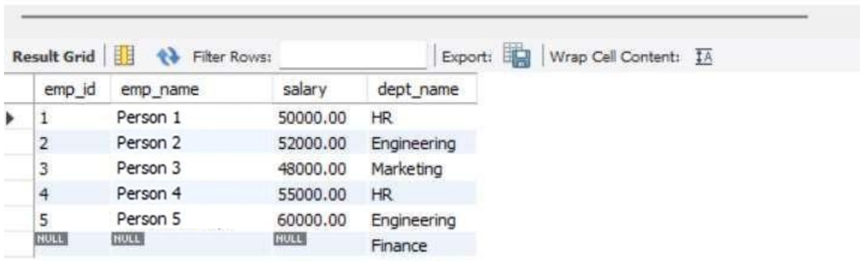
emp_id	emp_name	salary	dept_name
4	Person 1	55000.00	HR
1	Person 2	50000.00	HR
5	Person 3	60000.00	Engineering
2	Person 4	52000.00	Engineering
3	Person 5	48000.00	Marketing
NULL	NULL	NULL	Finance

iv) Full Outer Join:

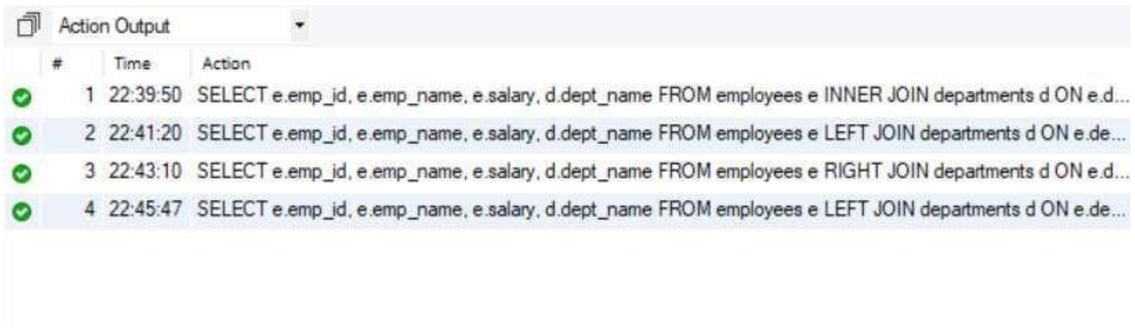
- Retrieves all employees and all departments, including those without matches on either side.



```
SELECT e.emp_id, e.emp_name, e.salary, d.dept_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.dept_id
UNION
SELECT e.emp_id, e.emp_name, e.salary, d.dept_name
FROM employees e
RIGHT JOIN departments d ON e.department_id = d.dept_id;
```



emp_id	emp_name	salary	dept_name
1	Person 1	50000.00	HR
2	Person 2	52000.00	Engineering
3	Person 3	48000.00	Marketing
4	Person 4	55000.00	HR
5	Person 5	60000.00	Engineering
NULL	NULL	NULL	Finance



```
#  Time      Action
1  22:39:50  SELECT e.emp_id, e.emp_name, e.salary, d.dept_name FROM employees e INNER JOIN departments d ON e.d...
2  22:41:20  SELECT e.emp_id, e.emp_name, e.salary, d.dept_name FROM employees e LEFT JOIN departments d ON e.de...
3  22:43:10  SELECT e.emp_id, e.emp_name, e.salary, d.dept_name FROM employees e RIGHT JOIN departments d ON e.d...
4  22:45:47  SELECT e.emp_id, e.emp_name, e.salary, d.dept_name FROM employees e LEFT JOIN departments d ON e.de...
```