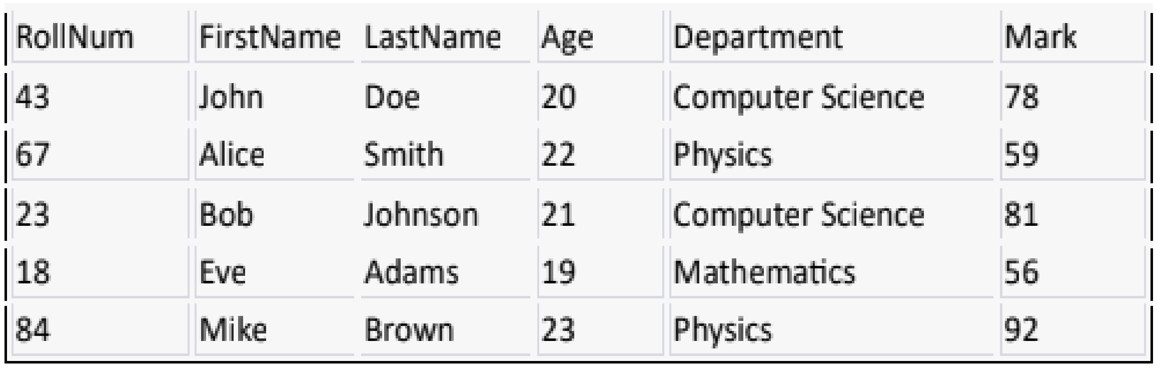
ASSIGNMENT – 5
Advanced Database Systems LAB
Name: Utsav Balar
Roll no: T24CS003

1. Design a MongoDB schema for a "Student" collection with the following fields:
    a. RollNum
    b. FirstName
    c. LastName
    d. Age
    e. Department
    f. Mark
2. Insert the following student data in the collection.



RollNum	FirstName	LastName	Age	Department	Mark
43	John	Doe	20	Computer Science	78
67	Alice	Smith	22	Physics	59
23	Bob	Johnson	21	Computer Science	81
18	Eve	Adams	19	Mathematics	56
84	Mike	Brown	23	Physics	92

3. Write a MongoDB query to find all students.
4. Write a MongoDB query to find all students in the "Computer Science" department.
5. Write a MongoDB query to find all students whose age is greater than or equal to 20.
6. Write a MongoDB query to find all students whose mark is less than 60.
7. Write a MongoDB query to show the first name and Mark of all students in the "Physics" department.
8. Write a MongoDB query to find all students in the descending order of Mark.
9. Write a MongoDB query to find the youngest student.
10. Write a MongoDB query to find all students in the "Physics" department whose RollNum is greater than or equal to 70.

Code:

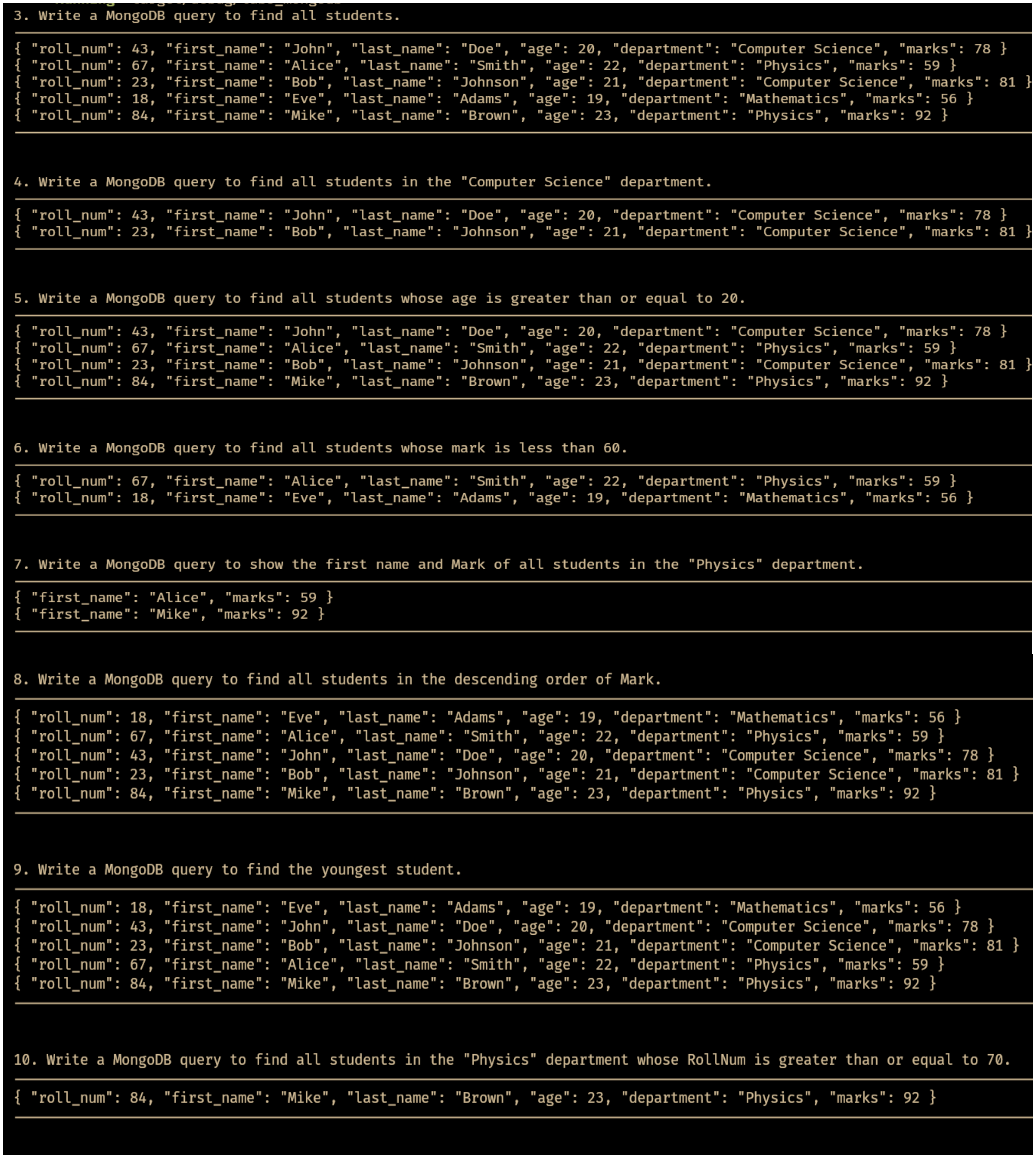
```
#![allow(dead_code)]

extern crate mongodb;
use futures_util::TryStreamExt;
use mongodb::bson::{doc, Document};
use mongodb::{options::ClientOptions, Client};
use serde::{Deserialize, Serialize};
use std::fmt::Formatter;
use termion::terminal_size;

#[derive(Serialize, Deserialize)]
struct Student {
    roll_num: usize,
    first_name: String,
    last_name: String,
    age: usize,
    department: String,
    marks: usize,
}

impl std::fmt::Display for Student {
    fn fmt(&self, f: &mut Formatter<'_>) -> std::fmt::Result {
        write!(
            f,
            "Roll Number: {}, First Name: {}, Last Name: {}, Age:
{}, Department: {}, Marks: {}",
            self.roll_num, self.first_name, self.last_name,
self.age, self.department, self.marks
        )
    }
}

impl Student {
    fn new(
        roll_num: usize,
        first_name: String,
        last_name: String,
        age: usize,
        department: String,
        marks: usize,
    ) -> Self {
        Self {
            roll_num,
            first_name,
            last_name,
            age,
            department,
            marks,
        }
    }
}

async fn get_filtered_documents(
    client: &Client,
    db_name: &str,
    coll_name: &str,
    filter: Document,
    projection: Option<Document>,
) {
    let db = client.database(db_name);
    let coll: mongodb::Collection<Document> =
db.collection(coll_name);

    let mut cursor = coll
        .find(filter)
        .projection({
            if let Some(projection) = projection {
                projection
            } else {
                doc! {"_id": 0, "roll_num": 1, "first_name": 1,
"last_name": 1, "age": 1, "department": 1, "marks": 1}
            }
        })
        .await
        .unwrap();

    if let Ok((width, _height)) = terminal_size() {
        println!("{}", "-".repeat(width as usize));
    }

    while let Some(doc) = cursor.try_next().await.unwrap() {
        println!("{}", doc);
    }

    if let Ok((width, _height)) = terminal_size() {
        println!("{}", "-".repeat(width as usize));
    }
    println!("\n");
}
```

```rust
async fn insert_student(client: &Client, db_name: &str, doc:
Student, coll_name: &str) {
    let db = client.database(db_name);
    let coll = db.collection(coll_name);

    coll.insert_one(doc).await.unwrap();
}

async fn get_aggregated_documents(
    client: &Client,
    db_name: &str,
    coll_name: &str,
    filter: Vec<Document>,
) {
    let db = client.database(db_name);
    let coll: mongodb::Collection<Document> =
db.collection(coll_name);

    let mut results = coll
        .aggregate(filter)
        .await
        .expect("Failed to aggregate documents");

    if let Ok((width, _height)) = terminal_size() {
        println!("{}", "-".repeat(width as usize));
    }

    while let Some(doc) = results
        .try_next()
        .await
        .expect("Failed to get next document")
    {
        println!("{}", doc);
    }

    if let Ok((width, _height)) = terminal_size() {
        println!("{}", "-".repeat(width as usize));
    }
    println!("\n");
}

fn generate_students_data() → Vec<Student> {
    vec![
        Student::new(
            43,
            "John".to_string(),
            "Doe".to_string(),
            20,
            "Computer Science".to_string(),
            78,
        ),
        Student::new(
            67,
            "Alice".to_string(),
            "Smith".to_string(),
            22,
            "Physics".to_string(),
            59,
        ),
        Student::new(
            23,
            "Bob".to_string(),
            "Johnson".to_string(),
            21,
            "Computer Science".to_string(),
            81,
        ),
        Student::new(
            18,
            "Eve".to_string(),
            "Adams".to_string(),
            19,
            "Mathematics".to_string(),
            56,
        ),
        Student::new(
            84,
            "Mike".to_string(),
            "Brown".to_string(),
            23,
            "Physics".to_string(),
            92,
        ),
    ]
}

#[tokio::main]
async fn main() {
    let client_options =
ClientOptions::parse("mongodb://localhost:27017")
        .await
        .expect("ClientOptions failed to parse");
    let client =
Client::with_options(client_options).expect("Failed to create
client");
    let db_name: &str = "t24cs004_lab5";
    let db = client.database(db_name);
    let collection: &str = "cs553";

    db.create_collection(collection)
        .await
        .expect("Failed to create collection");

    let students: Vec<Student> = generate_students_data();

    for student in students {
        insert_student(&client, db_name, student,
collection).await;
    }

    println!("3. Write a MongoDB query to find all students.");
    let filter = doc! {"roll_num": doc! {"$exists": true}};
    get_filtered_documents(&client, db_name, collection, filter,
None).await;

    println!(
        "4. Write a MongoDB query to find all students in the
\"Computer Science\" department."
    );
    let filter = doc! {"department": "Computer Science"};
    get_filtered_documents(&client, db_name, collection, filter,
None).await;

    println!(
        "5. Write a MongoDB query to find all students whose age is
greater than or equal to 20."
    );
    let filter = doc! {"age": doc! {"$gte": 20}};
    get_filtered_documents(&client, db_name, collection, filter,
None).await;

    println!("6. Write a MongoDB query to find all students whose
mark is less than 60.");
    let filter = doc! {"marks": doc! {"$lt": 60}};
    get_filtered_documents(&client, db_name, collection, filter,
None).await;
```

```
    println!("7. Write a MongoDB query to show the first name and
Mark of all students in the \"Physics\" department.");
    let filter = doc! {"department": "Physics"};
    let projection = doc! {"first_name": 1, "marks": 1, "_id": 0};
    get_filtered_documents(&client, db_name, collection, filter,
Some(projection)).await;

    println!("8. Write a MongoDB query to find all students in the
descending order of Mark.");
    let filter = vec![
        doc! {
        "$sort": {
           "marks": 1
        }},
        doc! {
           "$project": {
              "_id": 0
           },
        },
    ];
    get_aggregated_documents(&client, db_name, collection,
filter).await;

    println!("9. Write a MongoDB query to find the youngest
student.");
    let filter = vec![
        doc! {
        "$sort": {
           "age": 1

         }},
        doc! {
            "$project": {
              "_id": 0
           },

        },
    ];
    get_aggregated_documents(&client, db_name, collection,
filter).await;

    println!("10. Write a MongoDB query to find all students in the
\"Physics\" department whose RollNum is greater than or equal to
70.");
    let filter = vec![
        doc! {
```

```
        "$match": {
           "department": "Physics",
           "roll_num": {
              "$gte": 70
           }
         }},
        doc! {
            "$project": {
              "_id": 0
           },

        },
    ];
    get_aggregated_documents(&client, db_name, collection,
filter).await;

    db.drop().await.expect("Failed to drop database");
}
```

Results:



```
3. Write a MongoDB query to find all students.
────────────────────────────────────────────────────────────────────────────────────────────────────────
{ "roll_num": 43, "first_name": "John", "last_name": "Doe", "age": 20, "department": "Computer Science", "marks": 78 }
{ "roll_num": 67, "first_name": "Alice", "last_name": "Smith", "age": 22, "department": "Physics", "marks": 59 }
{ "roll_num": 23, "first_name": "Bob", "last_name": "Johnson", "age": 21, "department": "Computer Science", "marks": 81 }
{ "roll_num": 18, "first_name": "Eve", "last_name": "Adams", "age": 19, "department": "Mathematics", "marks": 56 }
{ "roll_num": 84, "first_name": "Mike", "last_name": "Brown", "age": 23, "department": "Physics", "marks": 92 }
────────────────────────────────────────────────────────────────────────────────────────────────────────


4. Write a MongoDB query to find all students in the "Computer Science" department.
────────────────────────────────────────────────────────────────────────────────────────────────────────
{ "roll_num": 43, "first_name": "John", "last_name": "Doe", "age": 20, "department": "Computer Science", "marks": 78 }
{ "roll_num": 23, "first_name": "Bob", "last_name": "Johnson", "age": 21, "department": "Computer Science", "marks": 81 }
────────────────────────────────────────────────────────────────────────────────────────────────────────


5. Write a MongoDB query to find all students whose age is greater than or equal to 20.
────────────────────────────────────────────────────────────────────────────────────────────────────────
{ "roll_num": 43, "first_name": "John", "last_name": "Doe", "age": 20, "department": "Computer Science", "marks": 78 }
{ "roll_num": 67, "first_name": "Alice", "last_name": "Smith", "age": 22, "department": "Physics", "marks": 59 }
{ "roll_num": 23, "first_name": "Bob", "last_name": "Johnson", "age": 21, "department": "Computer Science", "marks": 81 }
{ "roll_num": 84, "first_name": "Mike", "last_name": "Brown", "age": 23, "department": "Physics", "marks": 92 }
────────────────────────────────────────────────────────────────────────────────────────────────────────


6. Write a MongoDB query to find all students whose mark is less than 60.
────────────────────────────────────────────────────────────────────────────────────────────────────────
{ "roll_num": 67, "first_name": "Alice", "last_name": "Smith", "age": 22, "department": "Physics", "marks": 59 }
{ "roll_num": 18, "first_name": "Eve", "last_name": "Adams", "age": 19, "department": "Mathematics", "marks": 56 }
────────────────────────────────────────────────────────────────────────────────────────────────────────


7. Write a MongoDB query to show the first name and Mark of all students in the "Physics" department.
────────────────────────────────────────────────────────────────────────────────────────────────────────
{ "first_name": "Alice", "marks": 59 }
{ "first_name": "Mike", "marks": 92 }
────────────────────────────────────────────────────────────────────────────────────────────────────────


8. Write a MongoDB query to find all students in the descending order of Mark.
────────────────────────────────────────────────────────────────────────────────────────────────────────
{ "roll_num": 18, "first_name": "Eve", "last_name": "Adams", "age": 19, "department": "Mathematics", "marks": 56 }
{ "roll_num": 67, "first_name": "Alice", "last_name": "Smith", "age": 22, "department": "Physics", "marks": 59 }
{ "roll_num": 43, "first_name": "John", "last_name": "Doe", "age": 20, "department": "Computer Science", "marks": 78 }
{ "roll_num": 23, "first_name": "Bob", "last_name": "Johnson", "age": 21, "department": "Computer Science", "marks": 81 }
{ "roll_num": 84, "first_name": "Mike", "last_name": "Brown", "age": 23, "department": "Physics", "marks": 92 }
────────────────────────────────────────────────────────────────────────────────────────────────────────


9. Write a MongoDB query to find the youngest student.
────────────────────────────────────────────────────────────────────────────────────────────────────────
{ "roll_num": 18, "first_name": "Eve", "last_name": "Adams", "age": 19, "department": "Mathematics", "marks": 56 }
{ "roll_num": 43, "first_name": "John", "last_name": "Doe", "age": 20, "department": "Computer Science", "marks": 78 }
{ "roll_num": 23, "first_name": "Bob", "last_name": "Johnson", "age": 21, "department": "Computer Science", "marks": 81 }
{ "roll_num": 67, "first_name": "Alice", "last_name": "Smith", "age": 22, "department": "Physics", "marks": 59 }
{ "roll_num": 84, "first_name": "Mike", "last_name": "Brown", "age": 23, "department": "Physics", "marks": 92 }
────────────────────────────────────────────────────────────────────────────────────────────────────────


10. Write a MongoDB query to find all students in the "Physics" department whose RollNum is greater than or equal to 70.
────────────────────────────────────────────────────────────────────────────────────────────────────────
{ "roll_num": 84, "first_name": "Mike", "last_name": "Brown", "age": 23, "department": "Physics", "marks": 92 }
────────────────────────────────────────────────────────────────────────────────────────────────────────
```