ASSIGNMENT – 7
Advanced Database Systems LAB
Name: Utsav Balar
Roll no: T24CS003

1. Design a MongoDB schema for an "Employee" collection with the following fields:
    a. EmployeeID
    b. FirstName
    c. LastName
    d. Age
    e. Department
    f. Salary

2. Insert the following employee data into the collection.

EmployeeID	FirstName	LastName	Age	Department	Salary
121	Emma	Johnson	30	Human Resources	50000
134	David	Smith	34	Marketing	55000
145	Mia	Davis	28	Information Tech	62000
167	Lucas	Brown	40	Sales	48000
153	Sophia	Wilson	33	Research	53000

3. Write a MongoDB query to find all employees.
4. Write a MongoDB query to find all employees in the "Marketing" department.
5. Write a MongoDB query to find all employees whose age is greater than or equal to 30.
6. Write a MongoDB query to find all employees whose salary is less than 50000.
7. Write a MongoDB query to show the first name and salary of all employees in the "Information Tech" department.
8. Write a MongoDB query to find all employees in descending order of salary.
9. Write a MongoDB query to find the oldest employee.
10. Write a MongoDB query to find all employees in the "Sales" department whose EmployeeID is greater than or equal to 150.

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 termion::terminal_size;

#[derive(Serialize, Deserialize, Debug)]
struct Employee {
    employee_id: usize,
    first_name: String,
    last_name: String,
    age: usize,
    department: String,
    salary: usize,
}

type CollectionType = Employee;

impl CollectionType {
    fn new(
        employee_id: usize,
        first_name: String,
        last_name: String,
        age: usize,
        department: String,
        salary: usize,
    ) → Self {
        Self {
            employee_id,
            first_name,
            last_name,
            age,
            department,
            salary,
        }
    }
}

fn create_employee_db() → Vec<CollectionType> {
    // EmployeeID FirstName LastName Age Department Salary
    // 121 Emma Johnson 30 Human Resources 50000
```

```
// 134 David Smith 34 Marketing 55000
// 145 Mia Davis 28 Information Tech 62000
// 167 Lucas Brown 40 Sales 48000
// 153 Sophia Wilson 33 Research 53000
vec![
    CollectionType::new(
        121,
        "Emma".to_string(),
        "Johnson".to_string(),
        30,
        "Human Resources".to_string(),
        50000,
    ),
    CollectionType::new(
        134,
        "David".to_string(),
        "Smith".to_string(),
        34,
        "Marketing".to_string(),
        55000,
    ),
    CollectionType::new(
        145,
        "Mia".to_string(),
        "Davis".to_string(),
        28,
        "Information Tech".to_string(),
        62000,
    ),
    CollectionType::new(
        167,
        "Lucas".to_string(),
        "Brown".to_string(),
        40,
        "Sales".to_string(),
        48000,
    ),
    CollectionType::new(
        153,
        "Sophia".to_string(),
        "Wilson".to_string(),
        33,
        "Research".to_string(),
        53000,
    ),
]
```

```
}

async fn insert_employee(client: &Client, db_name: &str, doc:
CollectionType, coll_name: &str) {
    let db = client.database(db_name);
    let coll = db.collection(coll_name);

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

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, "employee_id": 1, "first_name": 1,
"last_name": 1, "age": 1, "department": 1, "salary": 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");
```

```
}

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");
}

async fn get_aggregated_documents_with_limit(
    client: &Client,
    db_name: &str,
    coll_name: &str,
    filter: Vec<Document>,
    limit: usize,
) {
    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));
    }

    let mut count = 0;
    while let Some(doc) = results
        .try_next()
        .await
        .expect("Failed to get next document")
    {
        if count == limit {
            break;
        }
        count += 1;
        println!("{}", doc);
    }

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

#[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 = "t24cs003_lab7";
    let db = client.database(db_name);
    let collection: &str = "cs553";

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

    let employees: Vec<CollectionType> = create_employee_db();
```

```
    for employee in employees {
        insert_employee(&client, db_name, employee,
collection).await;
    }

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

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

    println!(
        "5. Write a MongoDB query to find all employees whose age
is greater than or equal to 30."
    );
    let filter = doc! {"age": doc! {"$gte": 30}};
    get_filtered_documents(&client, db_name, collection, filter,
None).await;

    println!("6. Write a MongoDB query to find all employees whose
salary is less than 50000.");
    let filter = doc! {"salary": doc! {"$lt": 50000}};
    get_filtered_documents(&client, db_name, collection, filter,
None).await;

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

    println!("8. Write a MongoDB query to find all employees in
descending order of salary.");
    let filter = vec![
        doc! {
        "$sort": {
           "salary": -1
        }},
        doc! {
```

```
            "$project": {
                "_id": 0
            },
        },
    ];
    get_aggregated_documents(&client, db_name, collection,
filter).await;

    println!("9. Write a MongoDB query to find the oldest
employee.");
    let filter = vec![
        doc! {
        "$sort": {
            "age": -1
        }},
        doc! {
            "$project": {
                "_id": 0
            },
        },
    ];
    get_aggregated_documents_with_limit(&client, db_name,
collection, filter, 1).await;

    println!("10. Write a MongoDB query to find all employees in
the \"Sales\" department whose EmployeeID is greater than or equal
to 150.");
    let filter = doc! {"department": "Sales", "employee_id": doc!
{"$gte": 150}};
    get_filtered_documents(&client, db_name, collection, filter,
None).await;

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

Results:

```
3. Write a MongoDB query to find all employees.
{ "employee_id": 121, "first_name": "Emma", "last_name": "Johnson", "age": 30, "department": "Human Resources", "salary": 50000 }
{ "employee_id": 134, "first_name": "David", "last_name": "Smith", "age": 34, "department": "Marketing", "salary": 55000 }
{ "employee_id": 145, "first_name": "Mia", "last_name": "Davis", "age": 28, "department": "Information Tech", "salary": 62000 }
{ "employee_id": 167, "first_name": "Lucas", "last_name": "Brown", "age": 40, "department": "Sales", "salary": 48000 }
{ "employee_id": 153, "first_name": "Sophia", "last_name": "Wilson", "age": 33, "department": "Research", "salary": 53000 }

4. Write a MongoDB query to find all employees in the "Marketing" department.
{ "employee_id": 134, "first_name": "David", "last_name": "Smith", "age": 34, "department": "Marketing", "salary": 55000 }

5. Write a MongoDB query to find all employees whose age is greater than or equal to 30.
{ "employee_id": 121, "first_name": "Emma", "last_name": "Johnson", "age": 30, "department": "Human Resources", "salary": 50000 }
{ "employee_id": 134, "first_name": "David", "last_name": "Smith", "age": 34, "department": "Marketing", "salary": 55000 }
{ "employee_id": 167, "first_name": "Lucas", "last_name": "Brown", "age": 40, "department": "Sales", "salary": 48000 }
{ "employee_id": 153, "first_name": "Sophia", "last_name": "Wilson", "age": 33, "department": "Research", "salary": 53000 }

6. Write a MongoDB query to find all employees whose salary is less than 50000.
{ "employee_id": 167, "first_name": "Lucas", "last_name": "Brown", "age": 40, "department": "Sales", "salary": 48000 }

7. Write a MongoDB query to show the first name and salary of all employees in the Information Tech department.
{ "first_name": "Mia", "salary": 62000 }
```

```
8. Write a MongoDB query to find all employees in descending order of salary.
{ "employee_id": 145, "first_name": "Mia", "last_name": "Davis", "age": 28, "department": "Information
{ "employee_id": 134, "first_name": "David", "last_name": "Smith", "age": 34, "department": "Marketing"
{ "employee_id": 153, "first_name": "Sophia", "last_name": "Wilson", "age": 33, "department": "Research
{ "employee_id": 121, "first_name": "Emma", "last_name": "Johnson", "age": 30, "department": "Human Res
{ "employee_id": 167, "first_name": "Lucas", "last_name": "Brown", "age": 40, "department": "Sales", "s

9. Write a MongoDB query to find the oldest employee.
{ "employee_id": 167, "first_name": "Lucas", "last_name": "Brown", "age": 40, "department": "Sales", "s

10. Write a MongoDB query to find all employees in the "Sales" department whose EmployeeID is greater t
{ "employee_id": 167, "first_name": "Lucas", "last_name": "Brown", "age": 40, "department": "Sales", "s
```