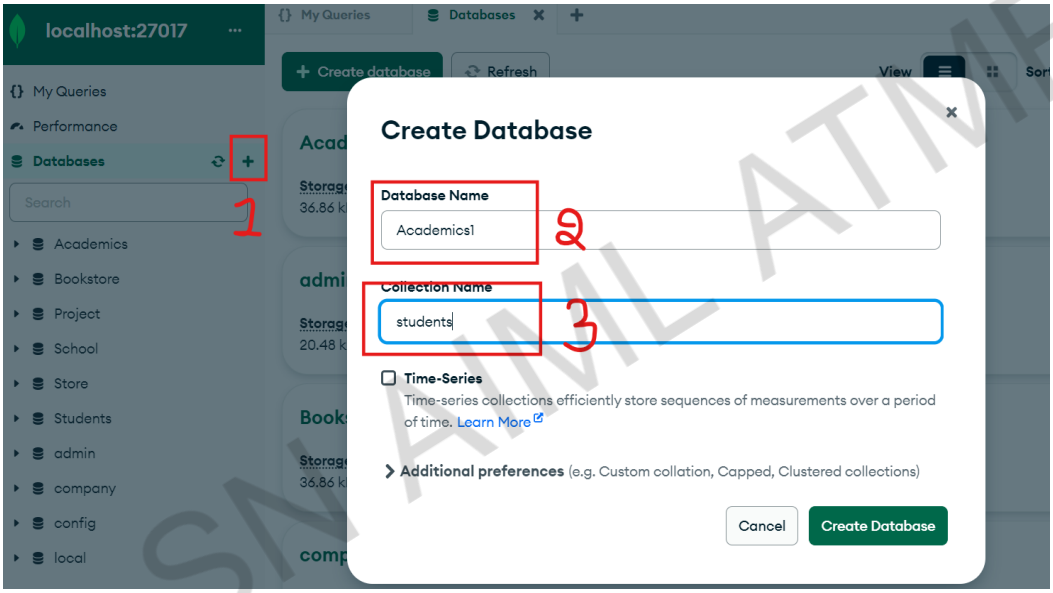
Program 6

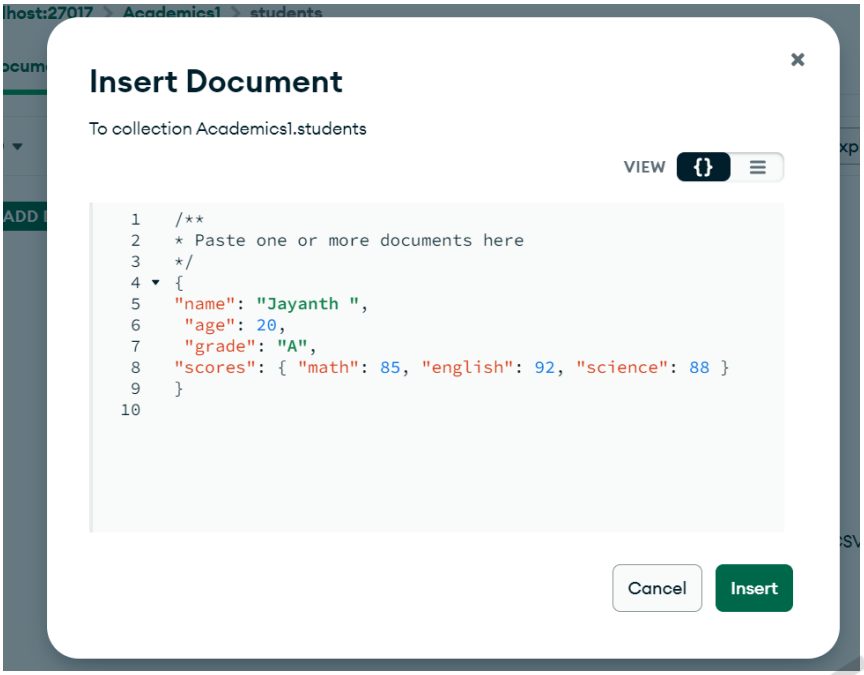
Execute Aggregation Pipeline and its operations (pipeline must contain $match, $group, $sort, $project, $skip etc. students encourage to execute several queries to demonstrate various aggregation operators)

Create a database **Academics1** and collection **students** in Mongo IDE.



Add the following documents in the **students collection** in MongoDB IDE.

```
{
        "name": "Jayanth ",
         "age": 20,
         "grade": "A",
        "scores": { "math": 85, "english": 92, "science": 88 }
}
```



{

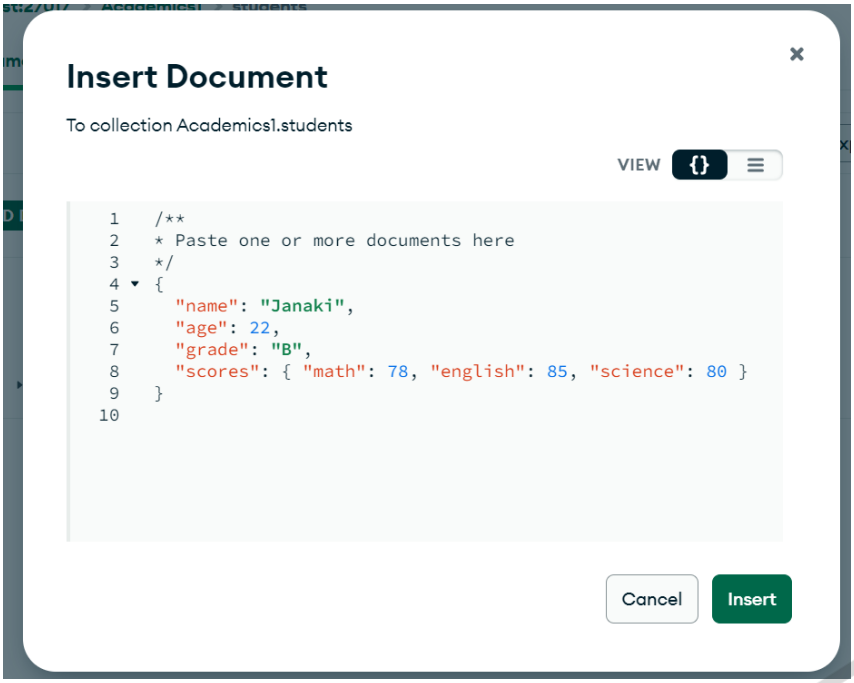
"name": "Janaki",

"age": 22,

"grade": "B",

"scores": { "math": 78, "english": 85, "science": 80 }

}



{

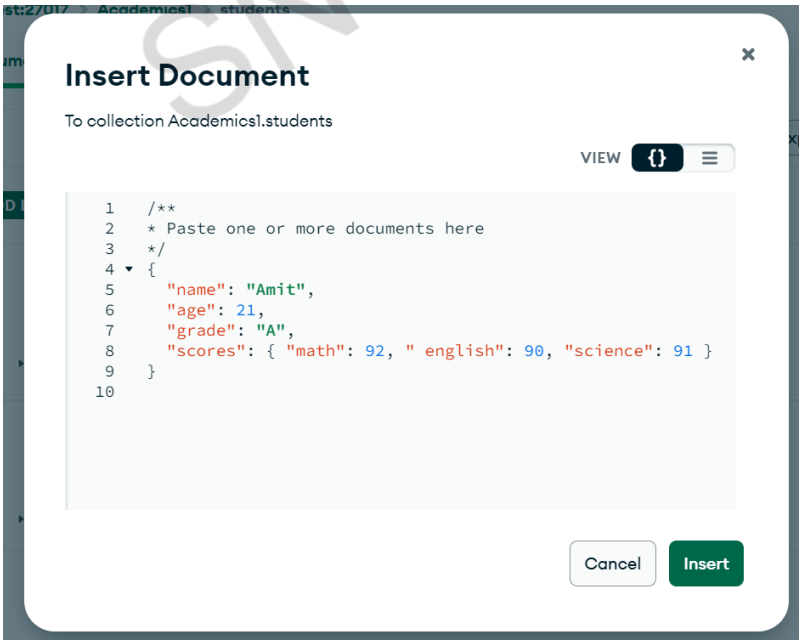
"name": "Amit",

"age": 21,

"grade": "A",

"scores": { "math": 92, " english": 90, "science": 91 }

}



{

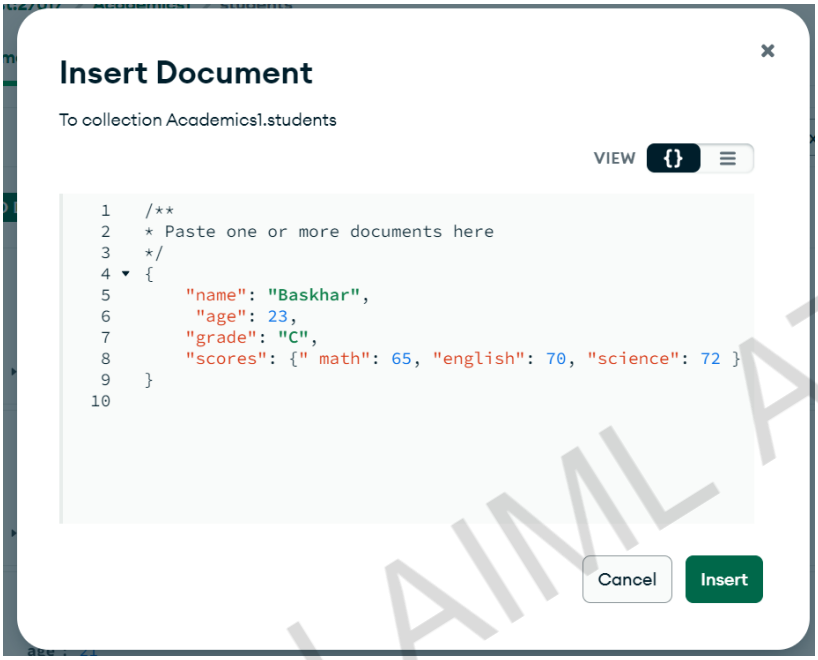
"name": "Baskhar",

"age": 23,

"grade": "C",

"scores": {" math": 65, "english": 70, "science": 72 }

}



{

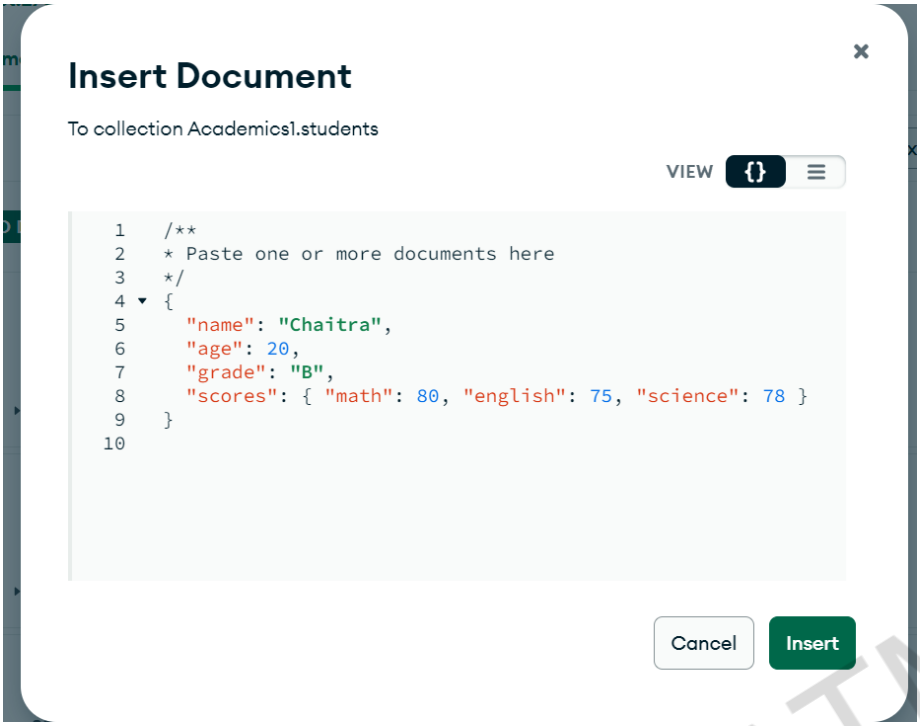
"name": "Chaitra",

"age": 20,

"grade": "B",

"scores": { "math": 80, "english": 75, "science": 78 }

}



In MongoDB Shell

>use Academics1

Now, let's execute an aggregation pipeline with several stages:

1. **$match**: Filter students who are 21 years or older.
2. **$group**: Group by grade and calculate the average age.
3. **$sort**: Sort by average age in descending order.
4. **$project**: Project the grade and average age.
5. **$skip**: Skip the first result.

db.students.aggregate([

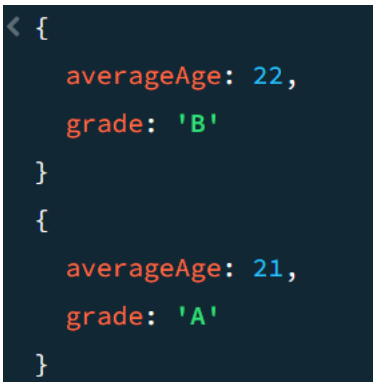
{

$match: { age: { $gte: 21 } }

},

```
{
     $group: {
     _id: "$grade",
     averageAge: { $avg: "$age" }
     }
},
{
   $sort: { averageAge: -1 }
},
{
   $project: {
     _id: 0,
     grade: "$_id",
     averageAge: 1
   }
},
{
   $skip: 1
}])
```

Output:



Let's break down each stage:

1. **$match**: Filters documents to include only those where `age` is greater than or equal to 21.
2. **$group**: Groups the documents by `grade` and computes the average age for each grade.
3. **$sort**: Sorts the resulting documents by `averageAge` in descending order.
4. **$project**: Projects the fields `grade` and `averageAge`, excluding the `_id` field.
5. **$skip**: Skips the first document in the sorted results.

When you execute this pipeline, you will get a result that first filters students by age, groups them by grade, calculates the average age, sorts by this average age in descending order, and finally skips the first result.