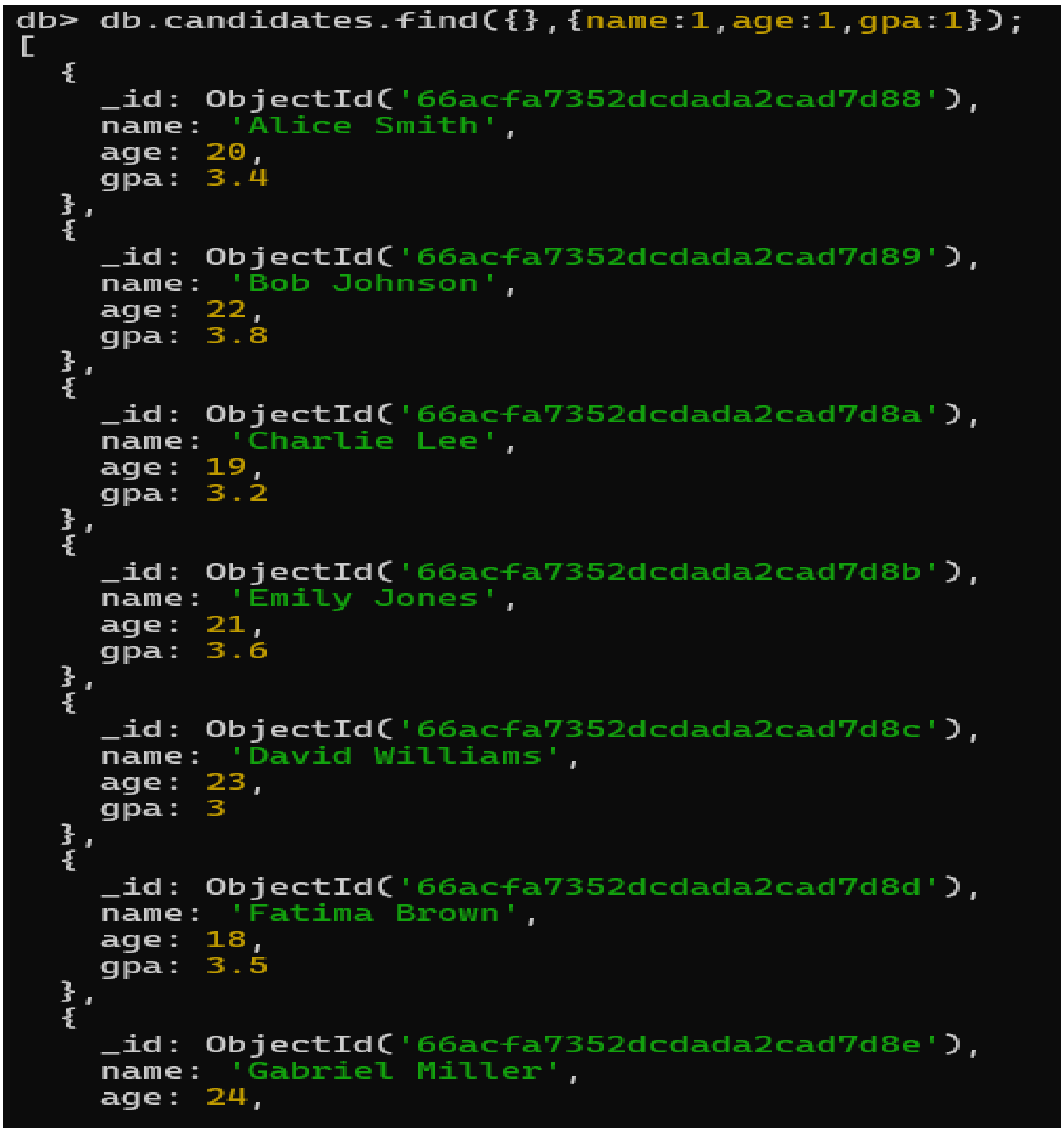
CLASS 5 : PROJECTION OPERATORS

➢ PROJECTION:

Create new collection called candidates.

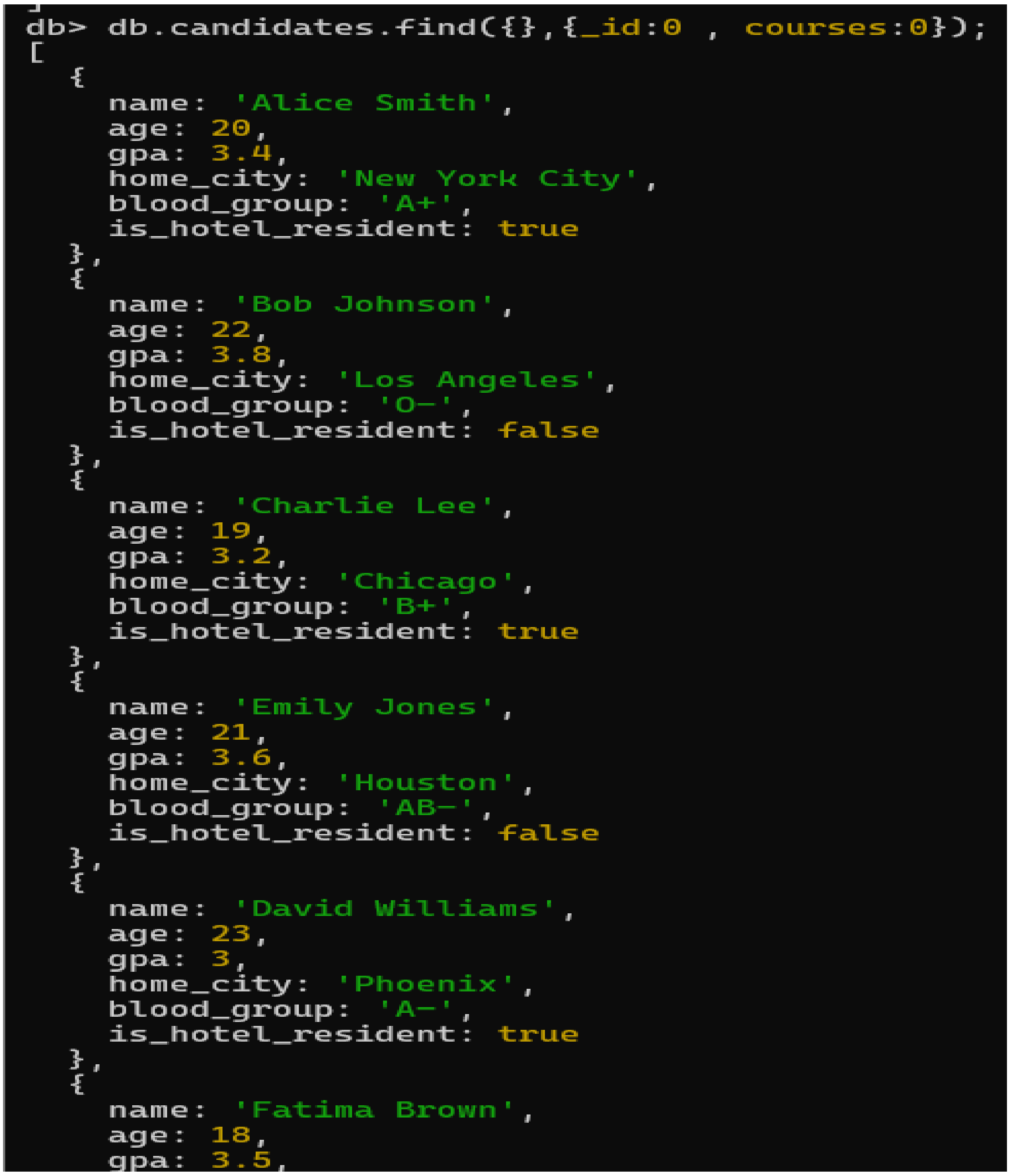
- Example 1: Retrieve Name , Age and GPA



```
db> db.candidates.find({},{name:1,age:1,gpa:1});
[
  {
    _id: ObjectId('66acfa7352dcdada2cad7d88'),
    name: 'Alice Smith',
    age: 20,
    gpa: 3.4
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d89'),
    name: 'Bob Johnson',
    age: 22,
    gpa: 3.8
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d8a'),
    name: 'Charlie Lee',
    age: 19,
    gpa: 3.2
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d8b'),
    name: 'Emily Jones',
    age: 21,
    gpa: 3.6
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d8c'),
    name: 'David Williams',
    age: 23,
    gpa: 3
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d8d'),
    name: 'Fatima Brown',
    age: 18,
    gpa: 3.5
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d8e'),
    name: 'Gabriel Miller',
    age: 24,
```

In this we are including every field (Name , Age and GPA).

Now we are doing some variations .



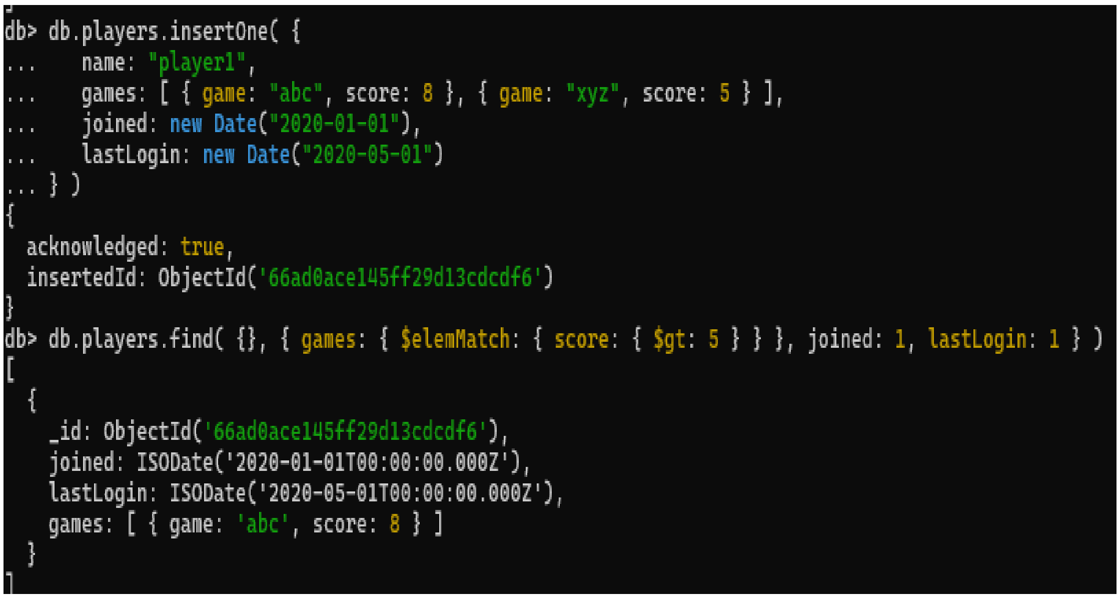
```
db> db.candidates.find({},{_id:0 , courses:0});
[
  {
    name: 'Alice Smith',
    age: 20,
    gpa: 3.4,
    home_city: 'New York City',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    name: 'Bob Johnson',
    age: 22,
    gpa: 3.8,
    home_city: 'Los Angeles',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    name: 'Charlie Lee',
    age: 19,
    gpa: 3.2,
    home_city: 'Chicago',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    name: 'Emily Jones',
    age: 21,
    gpa: 3.6,
    home_city: 'Houston',
    blood_group: 'AB-',
    is_hotel_resident: false
  },
  {
    name: 'David Williams',
    age: 23,
    gpa: 3,
    home_city: 'Phoenix',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    name: 'Fatima Brown',
    age: 18,
    gpa: 3.5,
```

Exclude Fields : _id and courses.

➢ $elemMatch:

The $elemMatch operator limits the contents of an <array> field from the query results to contain only the **first** element matching the $elemMatch condition.

- Example 2:



```
db> db.players.insertOne( {
...    name: "player1",
...    games: [ { game: "abc", score: 8 }, { game: "xyz", score: 5 } ],
...    joined: new Date("2020-01-01"),
...    lastLogin: new Date("2020-05-01")
... } )
{
  acknowledged: true,
  insertedId: ObjectId('66ad0ace145ff29d13cdcdf6')
}
db> db.players.find( {}, { games: { $elemMatch: { score: { $gt: 5 } } }, joined: 1, lastLogin: 1 } )
[
  {
    _id: ObjectId('66ad0ace145ff29d13cdcdf6'),
    joined: ISODate('2020-01-01T00:00:00.000Z'),
    lastLogin: ISODate('2020-05-01T00:00:00.000Z'),
    games: [ { game: 'abc', score: 8 } ]
  }
]
```

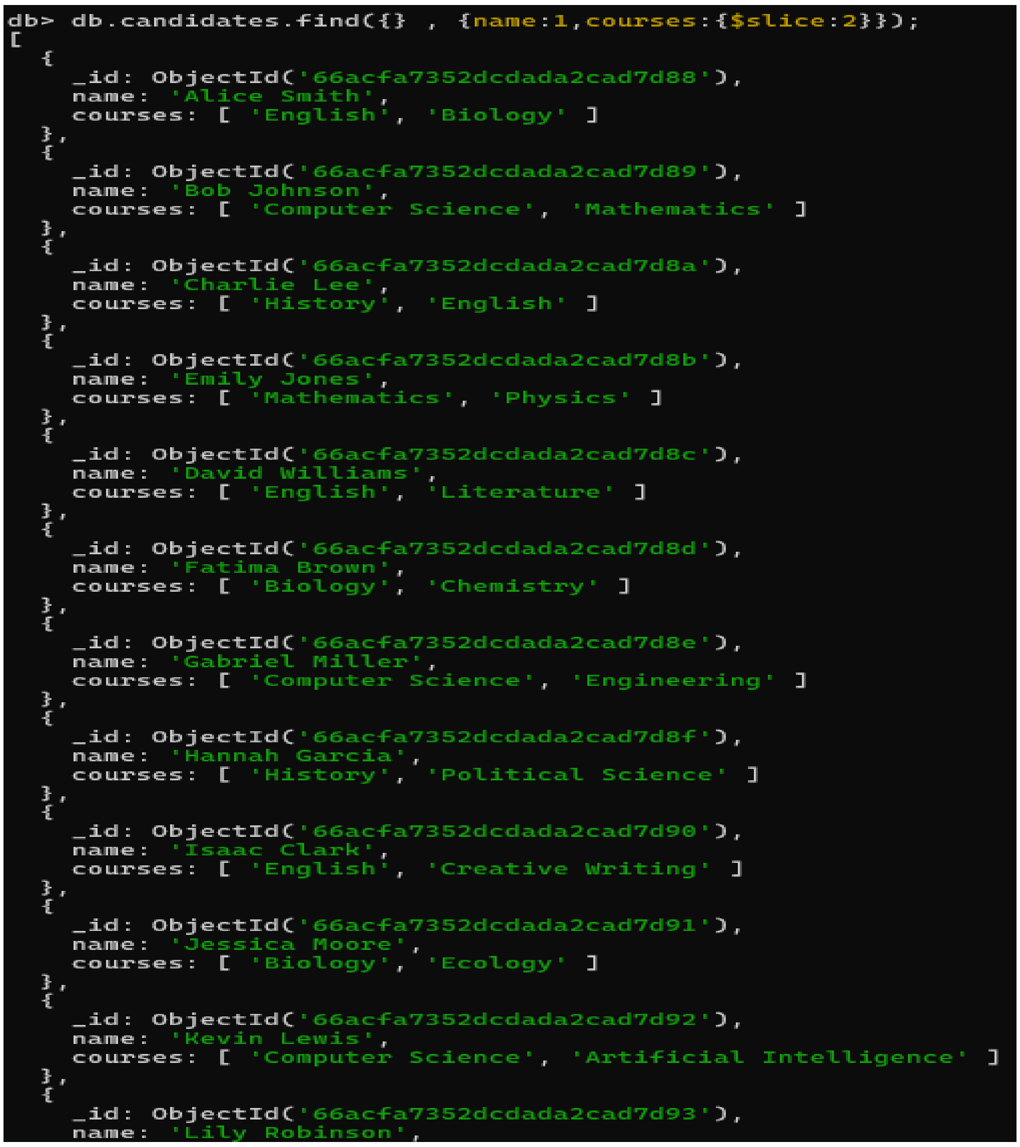
$slice:

The $slice projection operator specifies the number of elements in an array to return in the query result.

Syntax:

```
db.collection.find(
      <query>,
       { <arrayField>:{$slice:<number>}}
);
```

- Example 3 : Retrieve all candidates with first two courses.


```
db> db.candidates.find({} , {name:1,courses:{$slice:2}});
[
  {
    _id: ObjectId('66acfa7352dcdada2cad7d88'),
    name: 'Alice Smith',
    courses: [ 'English', 'Biology' ]
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d89'),
    name: 'Bob Johnson',
    courses: [ 'Computer Science', 'Mathematics' ]
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d8a'),
    name: 'Charlie Lee',
    courses: [ 'History', 'English' ]
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d8b'),
    name: 'Emily Jones',
    courses: [ 'Mathematics', 'Physics' ]
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d8c'),
    name: 'David Williams',
    courses: [ 'English', 'Literature' ]
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d8d'),
    name: 'Fatima Brown',
    courses: [ 'Biology', 'Chemistry' ]
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d8e'),
    name: 'Gabriel Miller',
    courses: [ 'Computer Science', 'Engineering' ]
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d8f'),
    name: 'Hannah Garcia',
    courses: [ 'History', 'Political Science' ]
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d90'),
    name: 'Isaac Clark',
    courses: [ 'English', 'Creative Writing' ]
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d91'),
    name: 'Jessica Moore',
    courses: [ 'Biology', 'Ecology' ]
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d92'),
    name: 'Kevin Lewis',
    courses: [ 'Computer Science', 'Artificial Intelligence' ]
  },
  {
    _id: ObjectId('66acfa7352dcdada2cad7d93'),
    name: 'Lily Robinson',
```

Class 6: Aggregation Operators

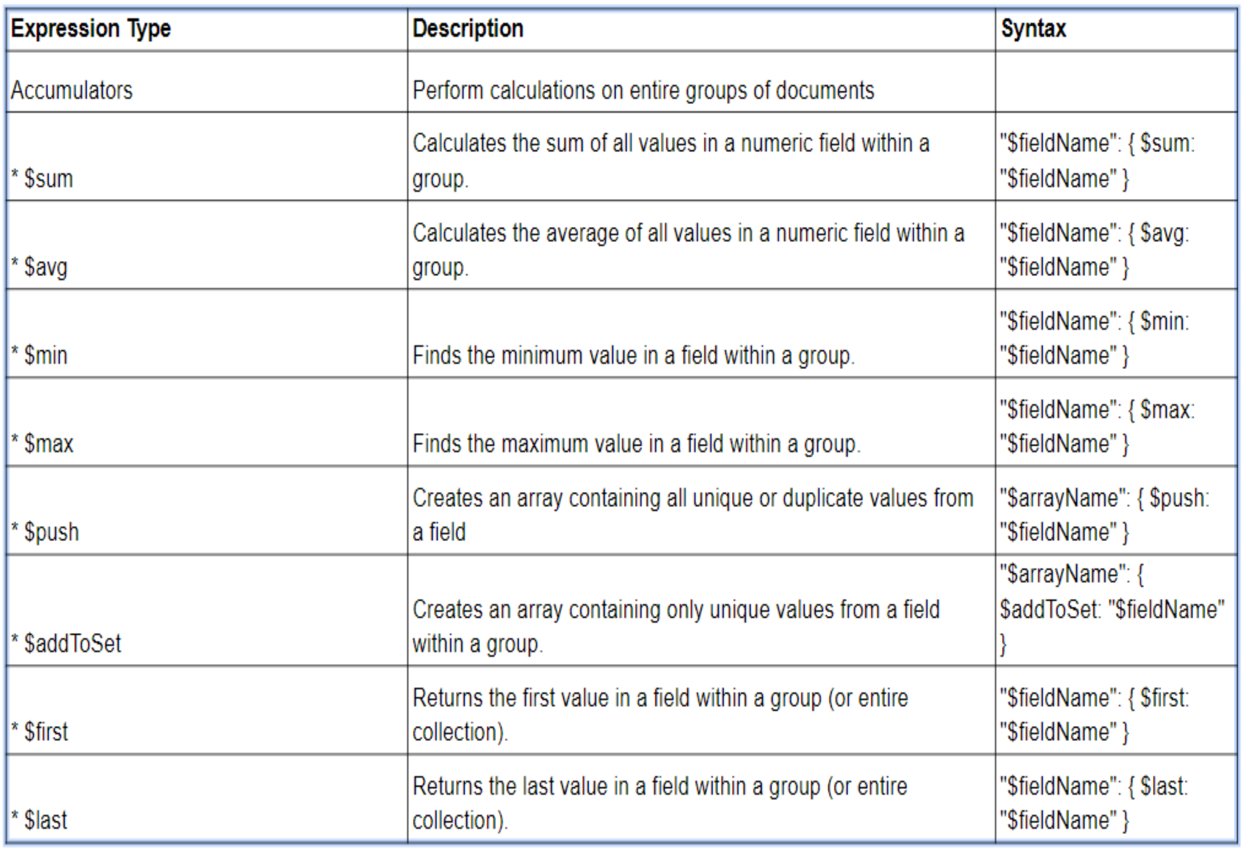
➢ Introduction:

- Aggregation means grouping together
- For example sum, avg, min, max etc…

Syntax:

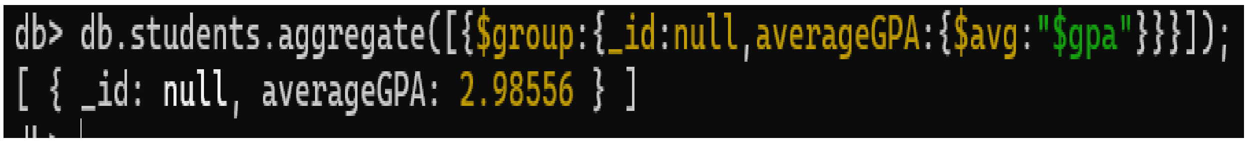
db.collection.aggregate(<AGGREGATE OPERATION>

Types:



Expression Type	Description	Syntax
Accumulators	Perform calculations on entire groups of documents	
* $sum	Calculates the sum of all values in a numeric field within a group.	"$fieldName": { $sum: "$fieldName" }
* $avg	Calculates the average of all values in a numeric field within a group.	"$fieldName": { $avg: "$fieldName" }
* $min	Finds the minimum value in a field within a group.	"$fieldName": { $min: "$fieldName" }
* $max	Finds the maximum value in a field within a group.	"$fieldName": { $max: "$fieldName" }
* $push	Creates an array containing all unique or duplicate values from a field	"$arrayName": { $push: "$fieldName" }
* $addToSet	Creates an array containing only unique values from a field within a group.	"$arrayName": { $addToSet: "$fieldName" }
* $first	Returns the first value in a field within a group (or entire collection).	"$fieldName": { $first: "$fieldName" }
* $last	Returns the last value in a field within a group (or entire collection).	"$fieldName": { $last: "$fieldName" }

<u>Example 1</u>: To find average GPA of all students.

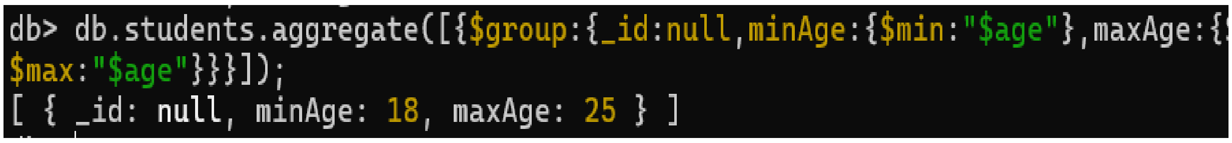


```
db> db.students.aggregate([{$group:{_id:null,averageGPA:{$avg:"$gpa"}}}]);
[ { _id: null, averageGPA: 2.98556 } ]
```

Explanation:

- ✓ $group: Groups all documents together.
- ✓ _id: null: Sets the group identifier to null (optional, as there's only one group in this case).
- ✓ averageGPA: Calculates the average value of the "gpa" field using the $avg operator.

Example 2: Minimum and Maximum Age.

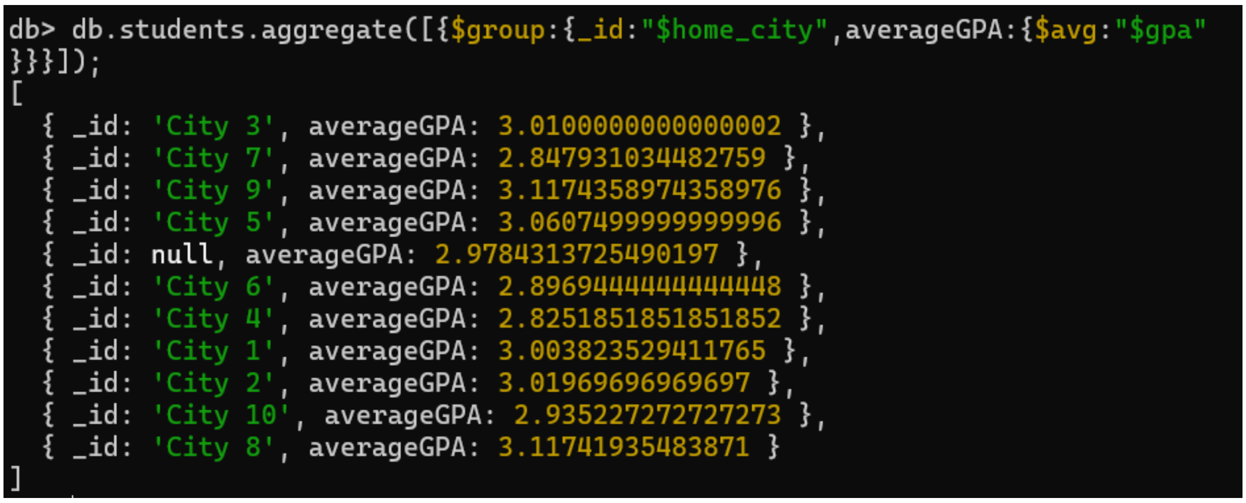


```
db> db.students.aggregate([{$group:{_id:null,minAge:{$min:"$age"},maxAge:{
$max:"$age"}}}]);
[ { _id: null, minAge: 18, maxAge: 25 } ]
```

Explanation:

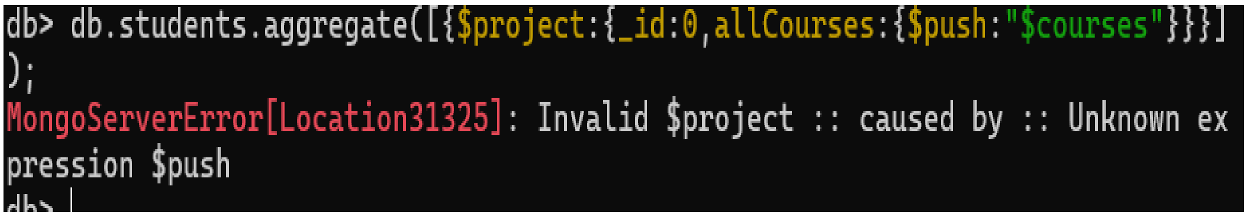
- Similar to the previous example, it uses $group to group all documents.
- minAge: Uses the $min operator to find the minimum value in the "age" field.
- maxAge: Uses the $max operator to find the maximum value in the "age" field.

Example 3:To get averagr GPA of all home cities.



```
db> db.students.aggregate([{$group:{_id:"$home_city",averageGPA:{$avg:"$gpa"
}}}]);
[
  { _id: 'City 3', averageGPA: 3.0100000000000002 },
  { _id: 'City 7', averageGPA: 2.847931034482759 },
  { _id: 'City 9', averageGPA: 3.1174358974358976 },
  { _id: 'City 5', averageGPA: 3.0607499999999996 },
  { _id: null, averageGPA: 2.9784313725490197 },
  { _id: 'City 6', averageGPA: 2.8969444444444448 },
  { _id: 'City 4', averageGPA: 2.8251851851851852 },
  { _id: 'City 1', averageGPA: 3.003823529411765 },
  { _id: 'City 2', averageGPA: 3.01969696969697 },
  { _id: 'City 10', averageGPA: 2.935227272727273 },
  { _id: 'City 8', averageGPA: 3.11741935483871 }
]
```

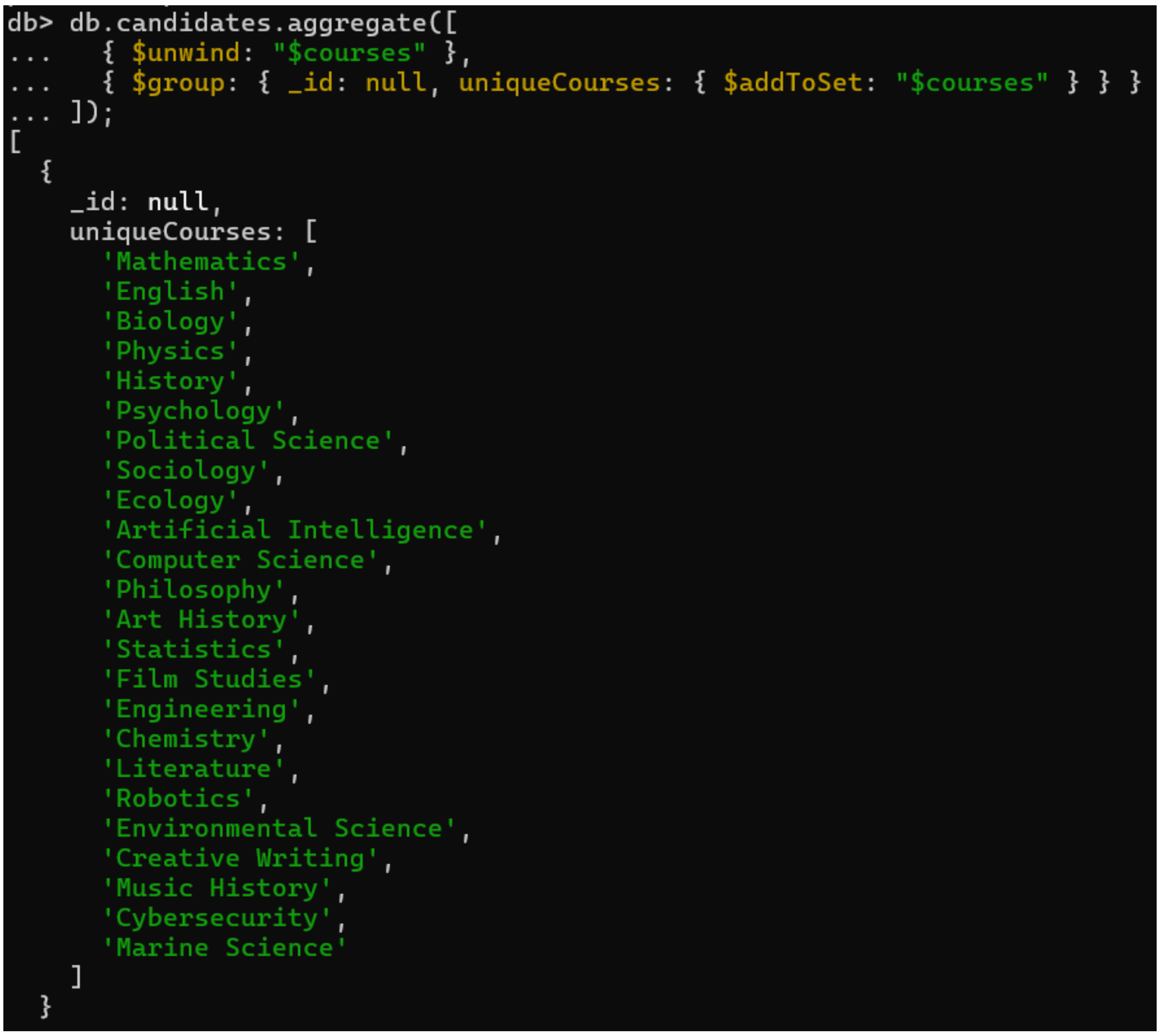
- **Pushing All Courses into a Single Array:**



```
db> db.students.aggregate([{$project:{_id:0,allCourses:{$push:"$courses"}}}]
);
MongoServerError[Location31325]: Invalid $project :: caused by :: Unknown ex
pression $push
db>
```

This is because our Array is incorrect

- **Collect Unique Courses Offered (Using $addToSet):**



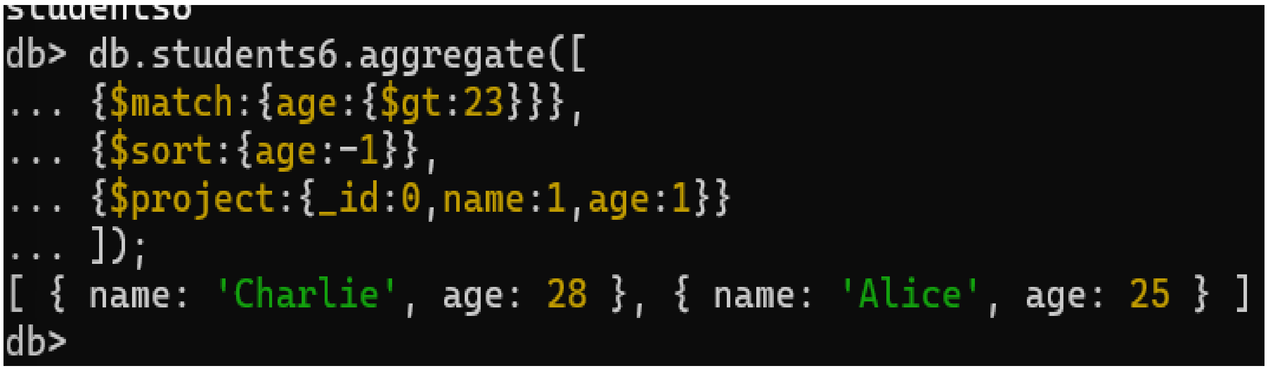
```
db> db.candidates.aggregate([
...   { $unwind: "$courses" },
...   { $group: { _id: null, uniqueCourses: { $addToSet: "$courses" } } }
... ]);
[
  {
    _id: null,
    uniqueCourses: [
      'Mathematics',
      'English',
      'Biology',
      'Physics',
      'History',
      'Psychology',
      'Political Science',
      'Sociology',
      'Ecology',
      'Artificial Intelligence',
      'Computer Science',
      'Philosophy',
      'Art History',
      'Statistics',
      'Film Studies',
      'Engineering',
      'Chemistry',
      'Literature',
      'Robotics',
      'Environmental Science',
      'Creative Writing',
      'Music History',
      'Cybersecurity',
      'Marine Science'
    ]
  }
```

Now all the courses are pushed into single array called “ uniqueCourses”

Class 7: Aggregation pipeline

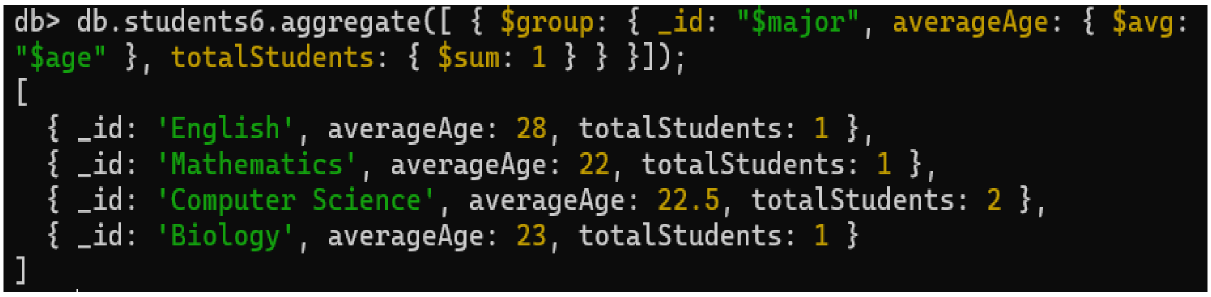
- **Operators:**
  - ✓ $match: Filters documents based on a conditions.
  - ✓ $group: Groups documents by a field and performs aggregations like $avg (average) and $sum (sum).
  - ✓ $sort: Sorts documents in a specified order (ascending or descending).
  - ✓ $project: Selects specific fields to include or exclude in the output documents.
  - ✓ $skip: Skips a certain number of documents from the beginning of the results.
  - ✓ $limit: Limits the number of documents returned.
  - ✓ $unwind: Deconstructs an array into separate documents for each element.

1. **Find students with age greater than 23, sorted by age in descending order, and only return name and age.**



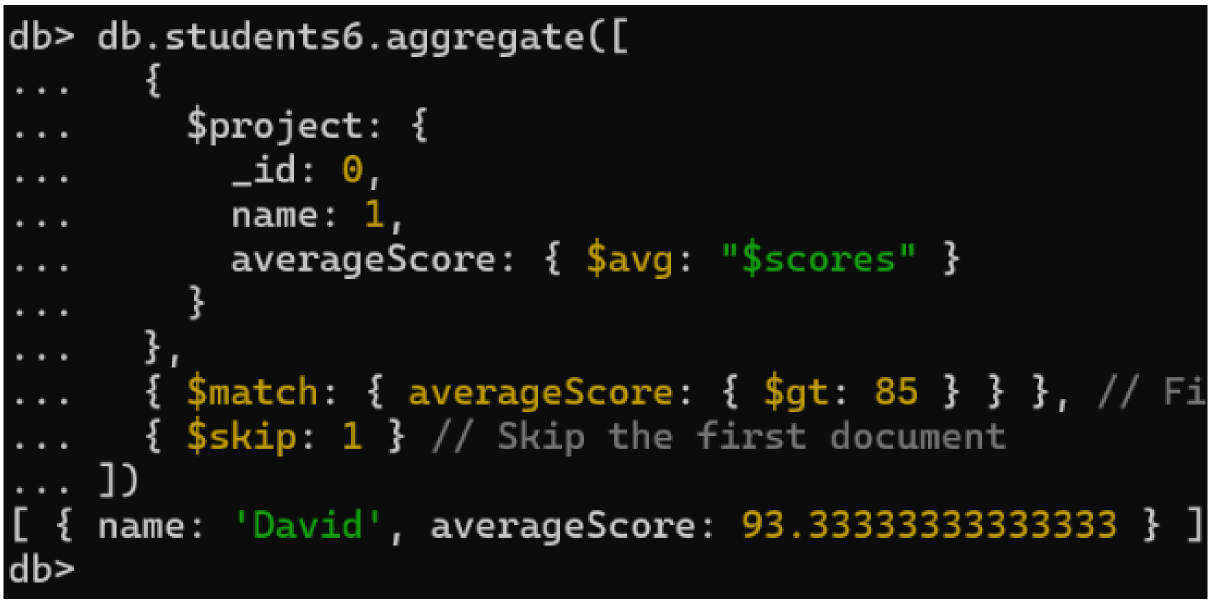
```
students6
db> db.students6.aggregate([
... {$match:{age:{$gt:23}}},
... {$sort:{age:-1}},
... {$project:{_id:0,name:1,age:1}}
... ]);
[ { name: 'Charlie', age: 28 }, { name: 'Alice', age: 25 } ]
db>
```

2. **Group students by major, calculate average age and total number of students in each major:**



```
db> db.students6.aggregate([ { $group: { _id: "$major", averageAge: { $avg:
"$age" }, totalStudents: { $sum: 1 } } }]);
[
  { _id: 'English', averageAge: 28, totalStudents: 1 },
  { _id: 'Mathematics', averageAge: 22, totalStudents: 1 },
  { _id: 'Computer Science', averageAge: 22.5, totalStudents: 2 },
  { _id: 'Biology', averageAge: 23, totalStudents: 1 }
]
```

3. **Find students with an average score (from scores array) above 85 and skip the first document**



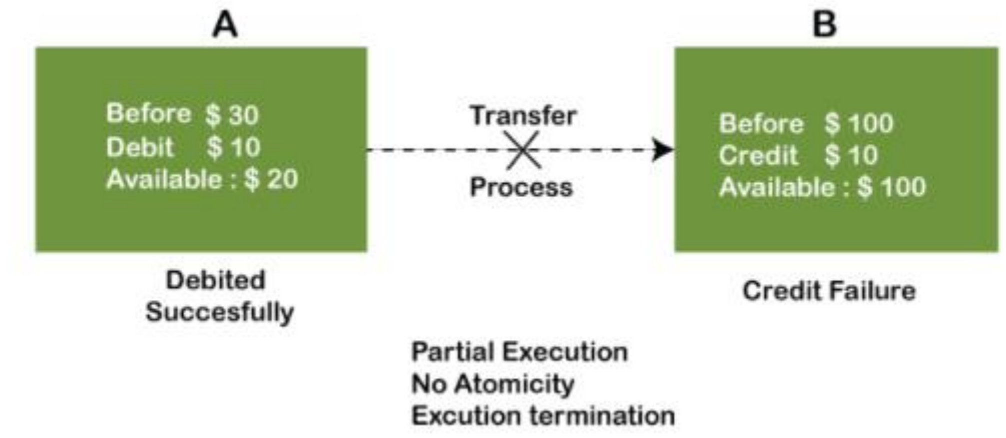
```
db> db.students6.aggregate([
...   {
...     $project: {
...       _id: 0,
...       name: 1,
...       averageScore: { $avg: "$scores" }
...     }
...   },
...   { $match: { averageScore: { $gt: 85 } } }, // Fi
...   { $skip: 1 } // Skip the first document
... ])
[ { name: 'David', averageScore: 93.33333333333333 } ]
db>
```

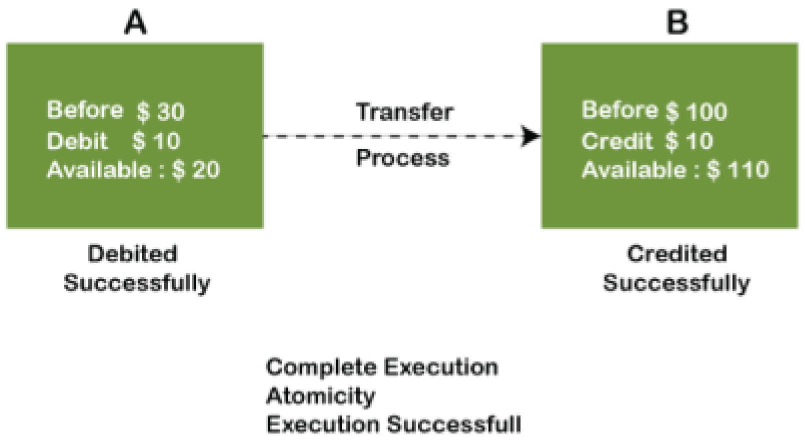
Class 8: ACID & Indexes

➢ CONCEPTS:

- Atomicity
- Consistency
- Replication
- Sharding

1. Atomicity:

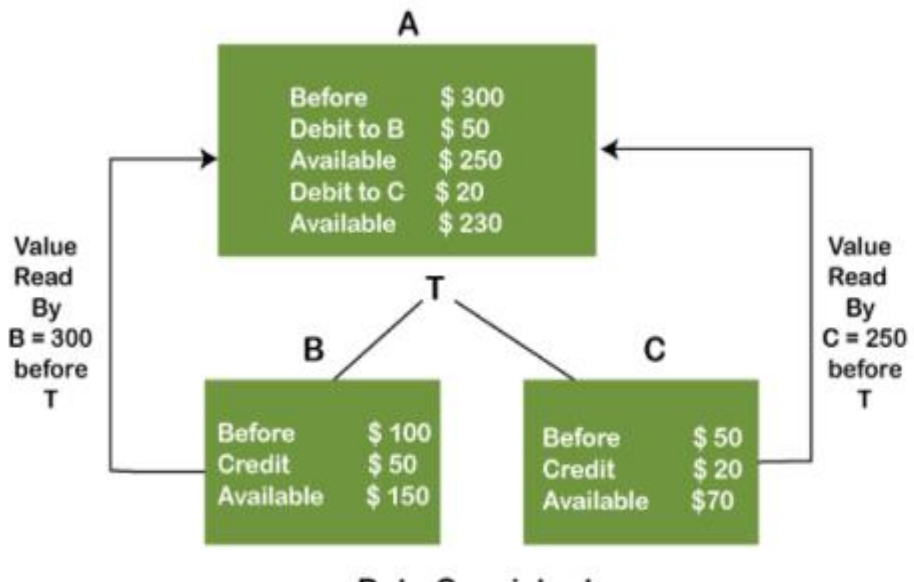




2. Consistency:

The word **consistency** means that the value should remain preserved always. In DBMS, the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

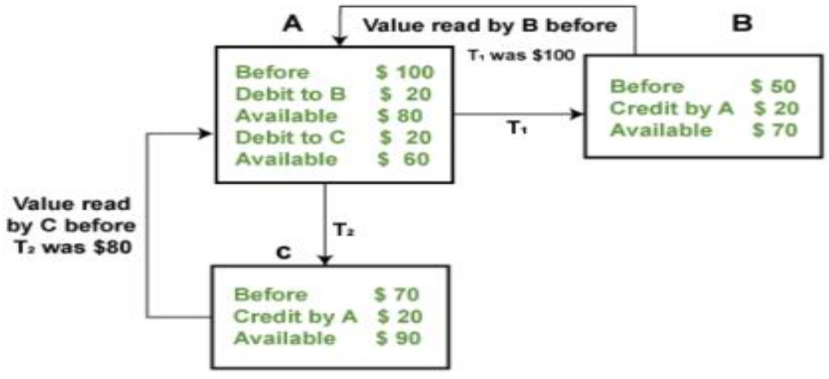
Example:



Data Consistent

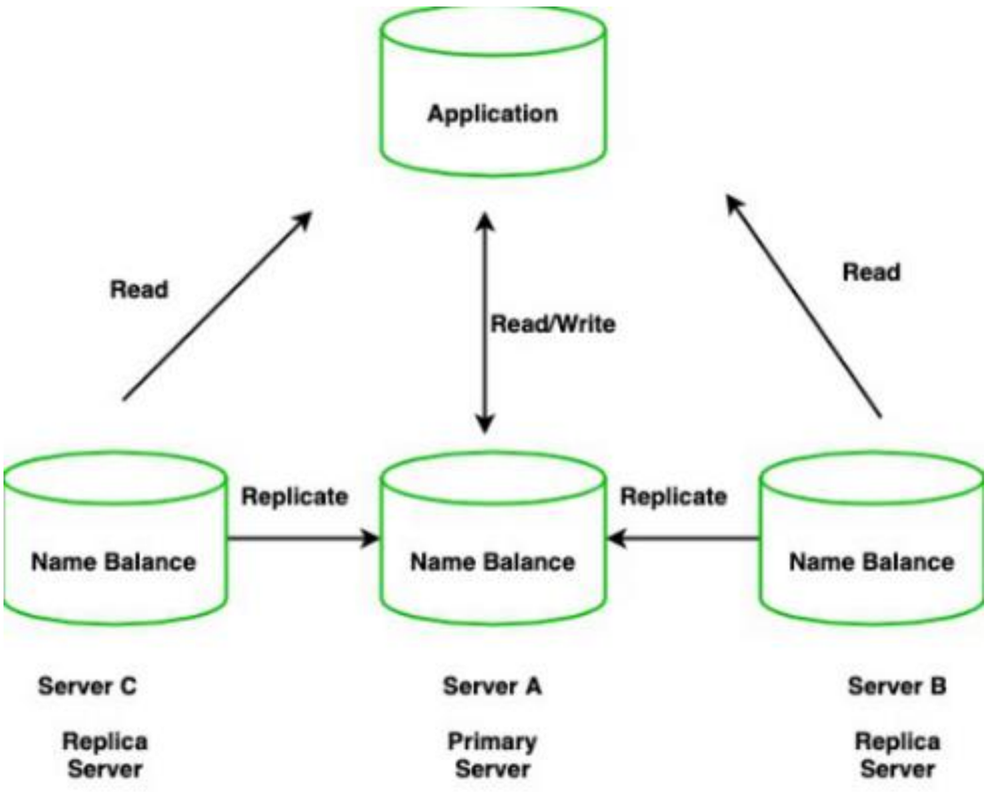
3. Isolation:

Example: If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.

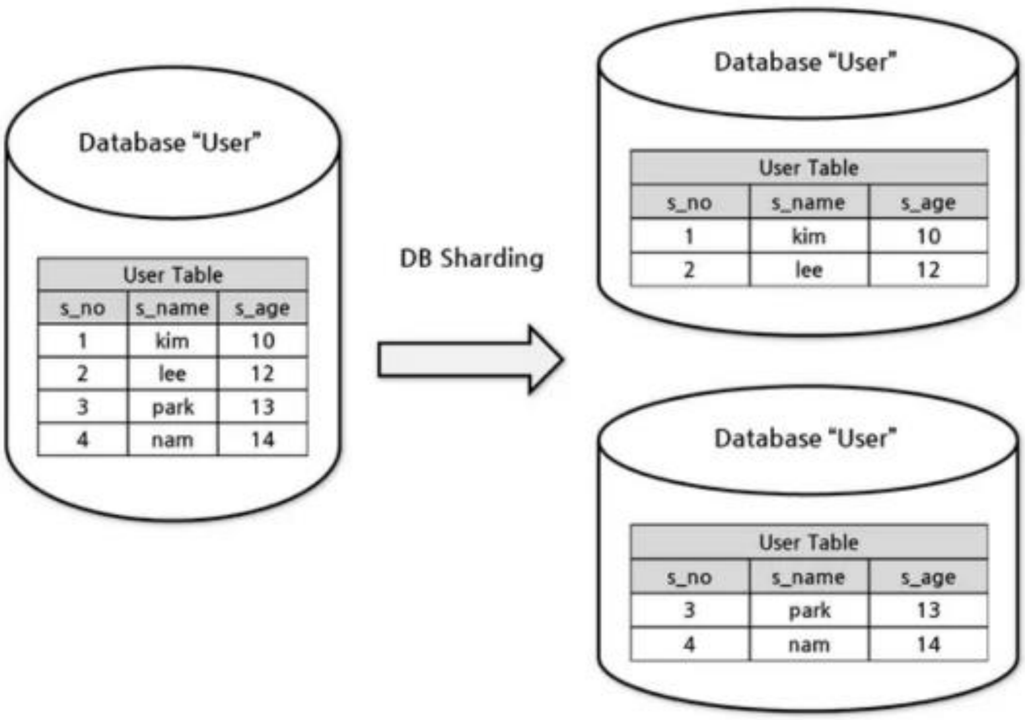


Isolation - Independent execution of T_1 & T_2 by A

4. Replication (Master - Slave):



5. Sharding:



User Table

s_no	s_name	s_age
1	kim	10
2	lee	12
3	park	13
4	nam	14

User Table

s_no	s_name	s_age
1	kim	10
2	lee	12

User Table

s_no	s_name	s_age
3	park	13
4	nam	14