CLASS-3

Where,AND,OR & CRUD

The database name is 'db' and our collection name is 'students'. Here, is the collection below

```
db> db.students.find()
[
  {
    _id: ObjectId('66671b08c474ff0bbecdb16e'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66671b08c474ff0bbecdb16f'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66671b08c474ff0bbecdb170'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66671b08c474ff0bbecdb171'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
  },
```

WHERE:

The where is the parameter of the MongoDB find() method allows us to specify a JavaScript expression that is used to filter the documents returned by the query. This expression can reference the fields of the documents in the collection, as well as any functions defined in the expression. It enables defining custom logic for filtering documents based on field values and relationships between them.
The WHERE clause would allow you to specify the city as the condition, and the database would return only the records that match that criteria. This is an essential feature for working with large datasets, as it enables you to focus on the specific information you need, rather than shifting through the entire dataset.

Comparision Operators:

Name	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$in	Matches any of the values specified in an array.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$nin	Matches none of the values specified in an array.

Example:

If we want to select all documents from the collection "students" that satisfy the condition – to find the students having a GPA of more than 3.5

Syntax+output:

```
db> db.student.find({gpa:{$gt:3.5}});
[
  {
    _id: ObjectId('665de5dd6e26f71bef17d06a'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665de5dd6e26f71bef17d06c'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('665de5dd6e26f71bef17d076'),
    name: 'Student 368',
    age: 20,
    courses: "['English', 'History', 'Physics', 'Computer Science']",
    gpa: 3.91,
    home_city: 'City 9',
    blood_group: 'O-',
    is_hotel_resident: false
  },
```

AND:

In MongoDB, the $and operator is used to perform a logical AND operation on multiple expressions within a query. It retrieves documents that satisfy all the conditions you specify. The AND operator can be implicitly applied when specifying multiple conditions within a single query object.

Combining Conditions: The $and operator in MongoDB allows you to combine multiple conditions in a single query, ensuring that all the specified

criteria are met.

Efficient Indexing: Using $and queries can leverage MongoDB's indexing capabilities to provide fast and efficient data retrieval.

Complex Filtering: The $and operator is particularly useful when you need to apply multiple filters to your data, allowing for more precise and targeted queries.

Example:

Let's say we want the students which follows both the conditions mentioned below.

- Age:21
- Blood group: A+

Syntax+output:

```
db> db.students.find({
... $and: [{ age: 21 },{ blood_group: 'A+'}]});
[
  {
    _id: ObjectId('66671b08c474ff0bbecdb174'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66671b08c474ff0bbecdb180'),
    name: 'Student 647',
    age: 21,
    courses: "['English', 'Physics']",
    gpa: 3.43,
    home_city: 'City 6',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66671b08c474ff0bbecdb276'),
    name: 'Student 411',
    age: 21,
    courses: "['Physics', 'English']",
    gpa: 3.74,
    home_city: 'City 10',
    blood_group: 'A+',
    is_hotel_resident: false
  }
```

OR:

MongoDB provides different types of logical query operators and $or operator is one of them. This operator is used to perform logical OR operation on the array of two or more expressions and select or retrieve only those documents that match at least one of the given expression in the array.

- You can use this operator in methods like find(), update(), etc. according to your requirements.
- You can also use this operator with text queries, GeoSpatial queries, and sort operations.
- When MongoDB evaluates the clauses in the $or expression, it performs a collection scan. Or if all the clauses in $or expression are supported by indexes, then MongoDB performs index scans.
- You can also nest $or operation.

The $or operator allows you to specify multiple conditions, and documents that satisfy at least one of the conditions will be returned.

Example:

Let us say we need to find students whose :
Age is 20 OR Blood group is O+.

```
db> db.students.find({ $or: [{ age: 20 }, { blood_group: 'O+' }] });
[
  {
    _id: ObjectId('66671b08c474ff0bbecdb16e'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66671b08c474ff0bbecdb16f'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66671b08c474ff0bbecdb170'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66671b08c474ff0bbecdb173'),
    name: 'Student 305',
    age: 24,
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']",
    gpa: 3.4,
    home_city: 'City 6',
    blood_group: 'O+',
    is_hotel_resident: true
```

Combining $or with Other Operators:

Example:

Finding all the students who are hostel residents OR have a GPA greater than 3.5.

```
db> db.students.find({ $or: [{ is_hostel_resident: true }, { gpa: {$gt:3.5 }}] });
[
  {
    _id: ObjectId('66671b08c474ff0bbecdb172'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66671b08c474ff0bbecdb174'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66671b08c474ff0bbecdb17e'),
    name: 'Student 368',
    age: 20,
    courses: "['English', 'History', 'Physics', 'Computer Science']",
    gpa: 3.91,
    home_city: 'City 9',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66671b08c474ff0bbecdb185'),
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
```

CURD:

This is applicable to a Collection (Table) or a Document (Row)

C- Create/ Insert

U- Update

R- Remove

D- Delete

Create/Insert:

The "Create" or "Insert" operation in CRUD is used to add new data to a

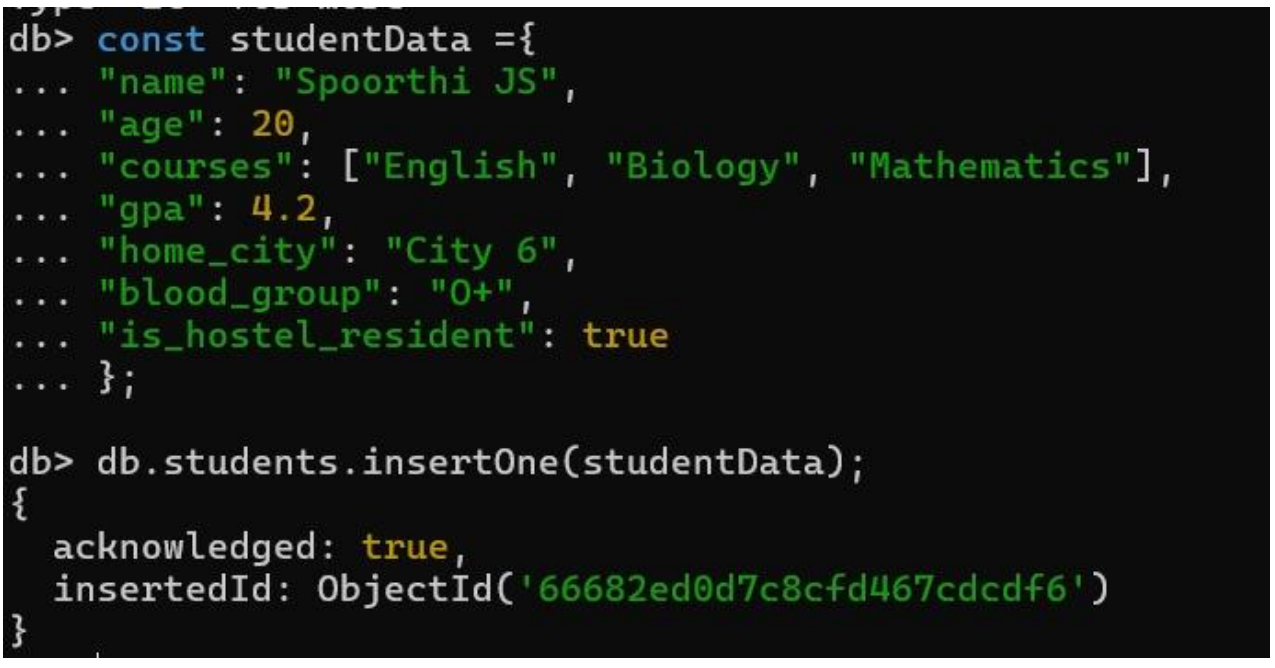
collection or table. This could be creating a new user account, adding a product to an inventory, or inserting a new row of data into a database. The "Create" operation is essential for expanding and populating a data store with new information as needed.

The insert() method in MongoDB inserts documents in the MongoDB collection. This method is also used to create a new collection by inserting documents.

Syntax:

```
db.Collection_name.insertOne(
<document or [document1,
document2,…]>,
{
    writeConcern: <document>,
    ordered:<Boolean>
})
```

Example:



Remove:

The "Remove" operation allows you to retrieve data from a collection or table. This could involve querying the database to find all users with a certain email address or pulling a specific product record.
The remove() method removes documents from the database. It can remove one or all documents from the collection that match the given query expression. If you pass an empty document ({}) in this method, then it will remove all documents from the specified collection. It takes four parameters and returns an object that contains the status of the operation

Syntax:

```
db.Collection_name.remove(
<matching_criteria>,
{
   justOne:<boolean>,
   writeConcern: <document>,
   collation: <document>
 })
```

Example:

```
db> const studentData = { "name": "Spoorthi JS", "age": 20, "courses":
["English", "Biology", "Mathematics"], "gpa": 4.2, "home_city": "City 6
", "blood_group": "O+", "is_hostel_resident": true };

db> db.students.remove(studentData);
{ acknowledged: true, deletedCount: 0 }
```

Update:

The "Update" operation is used to modify existing data in a collection or table. This could include changing a user's contact information, updating a product's price, or editing a record in the database. The "Update" function is essential for maintaining accurate and up-to-date data in your system.
The updateOne() method in MongoDB updates the first matched document within the collection based on the given query.

Syntax:

```
db.collection.updateOne(<filter>,<update> , {
 upsert: <boolean>,
 writeConcern: <document>,
 collation: <document>,
 arrayFilters: [<filterdocument>, ...],
 hint: <document/string>
 })
```

Example:

```
db> db.students.updateOne({ name: 'Student 402'}, {$set: {gpa: 3.45}});

{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

updateMany():

The updateMany() method updates all the documents in MongoDB collections that match the given query. When you update your document, the value of the _id field remains unchanged. This method can also add new fields in the document. Specify an empty document({}) in the selection criteria to update all collection documents.

Example:

```
db> db.students.updateMany({ gpa: { $lt: 3.3}},{ $inc: {gpa: 0.5}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 322,
  modifiedCount: 322,
  upsertedCount: 0
}
```

Delete:

The "Delete" operation allows you to remove data from a collection or table. This could involve deleting a user account, removing a product from inventory, or erasing a specific record from the database. The "Delete" function is important for cleaning up and managing the data in your system as needed.

The deleteOne() method in MongoDB deletes the first document from the collection that matches the given selection criteria. It will delete/remove a single document from the collection. If you use this method in the capped collection, then this method will give a WriteError exception, so to delete documents from the capped collection use the drop() method. If you use this method in the sharded collection, then the query expression passed in this method must contain the sharded key/_id field, if not then this method will give an error. You can also use this method in multi-document transactions.

Syntax:

```
db.Collection_name.deleteOne(
 selection_criteria:<document>,
 {
    writeConcern:<document>,
    collation:<document>,
    hint:<document/string>
})
```

Example:

Delete Many:

In MongoDB, the deleteMany method is used to delete multiple documents from a collection that match a specified filter. This operation is useful when you want to remove all documents that meet certain criteria.

Example: