Operators in MongoDB

1.$exists

The `$exists` operator is used to match documents that contain a specific field, or alternatively, those that do not. It checks if a field is present (`true`) or absent (`false`) in a document.

- Syntax:

```
{ field: { $exists: <boolean> } }
```

-Example:

```
// Find all documents where the "age" field exists.
{ age: { $exists: true } }
```

```
// Find all documents where the "address" field does not exist.
{ address: { $exists: false } }
```

2.$type

The `$type` operator matches documents where the specified field is of a certain BSON data type. You can use either type numbers (1-18) or aliases (e.g., "int", "string").

```
- **Syntax**:


  { field: { $type: <BSON type> } }


  // Find documents where "price" is stored as a double (decimal).
  { price: { $type: "double" } }

  // Find documents where "name" is stored as a string.
  { name: { $type: "string" } }
```

- **Supported Types**:
  - `double` (1)
  - `string` (2)
  - `object` (3)
  - `array` (4)
  - `binary` (5)
  - `bool` (8)
  - `date` (9), etc.

3`$all

The `$all` operator matches documents where an array field contains all specified elements, regardless of their order.

```
- **Syntax**:
  ```javascript
  { arrayField: { $all: [ <value1>, <value2>, ... ] } }
  ```
- **Example**:
  ```javascript
  // Find documents where "tags" contain both "mongodb" and "database".
  { tags: { $all: ["mongodb", "database"] } }
  ``
```
```

4.`$elemMatch`

The `$elemMatch` operator matches documents where at least one element in an array field satisfies multiple conditions.

```
- **Syntax**:
  ```javascript
  { arrayField: { $elemMatch: { <condition1>, <condition2>, ... } } }
  ```
- **Example**:
  ```javascript
  // Find documents where there's a score between 80 and 90.
  { scores: { $elemMatch: { $gt: 80, $lt: 90 } } }
  ```
```

5.`$text`

The `$text` operator performs a text search on fields indexed with a text index, allowing for search capabilities like full-text search.

```
- **Syntax**:
  ```javascript
  { $text: { $search: "<string>" } }
  ```
- **Example**:
  ```javascript
  // Find documents containing the term "database".
  { $text: { $search: "database" } }

  // Find documents that must include "database" and optionally
"system".
  { $text: { $search: "database system" } }
  ```
```

6.$push

The `$push` operator appends a specified value to an array field in a document. If the field doesn't exist, `$push` will create it.

```
- **Syntax**:
  ```javascript
  { $push: { arrayField: <value> } }
  ```
- **Example**:
  ```javascript
  // Add "new_tag" to the "tags" array field.
  { $push: { tags: "new_tag" } }
  ```
```

7.$pop

The `$pop` operator removes the first or last element of an array. It takes either `-1` (first element) or `1` (last element) as an argument.

```
- **Syntax**:
  ```javascript
  { $pop: { arrayField: <1 | -1> } }
  ```
- **Example**:
  ```javascript
  // Remove the last element from the "tags" array.
  { $pop: { tags: 1 } }

  // Remove the first element from the "tags" array.
  { $pop: { tags: -1 } }
  ```
```

8.$sort

The `$sort` operator orders documents in a specified direction. It can be used in aggregation pipelines or with the `find()` method.

````
- **Syntax**:
  ```javascript
  { $sort: { field1: <1 | -1>, field2: <1 | -1>, ... } }
  ```
- **Example**:
  ```javascript
  // Sort documents by "age" ascending, and "name" descending.
  { $sort: { age: 1, name: -1 } }
  ```
````

9.$limit

The `$limit` operator restricts the number of documents passed to the next stage of the aggregation pipeline or returned by a query.

````
- **Syntax**:
  ```javascript
  { $limit: <number> }
  ```
- **Example**:
  ```javascript
  // Limit the result set to 5 documents.
  { $limit: 5 }
  ```
````

10.$project

The `$project` operator shapes the output by including or excluding specific fields, adding computed fields, or renaming fields in aggregation pipelines.

````
- **Syntax**:
  ```javascript
  { $project: { field1: <0 | 1>, field2: <0 | 1>, newField: <expression>
} }
  ```
- **Example**:
  ```javascript
  // Project only "name" and "age", and create a new field
"year_of_birth".
  {
    $project: {
      name: 1,
      age: 1,
      year_of_birth: { $subtract: [2024, "$age"] }
    }
  }
  ```
````

Comparison Operators

Comparison operators are used to compare field values in documents with specified values. All comparison operators are prefixed with `$`.

1. Equality: $eq

- Matches documents where the value of a field is equal to a specified value.
- Syntax: `{ field: { $eq: value } }`
- Example: `{ age: { $eq: 25 } }`
  Finds documents where `age` is equal to 25.

---

2. Not Equal: $ne

- Matches documents where the value of a field is not equal to a specified value.
- Syntax: `{ field: { $ne: value } }`
- Example: `{ age: { $ne: 25 } }`
  Finds documents where `age` is not 25.

---

3. Greater Than: $gt

- Matches documents where the value of a field is greater than a specified value.

- Syntax: `{ field: { $gt: value } }`
- Example: `{ age: { $gt: 25 } }`
  Finds documents where `age` is greater than 25.

---

4. Greater Than or Equal To: $gte

- Matches documents where the value of a field is greater than or equal to a specified value.
- Syntax: `{ field: { $gte: value } }`
- Example: `{ age: { $gte: 25 } }`
  Finds documents where `age` is 25 or more.

---

5. Less Than: $lt

- Matches documents where the value of a field is less than a specified value.
- Syntax: `{ field: { $lt: value } }`
- Example: `{ age: { $lt: 25 } }`
  Finds documents where `age` is less than 25.

---

6. Less Than or Equal To: $lte

- Matches documents where the value of a field is less than or equal to a specified value.
- Syntax: `{ field: { $lte: value } }`

- Example: `{ age: { $lte: 25 } }`
  Finds documents where `age` is 25 or less.

---

7. In: $in

- Matches documents where the value of a field equals any value in the specified array.
- Syntax: `{ field: { $in: [value1, value2, ...] } }`
- Example: `{ age: { $in: [20, 25, 30] } }`
  Finds documents where `age` is 20, 25, or 30.

---

8. Not In: $nin

- Matches documents where the value of a field does not equal any value in the specified array.
- Syntax: `{ field: { $nin: [value1, value2, ...] } }`
- Example: `{ age: { $nin: [20, 25, 30] } }`
  Finds documents where `age` is not 20, 25, or 30.

---

9. Exists: $exists

- Matches documents where a field exists or does not exist.
- Syntax: `{ field: { $exists: true | false } }`

- Example: `{ phone: { $exists: true } }`
  Finds documents where the `phone` field exists.

Logical Operators

Logical operators combine multiple conditions or expressions. All logical operators are prefixed with `$`.

1. AND: $and

- Matches documents that satisfy all the conditions in the array.
- Syntax: `{ $and: [ { condition1 }, { condition2 }, ... ] }`
- Example: `{ $and: [ { age: { $gt: 20 } }, { age: { $lt: 30 } } ] }`
  Finds documents where `age` is between 21 and 29.

---

2. OR: $or

- Matches documents that satisfy at least one condition in the array.
- Syntax: `{ $or: [ { condition1 }, { condition2 }, ... ] }`
- Example: `{ $or: [ { age: { $lt: 20 } }, { age: { $gt: 30 } } ] }`
  Finds documents where `age` is less than 20 or greater than 30.

---

3. NOT: $not

- Matches documents that do not satisfy a specified condition.
- Syntax: `{ field: { $not: { condition } } }`
- Example: `{ age: { $not: { $gte: 25 } } }`
  Finds documents where `age` is less than 25.

---

4. NOR: $nor

- Matches documents that do not satisfy any of the conditions in the array.
- Syntax: `{ $nor: [ { condition1 }, { condition2 }, ... ] }`
- Example: `{ $nor: [ { age: { $lt: 20 } }, { age: { $gt: 30 } } ] }`
  Finds documents where age is between 20 and 30 (inclusive).