MongoDB Operators

Operators for Update Operations

MongoDB provides update operators for modifying documents.

Field Update Operators

- **`$set`**: Sets the value of a field.
  Example: `{ $set: { age: 25 } }`
- **`$unset`**: Removes a field from a document.
  Example: `{ $unset: { age: "" } }`
- **`$inc`**: Increments a field by a specified value.
  Example: `{ $inc: { score: 5 } }`
- **`$mul`**: Multiplies the value of a field by a specified value.
  Example: `{ $mul: { price: 1.1 } }`
- **`$rename`**: Renames a field.
  Example: `{ $rename: { oldName: "newName" } }`
- **`$min`**: Updates the field only if the specified value is less than the current value.
  Example: `{ $min: { price: 50 } }`
- **`$max`**: Updates the field only if the specified value is greater than the current value.
  Example: `{ $max: { age: 40 } }`
- **`$currentDate`**: Sets the field to the current date.
  Example: `{ $currentDate: { lastUpdated: true } }`
- **`$addToSet`**: Adds a value to an array only if it doesn't already exist.
  Example: `{ $addToSet: { tags: "newTag" } }`
- **`$push`**: Appends a value to an array.
  Example: `{ $push: { comments: "Great!" } }`
- **`$pop`**: Removes the first or last element of an array.
  Example: `{ $pop: { comments: -1 } } // Removes first element`
- **`$pull`**: Removes all array elements that match a condition.
  Example: `{ $pull: { scores: { $lt: 50 } } }`
- **`$pullAll`**: Removes multiple specified values from an array.
  Example: `{ $pullAll: { tags: ["tag1", "tag2"] } }`
- **`$each`**: Used with `$push` or `$addToSet` to add multiple elements.
  Example: `{ $push: { comments: { $each: ["Nice", "Great!"] } } }`

- **`$position`**: Specifies the position to insert elements in an array (used with `$push`).
  Example: `{ $push: { comments: { $each: ["Hello"], $position: 1 } } }`

Operators for Find Queries

MongoDB provides query operators for filtering documents.

Comparison Operators

- **`$eq`**: Matches documents where the field equals the specified value.
  Example: `{ age: { $eq: 25 } }`
- **`$ne`**: Matches documents where the field does not equal the specified value.
  Example: `{ age: { $ne: 25 } }`
- **`$gt`**: Matches documents where the field is greater than the specified value.
  Example: `{ age: { $gt: 25 } }`
- **`$gte`**: Matches documents where the field is greater than or equal to the specified value.
  Example: `{ age: { $gte: 25 } }`
- **`$lt`**: Matches documents where the field is less than the specified value.
  Example: `{ age: { $lt: 25 } }`
- **`$lte`**: Matches documents where the field is less than or equal to the specified value.
  Example: `{ age: { $lte: 25 } }`
- **`$in`**: Matches documents where the field's value is in a specified array.
  Example: `{ age: { $in: [20, 25, 30] } }`
- **`$nin`**: Matches documents where the field's value is not in a specified array.
  Example: `{ age: { $nin: [20, 25, 30] } }`

Logical Operators

- **`$and`**: Matches documents that satisfy all conditions.
  Example: `{ $and: [{ age: { $gt: 20 } }, { age: { $lt: 30 } }] }`
- **`$or`**: Matches documents that satisfy at least one condition.
  Example: `{ $or: [{ age: { $lt: 20 } }, { age: { $gt: 30 } }] }`
- **`$not`**: Matches documents that do not match the condition.
  Example: `{ age: { $not: { $gte: 30 } } }`
- **`$nor`**: Matches documents that do not satisfy any of the conditions.
  Example: `{ $nor: [{ age: { $gt: 30 } }, { status: "Active" }] }`

Element Operators

- `$exists`: Matches documents that have the specified field.
  Example: `{ email: { $exists: true } }`
- `$type`: Matches documents where the field is of a specified type.
  Example: `{ age: { $type: "int" } }`

Array Operators

- `$all`: Matches documents where the array contains all specified elements.
  Example: `{ tags: { $all: ["tag1", "tag2"] } }`
- `$elemMatch`: Matches documents where at least one array element matches all specified conditions.
  Example: `{ scores: { $elemMatch: { $gt: 80, $lt: 90 } } }`
- `$size`: Matches documents where the array has a specified length.
  Example: `{ tags: { $size: 3 } }`

Evaluation Operators

- `$regex`: Matches strings using regular expressions.
  Example: `{ name: { $regex: /^A/ } }`
- `$expr`: Allows the use of aggregation expressions in queries.
  Example: `{ $expr: { $gt: ["$field1", "$field2"] } }`
- `$text`: Performs text search on indexed fields.
  Example: `{ $text: { $search: "hello world" } }`
- `$where`: Matches documents that satisfy a JavaScript expression.
  Example: `{ $where: "this.age > 25" }`

MongoDB Aggregation Pipeline Operators

The aggregation pipeline processes data by passing documents through a series of stages, where each stage performs specific operations. Here are some commonly used aggregation operators grouped by their purpose:

---

1. Filtering Operators

Used to filter documents in the pipeline.

- `$match`: Filters documents based on a condition (similar to `find`).

```
db.collection.aggregate([
  { $match: { status: "active" } }
]);
```

---

2. Projection Operators

Used to shape the structure of the output documents.

- `$project`: Specifies the fields to include or exclude in the output.

```
db.collection.aggregate([
  { $project: { name: 1, age: 1, _id: 0 } }
]);
```

- `$addFields`: Adds new fields or modifies existing ones.

```
db.collection.aggregate([
  { $addFields: { fullName: { $concat:
   ["$firstName", " ", "$lastName"] } } }
]);
```

- `$unset`: Removes specified fields.

```
db.collection.aggregate([
  { $unset: "unnecessaryField" }
]);
```

3. Grouping and Sorting Operators

Used to group and arrange documents.

- `$group`: Groups documents by a specified key and performs aggregations (e.g., sum, count).

```
db.collection.aggregate([
  { $group: { _id: "$category",
  total: { $sum: "$price" } } }
]);
```

- `$sort`: Sorts documents in ascending (1) or descending (-1) order.

```
db.collection.aggregate([
  { $sort: { age: -1 } }
]);
```

---

4. Array Operators

Used to handle array fields.

- `$unwind`: Deconstructs an array field into multiple documents (one per array element).

```
db.collection.aggregate([
  { $unwind: "$tags" }
]);
```

---

5. Lookup and Joining Operators

Used to join collections.

- `$lookup`: Performs a left outer join with another collection.

```
db.collection.aggregate([
  {
    $lookup: {
      from: "orders",
      localField: "_id",
      foreignField: "customerId",
      as: "orders"
    }
  }
]);
```

6. Conditional Operators

Used to apply conditional logic.

- `$cond`: Evaluates a condition and returns a value based on true or false.

```
db.collection.aggregate([
  { $project: { isAdult: { $cond:
  { if: { $gte: ["$age", 18] },
  then: true, else: false } } } }
]);
```

- `$ifNull`: Replaces `null` or missing values with a specified value.

```
db.collection.aggregate([
  { $project: { name:
  { $ifNull: ["$name", "Unknown"] } } }
]);
```

---

7. Accumulator Operators

Used within `$group` or `$project` stages to perform calculations.

- **`$sum`**: Calculates the sum of numeric values.
- **`$avg`**: Calculates the average value.
- **`$min`**: Finds the minimum value.
- **`$max`**: Finds the maximum value.
- **`$count`**: Counts the number of documents.

```
db.collection.aggregate([
  { $group: { _id: "$category",
  totalCount: { $count: {} } } }
]);
```

---

8. Miscellaneous Operators

Used for additional functionality.

- **`$limit`**: Limits the number of documents in the output.

```
db.collection.aggregate([
  { $limit: 5 }
]);
```

- **`$skip`**: Skips a specified number of documents.

```
db.collection.aggregate([
  { $skip: 10 }
]);
```

- **`$sample`**: Selects random documents.

```
db.collection.aggregate([
  { $sample: { size: 3 } }
]);
```