MongoDB Operators Notes

MongoDB operators are used to query, update, and aggregate data. They are prefixed with $ and passed as JavaScript objects when interacting with MongoDB via its driver (e.g., Node.js). Operators are categorized into query operators, update operators, and aggregation operators.

---

1. Query Operators

Used in .find() or $match to filter documents based on conditions.

Comparison Operators

Match documents based on value comparisons.

- **$eq**
  Matches values equal to a specified value.
  - Syntax: {
    field: {
    $eq: value
    }
    }
  - Example: { age: { $eq: 25 } } → Matches documents where age is 25.
- **$gt**
  Greater than.
  - Syntax: { field: { $gt: value } }
  - Example: { age: { $gt: 25 } } → Matches documents where age > 25.
- **$gte**
  Greater than or equal to.
  - Syntax: { field: { $gte: value } }
  - Example: { age: { $gte: 25 } } → Matches documents where age ≥ 25.
- **$lt**
  Less than.
  - Syntax: { field: { $lt: value } }
  - Example: { age: { $lt: 25 } } → Matches documents where age < 25.
- **$lte**
  Less than or equal to.

    - Syntax: { field: { $lte: value } }
    - Example: { age: { $lte: 25 } } → Matches documents where age ≤ 25.

- **$ne**
  Not equal to.
    - Syntax: { field: { $ne: value } }
    - Example: { age: { $ne: 25 } } → Matches documents where age ≠ 25.

- **$in**
  Matches any value in an array.
    - Syntax: { field: { $in: [value1, value2, ...] } }
    - Example: { city: { $in: ["New York", "London"] } } → Matches documents where city is "New York" or "London".

- **$nin**
  Matches none of the values in an array.
    - Syntax: { field: { $nin: [value1, value2, ...] } }
    - Example: { city: { $nin: ["New York", "London"] } } → Matches documents where city is not "New York" or "London".

Logical Operators

Combine conditions for complex queries.

- **$and**
  All conditions must be true.
    - Syntax: { $and: [{ condition1 }, { condition2 }] }
    - Example: { $and: [{ age: { $gt: 25 } }, { city: "New York" }] } → Matches documents where age > 25 and city is "New York".

- **$or**
  At least one condition must be true.
    - Syntax: { $or: [{ condition1 }, { condition2 }] }
    - Example: { $or: [{ age: { $gt: 25 } }, { city: "New York" }] } → Matches documents where age > 25 or city is "New York".

- **$not**
  Inverts a condition.
    - Syntax: { field: { $not: { condition } } }
    - Example: { age: { $not: { $gt: 25 } } } → Matches documents where age ≤ 25.

- **$nor**
  None of the conditions are true.
  - Syntax: { $nor: [{ condition1 }, { condition2 }] }
  - Example: { $nor: [{ age: { $gt: 25 } }, { city: "New York" }] } → Matches documents where age ≤ 25 and city is not "New York".

Element Operators

Check for the existence or type of fields.

- **$exists**
  Matches documents with a specified field.
  - Syntax: { field: { $exists: true/false } }
  - Example: { age: { $exists: true } } → Matches documents with an age field.
- **$type**
  Matches documents where a field is of a specified type.
  - Syntax: { field: { $type: "type" } } (e.g., "string", "number").
  - Example: { age: { $type: "number" } } → Matches documents where age is a number.

Array Operators

Work with array fields.

- **$all**
  Matches arrays containing all specified elements.
  - Syntax: { field: { $all: [value1, value2, ...] } }
  - Example: { tags: { $all: ["tech", "gadgets"] } } → Matches documents where tags contains both "tech" and "gadgets".
- **$elemMatch**
  Matches documents where an array field satisfies multiple conditions.
  - Syntax: { field: { $elemMatch: { condition1, condition2 } } }
  - Example: { scores: { $elemMatch: { $gte: 80, $lte: 90 } } } → Matches documents where scores has at least one element between 80 and 90.
- **$size**
  Matches arrays of a specific length.
  - Syntax: { field: { $size: length } }
  - Example: { tags: { $size: 2 } } → Matches documents where tags has exactly 2 elements.

2. Update Operators

Used in .updateOne(), .updateMany(), etc., to modify documents.

Field Operators

Modify individual fields.

- **$set**
  Sets the value of a field (creates it if it doesn't exist).
  - Syntax: { $set: { field: value } }
  - Example: { $set: { age: 30 } } → Sets age to 30.
- **$unset**
  Removes a field.
  - Syntax: { $unset: { field: "" } }
  - Example: { $unset: { age: "" } } → Removes the age field.
- **$inc**
  Increments a numeric field by a value.
  - Syntax: { $inc: { field: value } }
  - Example: { $inc: { age: 1 } } → Increases age by 1.
- **$mul**
  Multiplies a numeric field by a value.
  - Syntax: { $mul: { field: value } }
  - Example: { $mul: { price: 2 } } → Doubles the price.
- **$rename**
  Renames a field.
  - Syntax: { $rename: { oldField: "newField" } }
  - Example: { $rename: { age: "userAge" } } → Renames age to userAge.
- **$setOnInsert**
  Sets a field only during an insert (used with upsert).
  - Syntax: { $setOnInsert: { field: value } }
  - Example: { $setOnInsert: { createdAt: new Date() } } → Sets createdAt only if the document is inserted.

Array Operators

Modify array fields.

- **$push**
  Adds an element to an array.
  - Syntax: { $push: { field: value } }
  - Example: { $push: { tags: "tech" } } → Adds "tech" to tags.
- **$pop**
  Removes the first (-1) or last (1) element of an array.
  - Syntax: { $pop: { field: 1/-1 } }
  - Example: { $pop: { tags: 1 } } → Removes the last element from tags.
- **$pull**
  Removes elements from an array that match a condition.
  - Syntax: { $pull: { field: condition } }
  - Example: { $pull: { tags: "tech" } } → Removes "tech" from tags.
- **$addToSet**
  Adds an element to an array only if it doesn't exist.
  - Syntax: { $addToSet: { field: value } }
  - Example: { $addToSet: { tags: "tech" } } → Adds "tech" to tags if not already present.

Array Modifiers

Enhance array operations.

- **$each**
  Modifies multiple values in $push or $addToSet.
  - Syntax: { $push: { field: { $each: [value1, value2] } } }
  - Example: { $push: { tags: { $each: ["tech", "gadgets"] } } } → Adds both "tech" and "gadgets".
- **$position**
  Specifies where to insert elements in $push.
  - Syntax: { $push: { field: { $each: [values], $position: index } } }
  - Example: { $push: { tags: { $each: ["tech"], $position: 0 } } } → Adds "tech" at the start.
- **$sort**
  Sorts an array after modification.

- Syntax: { $push: { field: { $each: [], $sort: 1/-1 } } }
- Example: { $push: { scores: { $each: [85], $sort: 1 } } } → Adds 85 and sorts scores ascending.

---

3. Aggregation Pipeline Operators

Used in .aggregate() to process and transform data.

Stage Operators

Define stages in the pipeline.

- **$match**
  Filters documents.
  - Syntax: { $match: { condition } }
  - Example: { $match: { age: { $gte: 25 } } } → Filters documents where age ≥ 25.
- **$group**
  Groups documents and computes aggregates.
  - Syntax: { $group: { _id: "groupByField", field: { $operation: "field" } } }
  - Example: { $group: { _id: "$city", total: { $sum: "$sales" } } } → Groups by city and sums sales.
- **$project**
  Reshapes documents (include/exclude/transform fields).
  - Syntax: { $project: { field: 1/0, newField: expression } }
  - Example: { $project: { name: 1, age: 0 } } → Includes name, excludes age.
- **$sort**
  Sorts documents.
  - Syntax: { $sort: { field: 1/-1 } }
  - Example: { $sort: { age: -1 } } → Sorts by age descending.
- **$limit**
  Limits the number of documents.
  - Syntax: { $limit: number }
  - Example: { $limit: 5 } } → Returns only 5 documents.
- **$skip**
  Skips a number of documents.

    - Syntax: { $skip: number }
    - Example: { $skip: 10 } } → Skips the first 10 documents.
- **$unwind**
  Expands an array into separate documents.
    - Syntax: { $unwind: "$field" }
    - Example: { $unwind: "$tags" } } → Creates a document for each element in tags.
- **$set**
  Adds or modifies fields.
    - Syntax: { $set: { newField: expression } }
    - Example: { $set: { fullName: { $concat: ["$firstName", " ", "$lastName"] } } } → Creates fullName.
- **$unset**
  Removes fields.
    - Syntax: { $unset: "field" }
    - Example: { $unset: "age" } } → Removes age.

Expression Operators

Used within stages for computations.

- Arithmetic
    - **$add:** Adds values.
    - **$subtract:** Subtracts values.
    - **$multiply:** Multiplies values.
    - **$divide:** Divides values.
    - **$mod:** Returns the remainder.
    - Example: { $project: { total: { $add: ["$price", "$tax"] } } } → Adds price and tax.
- **Aggregation**
    - **$sum:** Sums values.
    - **$avg:** Computes the average.
    - **$min:** Finds the minimum.
    - **$max:** Finds the maximum.

    - Example: { $group: { _id: "$city", avgSales: { $avg: "$sales" } } } → Averages sales by city.

- **String**
    - **$concat:** Concatenates strings.
    - **$substr:** Extracts a substring.
    - **$toLower:** Converts to lowercase.
    - **$toUpper:** Converts to uppercase.
    - **$split:** Splits a string into an array.
    - Example: { $project: { name: { $concat: ["$firstName", " ", "$lastName"] } } } → Concatenates names.
- **Array**
    - **$arrayElemAt:** Returns an element at a specific index.
    - **$concatArrays:** Combines arrays.
    - **$filter:** Filters an array.
    - **$map**: Applies an expression to each array element.
    - **$size:** Returns the array length.
    - Example: { $project: { firstTag: { $arrayElemAt: ["$tags", 0] } } } → Gets the first tag.
- **Conditional**
    - **$ifNull:** Returns a fallback value if null.
    - **$cond:** Evaluates a conditional expression.
    - **$switch:** Evaluates multiple conditions.
    - Example: { $project: { status: { $cond: { if: { $gte: ["$age", 18] }, then: "Adult", else: "Minor" } } } } → Sets status based on age.

Key Points

- Operators are prefixed with $ and used as JavaScript object properties when interacting with MongoDB.
- Some operators (e.g., $set, $or) appear in multiple contexts but may behave differently.
- Aggregation pipelines combine stages and expressions for advanced data processing.

- Refer to MongoDB documentation for version-specific features or additional operators.
