1. Accumulator Operators (For Grouping & Aggregation)

Accumulator operators are used with `$group` to perform calculations on grouped data.

1.1 `$sum` - Sum of Values

Finds the total sum of a numeric field.
Example:

```
db.orders.aggregate([
  { $group: { _id: "$customerId", totalAmount: { $sum: "$amount" } } }
]);
```

💡 **Function:** Adds up numeric values.

1.2 `$avg` - Average Value

Finds the average of numeric values.
Example:

```
db.orders.aggregate([
  { $group: { _id: "$category", avgPrice: { $avg: "$price" }}}
]);
```

💡 **Function:** Computes the mean value.

1.3 `$min` - Minimum Value

Finds the smallest value.
Example:

```
db.orders.aggregate([
  { $group: { _id: "$category", minPrice: { $min: "$price" }}}
]);
```

💡 **Function:** Returns the lowest value.

1.4 `$max` - Maximum Value

Finds the largest value.
Example:

```
db.orders.aggregate([
  { $group: { _id: "$category", maxPrice: { $max: "$price" } } }
]);
```

💡 **Function:** Returns the highest value.

1.5 `$first` - First Document

Returns the first document in each group.
Example:

```
db.orders.aggregate([
  { $sort: { orderDate: 1 } },
  { $group: { _id: "$customerId", firstOrder: { $first: "$orderDate" } } }
]);
```

💡 **Function:** Returns the first value based on sorting.

1.6 `$last` - Last Document

Returns the last document in each group.
Example:

```
db.orders.aggregate([
  { $sort: { orderDate: 1 } },
  { $group: { _id: "$customerId", lastOrder: { $last: "$orderDate" } } }
]);
```

💡 **Function:** Returns the last value based on sorting.

1.7 `$push` - Push Values into an Array

Stores multiple values in an array.
Example:

```
db.orders.aggregate([
  { $group: { _id: "$customerId", orderList: { $push:
"$orderId" } } }
]);
```

💡 **Function:** Collects all values in an array.

1.8 `$addToSet` - Unique Values in an Array

Adds only unique values to an array.
Example:

```
db.orders.aggregate([
  { $group: { _id: "$customerId", uniqueProducts: { $addToSet:
"$productId" } } }
]);
```

💡 **Function:** Prevents duplicate values.

1.9 `$count` - Count Documents

Counts the number of documents in a pipeline.
Example:

```
db.orders.aggregate([
  { $match: { status: "completed" } },
  { $count: "completedOrders" }
]);
```

💡 **Function:** Returns the count of documents.

---

2. Arithmetic Operators

Used to perform mathematical calculations.

2.1 `$add` - Addition

Example:

```
db.orders.aggregate([
  { $project: { totalWithTax: { $add: ["$price", 5] } } }]);
```

💡 **Function:** Adds numbers.

2.2 `$subtract` - Subtraction

Example:

```
db.orders.aggregate([
  { $project: { discount: { $subtract: ["$price",
"$discountAmount"] } } }
]);
```

💡 **Function:** Subtracts numbers.

2.3 `$multiply` - Multiplication

Example:

```
db.orders.aggregate([
  { $project: { discountedPrice: { $multiply: ["$price", 0.9]
} } }
]);
```

💡 **Function:** Multiplies numbers.

2.4 `$divide` - Division

Example:

```
db.orders.aggregate([
  { $project: { pricePerUnit: { $divide: ["$totalPrice",
"$quantity"] } } } ]);
```

💡 **Function:** Divides numbers.

2.5 `$mod` - Modulus (Remainder)

Example:

```
db.orders.aggregate([
  { $project: { remainder: { $mod: ["$totalPrice", 2] } } }
]);
```

💡 **Function:** Returns remainder after division.

---

3. Comparison Operators

Used for comparing values.

3.1 `$eq` - Equal to

Example:

```
db.orders.aggregate([
  { $match: { status: { $eq: "completed" } } }
]);
```

💡 **Function:** Matches exact values.

3.2 `$ne` - Not Equal to

Example:

```
db.orders.aggregate([
  { $match: { status: { $ne: "pending" } } }
]);
```

💡 **Function:** Excludes specified value.

3.3 `$gt` - Greater than

Example:

```
db.orders.aggregate([
  { $match: { price: { $gt: 100 } } }
]);
```

💡 **Function:** Matches values greater than the given value.

3.4 `$gte` - Greater than or Equal to

Example:

```
db.orders.aggregate([
  { $match: { price: { $gte: 100 } } }
]);
```

💡 **Function:** Matches values equal to or greater than.

3.5 `$lt` - Less than

Example:

```
db.orders.aggregate([
  { $match: { price: { $lt: 100 } } }
]);
```

💡 **Function:** Matches values less than.

3.6 `$lte` - Less than or Equal to

Example:

```
db.orders.aggregate([
  { $match: { price: { $lte: 100 } } }
]);
```

💡 **Function:** Matches values equal to or less than.

---

4. Boolean Operators

Used for logical operations.

4.1 `$and` - Logical AND

Example:

```
db.orders.aggregate([
  { $match: { $and: [{ status: "completed" }, { price: { $gt: 100 } }] } }]);
```

💡 **Function:** Matches all conditions.

4.2 `$or` - Logical OR

Example:

```
db.orders.aggregate([
  { $match: { $or: [{ status: "pending" }, { price: { $lt: 50 } }] } }]);
```

💡 **Function:** Matches any condition.

4.3 `$not` - Logical NOT

Example:

```
db.orders.aggregate([
  { $match: { price: { $not: { $gte: 100 } } } }]);
```

💡 **Function:** Negates a condition.

4.4 `$nor` - Logical NOR

Example:

```
db.orders.aggregate([
  { $match: { $nor: [{ status: "completed" }, { price: { $gte: 100 } }] } }]);
```

💡 **Function:** Matches documents where none of the conditions are true.

---

5. String Operators

Used for manipulating and analyzing strings in documents.

5.1 `$concat` - Concatenate Strings

Joins multiple strings into one.
Example:

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

💡 **Function:** Merges strings into a single string.

5.2 `$substr` - Extract a Substring

Extracts a portion of a string based on the starting index and length.
Example:

```
db.products.aggregate([
  { $project: { shortCode: { $substr: ["$productName", 0, 5] }
} }
]);
```

💡 **Function:** Extracts a substring from a string.

5.3 `$toLower` - Convert to Lowercase

Converts a string to lowercase.
Example:

```
db.users.aggregate([
  { $project: { emailLower: { $toLower: "$email" } } }
]);
```

💡 **Function:** Transforms text to lowercase.

5.4 `$toUpper` - Convert to Uppercase

Converts a string to uppercase.
Example:

```
db.users.aggregate([
  { $project: { usernameUpper: { $toUpper: "$username" } } }
]);
```

💡 **Function:** Transforms text to uppercase.

5.5 `$strLenCP` - String Length

Finds the number of characters in a string.
Example:

```
db.users.aggregate([
  { $project: { nameLength: { $strLenCP: "$name" } } }
]);
```

💡 **Function:** Counts the number of characters.

5.6 `$regexMatch` - Match a String Using Regex

Checks if a string matches a regular expression pattern.
Example:

```
db.users.aggregate([
  { $match: { email: { $regexMatch: { input: "$email", regex: "@gmail.com$" } } } }
]);
```

💡 **Function:** Matches text using a regular expression.

---

6. Date Operators

Used for extracting parts of a date.

6.1 `$year` - Extract Year

Extracts the year from a date.
Example:

```
db.orders.aggregate([
  { $project: { orderYear: { $year: "$orderDate" } } }]);
```

💡 **Function:** Gets the year from a date field.

6.2 `$month` - Extract Month

Extracts the month from a date.
Example:

```
db.orders.aggregate([
  { $project: { orderMonth: { $month: "$orderDate" } } }]);
```

💡 **Function:** Gets the month from a date field.

6.3 `$dayOfMonth` - Extract Day of Month

Extracts the day of the month from a date.
Example:

```
db.orders.aggregate([
  { $project: { orderDay: { $dayOfMonth: "$orderDate" } } }]);
```

💡 **Function:** Gets the day of the month from a date field.

6.4 `$hour` - Extract Hour

Extracts the hour from a date.
Example:

```
db.orders.aggregate([
  { $project: { orderHour: { $hour: "$orderDate" } } }]);
```

💡 **Function:** Gets the hour from a date field.

6.5 `$minute` - Extract Minutes

Extracts the minutes from a date.
Example:

```
db.orders.aggregate([
  { $project: { orderMinute: { $minute: "$orderDate" } } }]);
```

💡 **Function:** Gets the minutes from a date field.

6.6 `$second` - Extract Seconds

Extracts the seconds from a date.
Example:

```
db.orders.aggregate([
  { $project: { orderSecond: { $second: "$orderDate" } } }]);
```

💡 **Function:** Gets the seconds from a date field.

6.7 `$dateToString` - Format a Date as a String

Converts a date field into a formatted string.
Example:

```
db.orders.aggregate([
  { $project: { formattedDate: { $dateToString: { format: "%Y-
%m-%d", date: "$orderDate" } } } }
]);
```

💡 **Function:** Formats a date as a string.

---

7. Conditional Operators

Used to apply conditional logic (if-else behavior).

7.1 `$cond` - If-Else Condition

Evaluates a condition and returns different values based on true/false.
Example:

```
db.orders.aggregate([
  { $project: { statusLabel: { $cond: { if: { $gte: ["$totalPrice", 100] }, then: "High", else: "Low" } } } }
]);
```

💡 **Function:** Implements if-else conditions.

7.2 `$ifNull` - Default Value for Null

Returns a default value if a field is null.
Example:

```
db.orders.aggregate([
  { $project: { discount: { $ifNull: ["$discountAmount", 0] } } }
]);
```

💡 **Function:** Provides a default value if a field is missing or null.

---

8. Pipeline Stages

Defines stages for processing documents in an aggregation pipeline.

8.1 `$match` - Filter Documents (Similar to `find`)

Filters documents based on a condition.
Example:

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

💡 **Function:** Filters documents based on conditions.

8.2 `$group` - Group Documents

Groups documents by a specified field and performs aggregation.
Example:

```
db.orders.aggregate([
  { $group: { _id: "$customerId", totalSpent: { $sum: "$price" } } }]);
```

💡 **Function:** Groups documents and performs calculations.

8.3 `$project` - Reshape Documents

Selects and transforms specific fields in the output.
Example:

```
db.orders.aggregate([
  { $project: { orderId: 1, totalPrice: 1, _id: 0 } }]);
```

💡 **Function:** Restructures documents.

8.4 `$sort` - Sort Documents

Sorts documents in ascending or descending order.
Example:

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

💡 **Function:** Orders results by a field.

8.5 `$limit` - Limit Documents

Restricts the number of documents in the output.
Example:

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

💡 **Function:** Limits the number of results.

8.6 `$skip` - Skip Documents

Skips a specified number of documents.
Example:

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

💡 **Function:** Skips the first N documents.
