NOSQL DATABASE

MONGODB DAY 2

Lecture 2 Agenda

- **Objective** :
    - Deep Dive into **Read/Find Operations** Using MongoDB Operators
    - Explore Advanced **Update Operations** in MongoDB
    - **Data Aggregation Using MongoDB Aggregation Pipeline** (Advanced Data Operations)
    - **Ranking Function**

Lecture 2 Agenda

- Mongo Query Operators
  - Comparison
  - Logical
  - Arrays
  - Element
- Update and Upsert
- MongoDB Aggregation Pipeline
- **Ranking Function**

Find Exists Data [Is Not Null in SQL]

- db.inventory.find({ qty: { $exists: true}})

- **Hide Column , Show Columns**
  - db.staff.find({},{qty:true})
  - db.staff.find({},{qty:false})
  - Mixed
- **Hide _Id**
  - db.staff.find({},{_id:0})

Comparison Query Operators

db.inventory.find({ "qty" : { $Operator : value} })

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.

$eq [equal]

{ <field>: { $eq: <value> } }

- db.inventory.find({ qty: { $eq: 20 } })
- db.inventory.find({ qty: 20 })

- db.inventory.find({ "item.name": { $eq: "ab" } })
- db.inventory.find({ "item.name": "ab" })

- db.inventory.find({ tags: { $eq: "B" } })
- db.inventory.find({ tags: "B" })

$ne [not equal]

{ <field>: { $nq: <value> } }

- db.inventory.find({ qty: { $ne: 20 } })
- db.inventory.find({ qty: 20 })

- db.inventory.find({ "item.name": { $ne "ab" } })
- db.inventory.find({ "item.name": "ab" })

- db.inventory.find({ tags: { $ne: "B" } })
- db.inventory.find({ tags: "B" })

$gt $gte [Greater than – Equal]
$lt $lte [Less than – Equal]

- db.inventory.find({ quantity: { $gt: 95} })
- db.inventory.find({ quantity: { $gte: 95 } })

- db.inventory.find({ quantity: { $lt: 95} })
- db.inventory.find({ quantity: { $lte: 95 } })

$in $nin [In , Not In]

- db.inventory.find({ quantity: { $in: [5, 15] } }, { _id: 0 })
- db.inventory.find({ quantity: { $nin: [5, 15] } }, { _id: 0 })

Logical Query Operators

```
db.inventory.find( { $Operator:  [ { price: { $ne: 1.99 } } , { price: { $exists: true } } ] } )
```

Name	Description
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$not	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

$and

Syntax: { $and: [{ <expression1> }, { <expression2> } , ... , { <expressionN> }] }

- db.inventory.find({ $and: [{ price: { $ne: 1.99 } }, { price: { $exists: true } }] })
- db.staff.find({$and:[{"qty":15},{"name.fname":"first"}]})
- db.inventory.find({"$and":[{"$or":[{"qty":{$lt:10}},{"qty":{$gt:50}}]},{"$or":[{"sale":true},{"status":"A"}]}]})

$and

- db.inventory.find({

 $and: [

 { $or: [{ qty: { $lt: 100 } },

{ qty: { $gt: 10 } }] },

 { $or: [{ sale: true }, { "size.h": { $lt: 150 } }

 ] }] })

- db.staff.find({$and:[{$or:[{qty:15},{qty:20}]},{$or:[{"name.fname":"first"},{"name.fname":"first2"}]}]})

$not

Syntax: { field: { $not: { <operator-expression> } } }

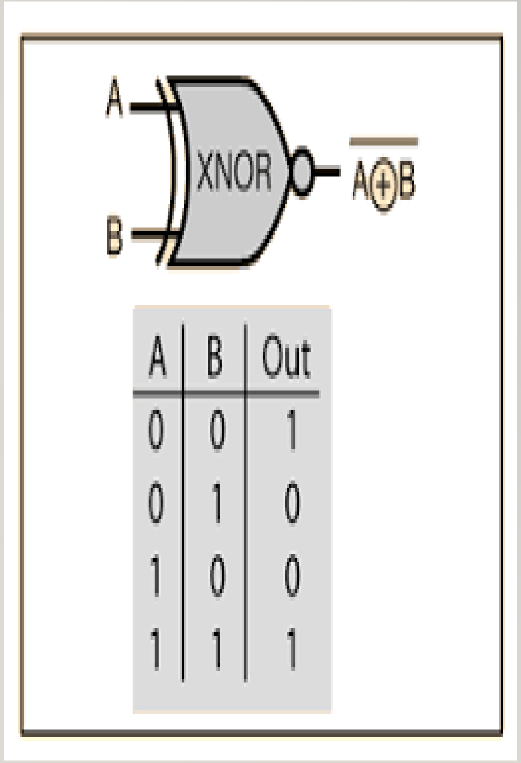
- db.staff.find({qty:{$not:{$gt:2}}})
- db.inventory.find({ price: { $not: { $gt: 1.99 } } })

$nor

db.inventory.find({ $nor: [{ price: 1.99 }, { sale: true }] })

- performs a logical NOR operation on an array of one or more query expression and selects the documents that fail all the query expressions in the array.

- db.inventory.find({ $nor: [{ price: 1.99 }, { sale: true }] })
- db.inventory.find({ $nor: [{ price: 1.99 }, { qty: { $lt: 20 } }, { sale: true }] })
- db.inventory.find({ $nor: [{ price: 1.99 }, { price: { $exists: false } },
- { sale: true }, { sale: { $exists: false } }] })



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

$or

{ $or: [{ <expression1> }, { <expression2> }, ... , { <expressionN> }] }

- db.inventory.find({ $or: [{ quantity: { $lt: 20 } }, { price: 10 }] })

- db.inventory.find({ $or: [{ quantity: { $lt: 20 } }, { price: 10 }] })

Regular Expression

- db.inventory.find({ item: { $not: /^p.*/ } })   //^ [Shift + 6]
- inventory collection where the item field value does not start with the letter p.
- db.inventory.find({ item: { $not: { $regex: "^p.*" } } })
- db.inventory.find({ item: { $not: { $regex: /^p.*/ } } })
- More Examples Regular Expression

  https://www.mongodb.com/docs/manual/reference/operator/query/regex/
- More Examples Evaluation Query Operators

  https://www.mongodb.com/docs/manual/reference/operator/query-evaluation/

Regular Expression

- db.products.insertMany(**[{**item:"ABC"**},{**item:"abc"**}])**
- **db**.**products**.**find({** item**:** **{** $regex: "(?i)abc" **} }** **)**

Regular Expression

Find name length : 3

```
db.inventory.find({
"name": /^.{3}$/
})
```

Array Query Operators

All [Taken Value]

{<filed>:{$all:[<value1>,<value2>,…]}}

- Below equal results:
- db.inventory.find({"tags":{$nin:["A","B"]}})
- db.inventory.find({"tags":{$all:["A","B"]}})
- db.inventory.find({ tags: { $all: ["ssl" , "security"] } })
- db.inventory.find({ $and: [{ tags: "ssl" }, { tags: "security" }] })
- db.inventory.find({ tags: ["ssl" , "security"] })

Nested Array

- All below are equals:
- db.articles.find({ tags: { $all: [["ssl", "security"]] } })
- db.articles.find({ $and: [{ tags: ["ssl", "security"] }] })
- db.articles.find({ tags: ["ssl", "security"] })
- db.inventory.find({ tags: { $all: ["appliance", "school", "book"] } })

The $all expression with a single element is for illustrative purposes since the $all

expression is **unnecessary if matching only a single element**. Instead, when matching a single element, a "contains" expression (i.e. arrayField: element) is more suitable.

$in , $all , $and

- db.inventory.find({tags:{$in:["red","blank"]}})
- db.inventory.find({tags:{$all:["red","blank"]}})
- db.inventory.find({$and:[{tags:"blank"},{tags:"blank"}]})

$size

- db.inventory.find({ tags: { $size: 2 } });
- db.inventory.find({ tags: { $size: 3 } });

Element Query Operator

- $exists

https://www.mongodb.com/docs/manual/reference/operator/query/type/#mongodb-query-op.-type

- db.data.find({ x: { $type: "minKey" } })
- db.data.find({ y: { $type: "maxKey" } })
- db.addressBook.find({ "zipCode" : { $type : 2 } });
- db.addressBook.find({ "zipCode" : { $type : "string" } });
- db.addressBook.find({ "zipCode" : { $type : 1 } })
- db.addressBook.find({ "zipCode" : { $type : "double" } })
- db.students.find({“alias" : { $type : 2 } });

Update

- db.inventory.find({"item":"paper"})
- updateOne
  - db.inventory.updateOne({ item: "paper" }, { $set: { "size.uom": "cm", status: "P" }, $currentDate: { lastModified: true } })
- updateMany
  - db.inventory.updateMany({ "qty": { $lt: 50 } }, { $set: { "size.uom": "in", status: "P" }, $currentDate: { lastModified: true } })
- replaceOne
  - db.inventory.replaceOne({ item: "paper" }, { item: "paper", instock: [{ warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 }] })

Update Operators

Name	Description
$currentDate	Sets the value of a field to current date, either as a Date or a Timestamp.
$inc	Increments the value of the field by the specified amount.
$min	Only updates the field if the specified value is less than the existing field value.
$max	Only updates the field if the specified value is greater than the existing field value.
$mul	Multiplies the value of the field by the specified amount.
$rename	Renames a field.
$set	Sets the value of a field in a document.
$setOnInsert	Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
$unset	Removes the specified field from a document.

$currentDate

-
  ```
  db.customers.updateOne(
  { _id: 1 },
  {
    $currentDate:
                {
                lastModified: true,
                "cancellation.date": { $type: "timestamp" }
                },
    $set: {
      "cancellation.reason": "user request",
      status: "D"
        }
  }
  )
  ```

$inc

- db.products.updateOne(

```
  { sku: "abc123" },
  { $inc: { quantity: -2, "metrics.orders": 1 } }
)
```

$min

- db.scores.updateOne({ _id: 1 }, { $min: { lowScore: 150 } })
- db.scores.updateOne({ _id: 1 }, { $min: { lowScore: 250 } })

Use $min to Compare Dates

- db.tags.insertOne(

```
  {
    _id: 1,
    desc: "crafts",
    dateEntered: ISODate("2013-10-01T05:00:00Z"),
    dateExpired: ISODate("2013-10-01T16:38:16Z")
  }
)
```

- db.tags.updateOne(

```
  { _id: 1 },
  { $min: { dateEntered: new Date("2013-09-25") } }
)
```

$max

- db.scores.updateOne({ _id: 1 }, { $max: { highScore: 950 } })
- Use $Max to Compare Dates
- db.tags.updateOne(

  { _id: 1 },

  { $max: { dateExpired: new Date("2013-09-30") } }

)

$mul

- db.products.insertOne({ "_id" : 1, "item" : "Hats", "price" : Decimal128("10.99"), "quantity" : 25 })
- db.products.updateOne(

```
  { _id: 1 },
  { $mul:
    {
      price: Decimal128( "1.25" ),
      quantity: 2
     }
  }
)
```

- db.products.updateOne(

```
  { _id: 3 },
  { $mul: { price: Int32(5) } }
)
```

$rename

- db.students.updateMany({}, { $rename: { "nmae": "name" } })

$set

```
• db.products.updateOne(
   { _id: 100 },
   { $set:
      {
        quantity: 500,
        details: { model: "2600", make: "Fashionaires" },
        tags: [ "coats", "outerwear", "clothing" ]
      }
   }
)
```

$set

- db.products.updateOne(
```
 { _id: 100 },
 { $set: { "details.make": "Kustom Kidz" } }
)
```
- db.products.updateOne(
```
 { _id: 100 },
 { $set:
   {
     "tags.1": "rain gear",
     "ratings.0.rating": 2
   }
 }
)
```

$setOnInsert

- db.products.updateOne(

```
 { _id: 1 },
 {
   $set: { item: "apple" },
   $setOnInsert: { defaultQty: 100 }
 },
 { upsert: true }
)
```

$unset

- db.products.updateOne(

  { sku: "unknown" },

  { $unset: { quantity: "", instock: "" } }

)

upsert

- Upsert with Replacement Document
- If no document matches the query criteria and the <update> parameter is a replacement document (i.e., contains only field and value pairs), the update inserts a new document with the fields and values of the replacement document.
- If you specify an _id field in either the query parameter or replacement document, MongoDB uses that _id field in the inserted document.
- If you do not specify an _id field in either the query parameter or replacement document, MongoDB generates adds the _id field with a randomly generated ObjectId value.

upsert

- db.books.update(

```
  { item: "ZZZ135" },   // Query parameter
  {                     // Replacement document
   $set:{ item: "ZZZ135",
    stock: 5,
    tags: [ "database" ]
  }},
  { upsert: true }      // Options
)
```

upsert

- db.people.update(

```
  { name: "Andy" },
  { $inc: { score: 1 } },
  {
    upsert: true,
    multi: true
  }
)
```

Mongo Aggregation Pipeline

- An aggregation pipeline consists of one or more **stages** that process documents:
- Each stage performs an operation on the **input** documents. For example, a stage can filter documents, group documents, and calculate values.
- The documents that are **output** from a stage are **passed to the next stage**.
- An aggregation pipeline can return results for groups of documents. For example, return the total, average, maximum, and minimum values.

Mongo Aggregation Pipeline

Stage	Description
$addFields	Adds new fields to documents. Similar to $project, $addFields reshapes each document in the stream; specifically, by adding new fields to output documents that contain both the existing fields from the input documents and the newly added fields. **$set is an alias for $addFields.**
$count	Returns a **count** of the number of documents at this stage of the aggregation pipeline. Distinct from the $count aggregation accumulator.
$fill	Populates **null** and **missing** field values within documents.

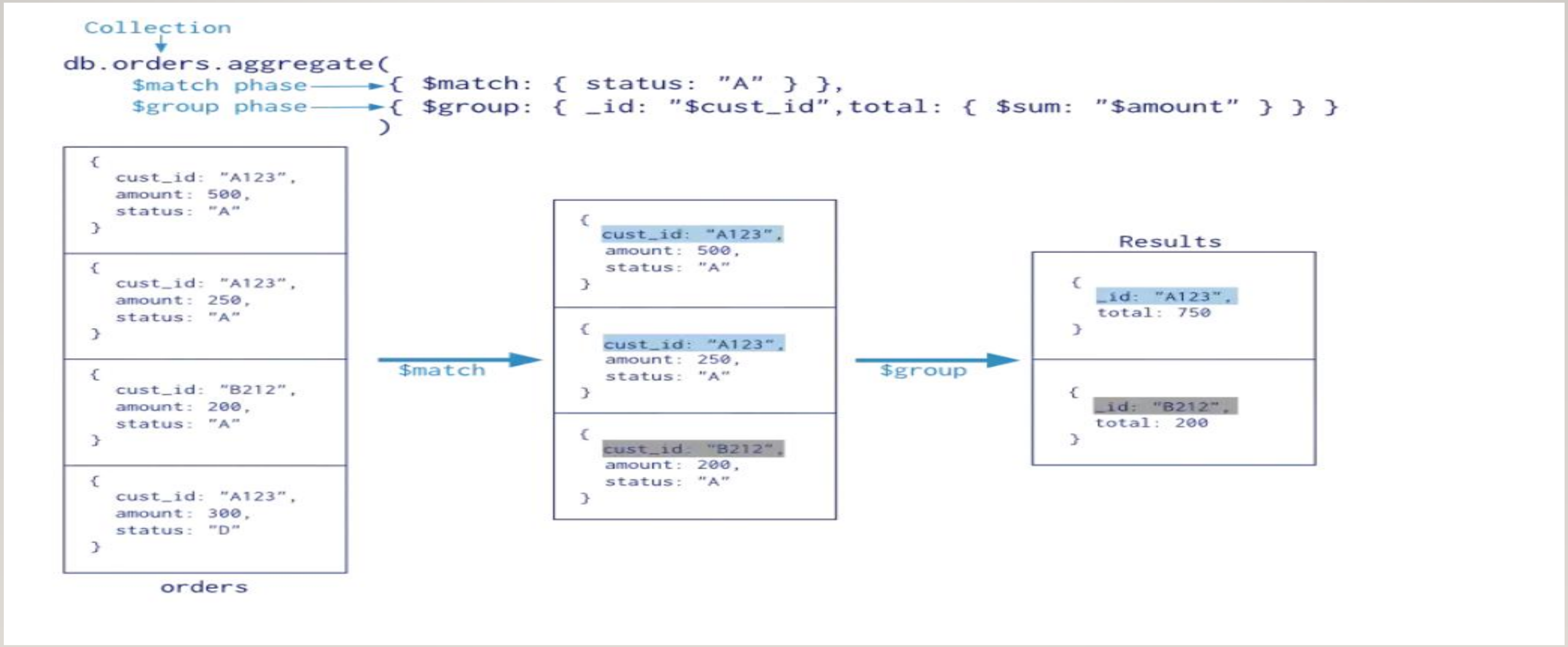
Mongo Aggregation Pipeline

Stage	Description
$match	Filters the document stream to allow only matching documents to pass unmodified into the next pipeline stage. $match uses standard MongoDB queries. For each input document, outputs either one document (a match) or zero documents (no match).
$out	**[Select into in SQL]** Writes the resulting documents of the aggregation pipeline to a collection. To use the $out stage, it must be the last stage in the pipeline.

https://www.mongodb.com/docs/manual/reference/operator/aggregation-pipeline/#std-label-aggregation-pipeline-operator-reference

Aggregation Pipeline Stages

Total Amount Order for Status A For each cust_id ,SQL ?



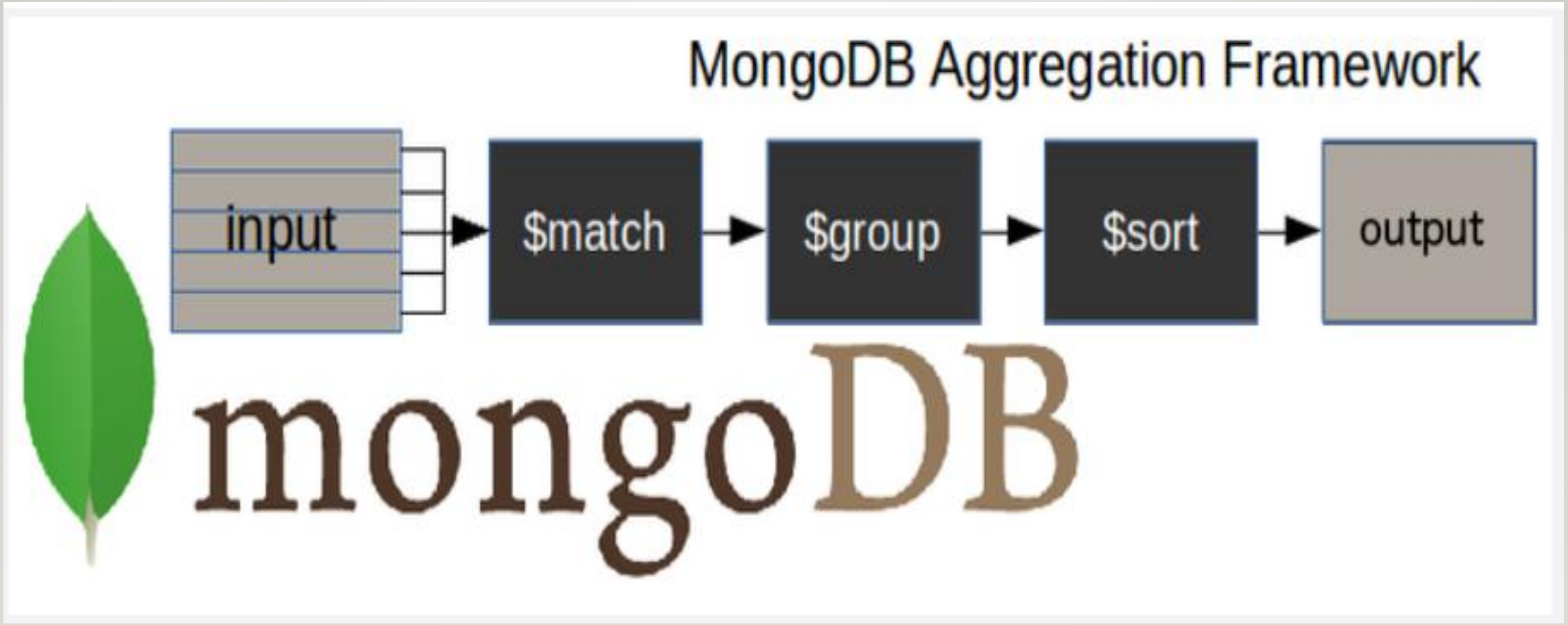
```
db.orders.aggregate(
    { $match: { status: "A" } },
    { $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
)
```

```
{
   cust_id: "A123",
   amount: 500,
   status: "A"
}
{
   cust_id: "A123",
   amount: 250,
   status: "A"
}
{
   cust_id: "B212",
   amount: 200,
   status: "A"
}
{
   cust_id: "A123",
   amount: 300,
   status: "D"
}
```

```
{
   cust_id: "A123",
   amount: 500,
   status: "A"
}
{
   cust_id: "A123",
   amount: 250,
   status: "A"
}
{
   cust_id: "B212",
   amount: 200,
   status: "A"
}
```

```
{
   _id: "A123",
   total: 750
}
{
   _id: "B212",
   total: 200
}
```

MongoDB Aggregation Framework



Calculate **Total Order Quantity**

The following aggregation pipeline example contains **two stages** and returns the total order quantity of *medium* size pizzas grouped by ***pizza name***:

```
db.orders.aggregate( [

   // Stage 1: Filter pizza order documents by pizza size

   {

      $match: { size: "medium" }

   },

   // Stage 2: Group remaining documents by pizza name and calculate total quantity

   {

      $group: { _id: "$name", totalQuantity: { $sum: "$quantity" } }

   }

] )
```

Calculate Total Order Value and Average Order Quantity
The following example calculates the total pizza order value and average order quantity between two dates:

```
 // Stage 1:

db.orders.aggregate( [

   // Stage 1: Filter pizza order documents by date range

   {

      $match:

      {

         "date": { $gte: new ISODate( "2020-01-30" ), $lt: new ISODate( "2022-01-30" ) }

      }

   },
```

// Stage 2

```
// Stage 2: Group remaining documents by date and calculate results
  {
    $group:
    {
      _id: { $dateToString: { format: "%Y-%m-%d", date: "$date" } },
      totalOrderValue: { $sum: { $multiply: [ "$price", "$quantity" ] } },
      averageOrderQuantity: { $avg: "$quantity" }
    }
  },
```

// Stage 3:

```
// Stage 3: Sort documents by totalOrderValue in descending order
  {
     $sort: { totalOrderValue: -1 }
  }

])
```

Mongo Aggregation

- db.orders.aggregate([/* Stage 1: Filter pizza order documents by date range*/ { $match: { "date": { $gte: new ISODate("2020-01-30"), $lt: new ISODate("2022-01-30") } } }, /* Stage 2: Group remaining documents by date and calculate results*/ { $group: { _id: { $dateToString: { format: "%Y-%m-%d", date: "$date" } }, totalOrderValue: { $sum: { $multiply: ["$price", "$quantity"] } }, averageOrderQuantity: { $avg: "$quantity" } } }, /* Stage 3: Sort documents by totalOrderValue in descending order*/ { $sort: { totalOrderValue: -1 } }])

$out

- Create New Collection
- operation creates a new collection if one does not already exist.
- Like Select into in SQL

$out Example

```
db.orders.aggregate( [

   // Stage 1: Filter pizza order documents by pizza size
   {
      $match: { size: "medium" }
   },
   // Stage 2: Group remaining documents by pizza name and calculate total quantity
   {
      $group: { _id: "$name", totalQuantity: { $sum: "$quantity" } }
   },
      {$out: "newCollectionName"}
] )
```

Aggreagtion $count

```
db.orders.aggregate( [
{ $match: { size: "large" } },
 { $count: "passing_scores" }
])
```

Some of Built-In Functions : limit() , count()

- db.orders.find({}).limit(1)
- db.orders.find({}).count()

Aggregation $count vs countDocuments

- db.orders.aggregate([

{ $match: { size: "large" } },

{ $count: "passing_scores" }

])

- db.orders.find({ size: "large"}).count() // Deprecated in **MongoDB 4.0+**
- db.orders.**countDocuments(**{size:"large"**})**

Aggregation $count vs countDocuments

Feature	aggregate([...]) with $count	find({}).countDocuments()
Performance	Slower (uses aggregation pipeline)	Faster for simple counts
Output	JSON document with a named field	Direct integer count
Suitable for	Complex queries requiring transformations When you are using aggregation pipelines and need to process data further.	Simple counting
MongoDB Version	Requires MongoDB 3.4+	Requires MongoDB 4.0+

count vs countDocuments

- Why countDocuments() Instead of count()?
- **Accurate** count: countDocuments() provides an accurate count of matching documents.
- **Indexes are considered**: It uses indexes when available.
- **Performance**: More efficient than count() when dealing with large collections.

min / max in the update operation (as operators) VS min / max in the aggregate pipeline

- **min / max in the update operation (as operators)**

```
db.products.updateOne(
  { _id: 1 },
  { $min: { price: 50 }, $max: { quantity: 200 } }
)
```

- **min / max in the aggregate pipeline**

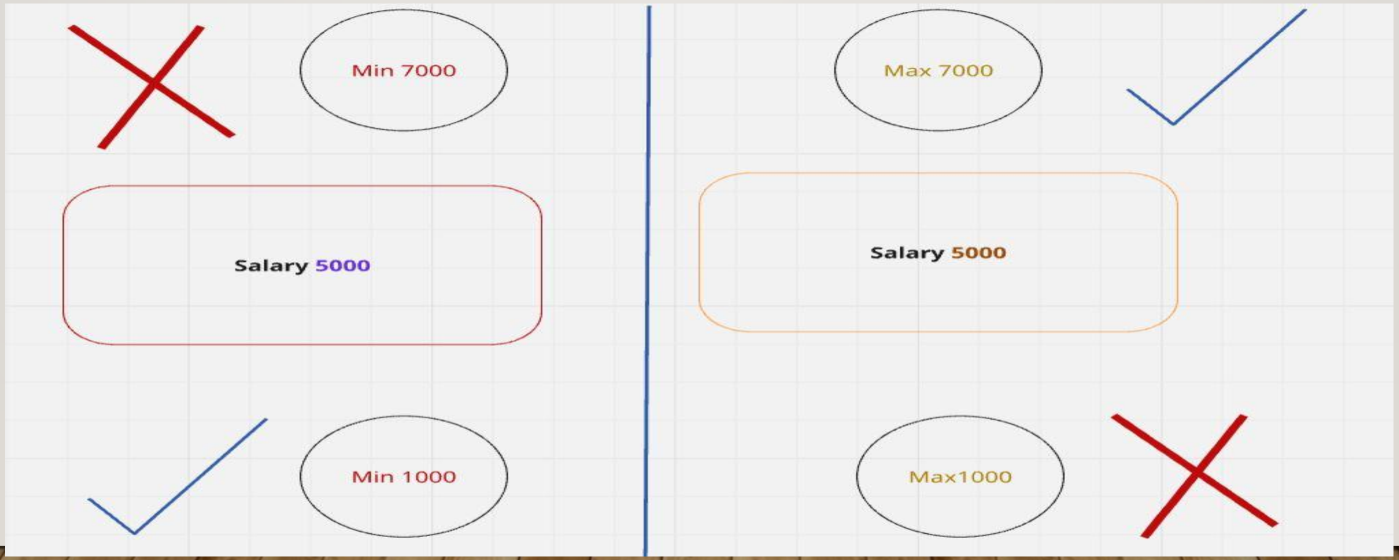
```
db.sales.aggregate([
  { $group: {
    _id: "$product",
    minPrice: { $min: "$price" },
    maxPrice: { $max: "$price" }
  }}])
```

min / max in the update operation (as operators)

- **min / max in the update operation (as operators)**

$**min**: Updates the field only if the new value is less than the current field value.

$**max**: Updates the field only if the new value is greater than the current field value.



min / max in the aggregate pipeline

min / max in the aggregate pipeline

•**$min**: Returns the ***lowest*** value in the group.

•**$max**: Returns the ***highest*** value in the group.

Aggregate Ranking Function

SQL:

```
SELECT name, department, salary,
       RowNumber() OVER (PARTITION BY department ORDER BY salary DESC) as rank
FROM employees
```

```
db.departments.insertMany([
  { _id: 1, name: "IT",       location: "Cairo" },
  { _id: 2, name: "HR",       location: "Alexandria" }
])
```

```
db.employees.insertMany([
  { "name": "Ahmed", "department": "IT", "salary": 5000 },
  { "name": "Omar",  "department": "IT", "salary": 7000 },
  { "name": "Sara",  "department": "IT", "salary": 7000 },
  { "name": "Eman",  "department": "IT", "salary": 8000 },
  { "name": "Noha",  "department": "IT", "salary": 8000 },
  { "name": "Mona",  "department": "HR", "salary": 4000 },
  { "name": "Ali",  "department": "HR", "salary": 3000 }
])
```

```
db.employees.aggregate([
 {
   $setWindowFields: {
     partitionBy: "$department",          // like PARTITION BY
     sortBy: { salary: -1 },              // like ORDER BY salary DESC
     output: {
       rank: { $rank: {} },               // RANK()
       rowNumber: { $documentNumber: {} },// ROW_NUMBER()
       denseRank: { $denseRank: {} }      // DENSE_RANK()
     }
   }
 }
])
```

THANK YOU

Any Question