- files of mongodb are saved as Javascripts

- to export colection → select
  → export
  → choose file type

- MongoDB aggregation framework:- (stages)

1- Input
2- $match
3- $ group ⇒ like adding values together
4- $ sort
5- output

- $ sum: "$ amount'
  → the value of the amount cell not the word so that we added $ to amount

$\boxed{1}$ find employees total salary for Sales Department

$\boxed{1}$ match stage where Sales department

$\boxed{2}$ group stage $\longrightarrow$ sum + Grouping

$\boxed{3}$ output

```
db.employees.aggregate ([
    {     //match stage where sales department
        $match: {
            "department" : "sales"          => you can
         }                                    check each
    },                                        stage separately
    {
      //group stage total salary
      //Final output sales, total salary
       //sum
      // grouping
        $group: {
            _id: "$department" ,              ┐
            totalSalary: { $sum : "$salary" } ┘→ to print total sal & depatment columns
        }
 }
 ])
```

$\boxed{2}$ avg salaries for each department sorted dec.

⇒ we have No where condition ⇒ match stage

⇒ group Avg Salaries
uniquenes

⇒ sort stage.

```
db.employees .aggregate([
    {  // No match
    },
    {
       // group stage
       $group: {
          _id : "$ department",
          avgSalary : { $avg : "$ salary" }
       }
    },
    {
         //sort stage
          avgSalary :-1
    }
}
])
```

if some values doesn't have department and was shown by mistake ⇒ you can add "department":{$exist: true} in match stage

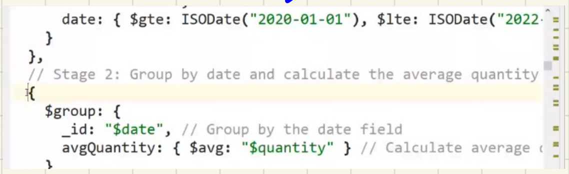
⇒ we mostly use it because it's unique +key value

⇒ if I want it Acen. ⇒ 1

to save the output we can add stage 4

```
// stage 4: Save output in new collection
{
  $out: "medium_pizza_avg"
}
```

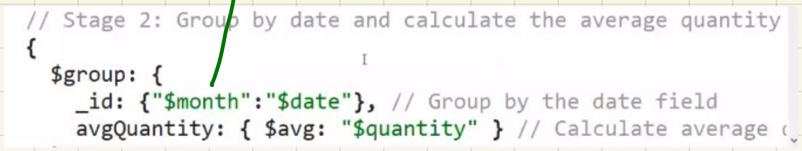
→ search about dates



```
      date: { $gte: ISODate("2020-01-01"), $lte: ISODate("2022-
    }
  },
  // Stage 2: Group by date and calculate the average quantity
  {
    $group: {
      _id: "$date", // Group by the date field
      avgQuantity: { $avg: "$quantity" } // Calculate average
```

⇒ we can adjust only the match and group not the others

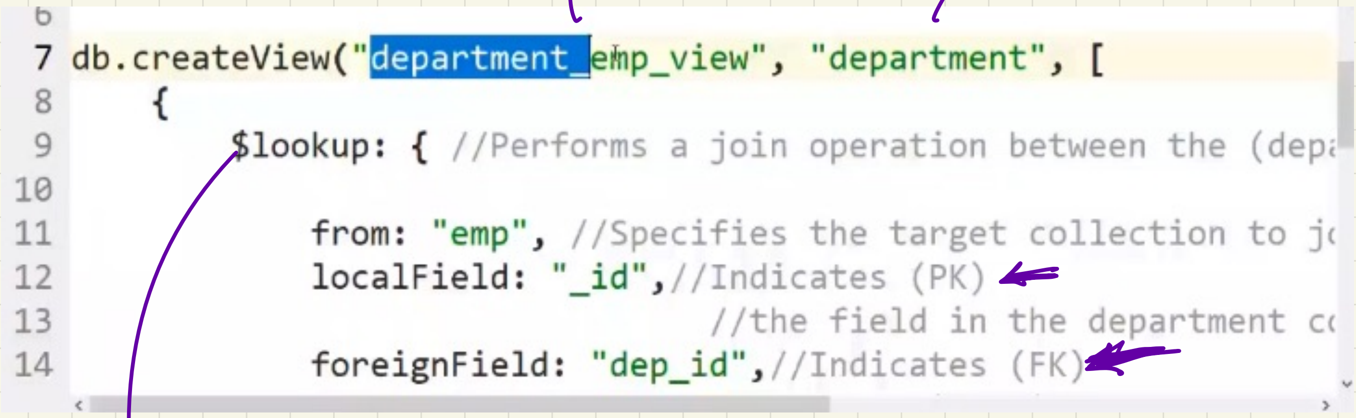
→ group by month.



```
// Stage 2: Group by date and calculate the average quantity
{
  $group: {
    _id: {"$month":"$date"}, // Group by the date field
    avgQuantity: { $avg: "$quantity" } // Calculate average
```

view name

parent table

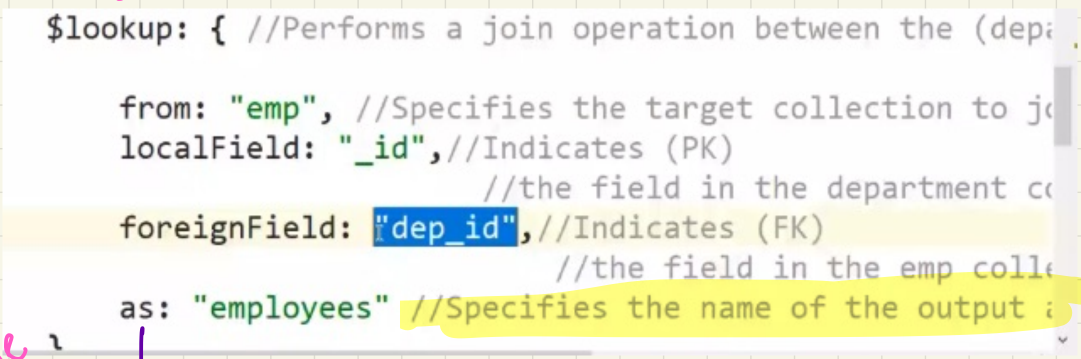


```
db.createView("department_emp_view", "department", [
    {
        $lookup: { //Performs a join operation between the (depa

            from: "emp", //Specifies the target collection to jo
            localField: "_id",//Indicates (PK)
                                    //the field in the department co
            foreignField: "dep_id",//Indicates (FK)
```

Create the view

first pipe line

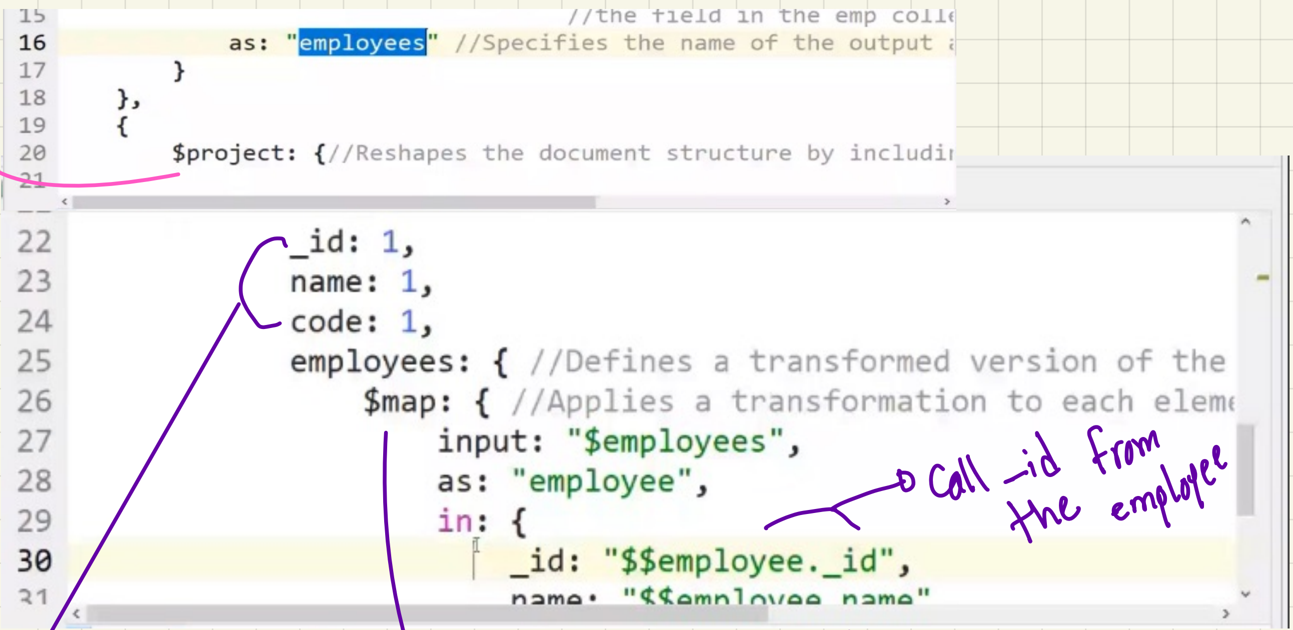


```
$lookup: { //Performs a join operation between the (depa

    from: "emp", //Specifies the target collection to jo
    localField: "_id",//Indicates (PK)
                             //the field in the department co
    foreignField: "dep_id",//Indicates (FK)
                                //the field in the emp colle
    as: "employees" //Specifies the name of the output a
```

2nd pipe line

alies name to use



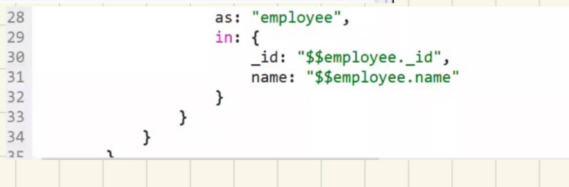
```
                              //the field in the emp colle
            as: "employees" //Specifies the name of the output a
        }
    },
    {
        $project: {//Reshapes the document structure by includi
            _id: 1,
            name: 1,
            code: 1,
            employees: { //Defines a transformed version of the
                $map: { //Applies a transformation to each eleme
                    input: "$employees",
                    as: "employee",
                    in: {
                        _id: "$$employee._id",
                        name: "$$employee.name"
```

Call _id from the employee

Show these Columns

perform mapping

to reshape input



```
                    as: "employee",
                    in: {
                        _id: "$$employee._id",
                        name: "$$employee.name"
                    }
                }
            }
```

* Create view Connect 2 collections in tree view (join)

from ⇒ child

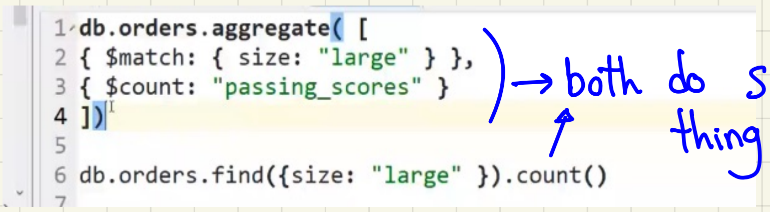
localField → PK of parent collection

foreignField ⇒ FK of child collection

as: ⇒ alies name for the result

* the aim of reshape is to view data as list

* there's a stage called Count in the aggregate



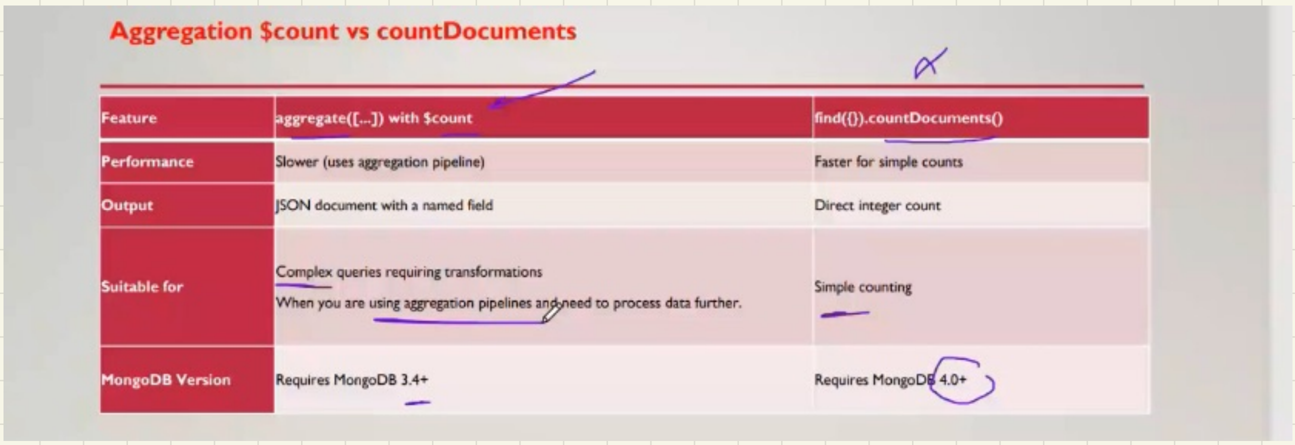
```
db.orders.aggregate( [
{ $match: { size: "large" } },
{ $count: "passing_scores" }
])

db.orders.find({size: "large" }).count()
```

→ both do same thing ① pipe line ② method

⇒ we use pipe line in the case that I need the output to go in the next stage ⇒ if levels

⇒ if it's Simple we better use methods



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+

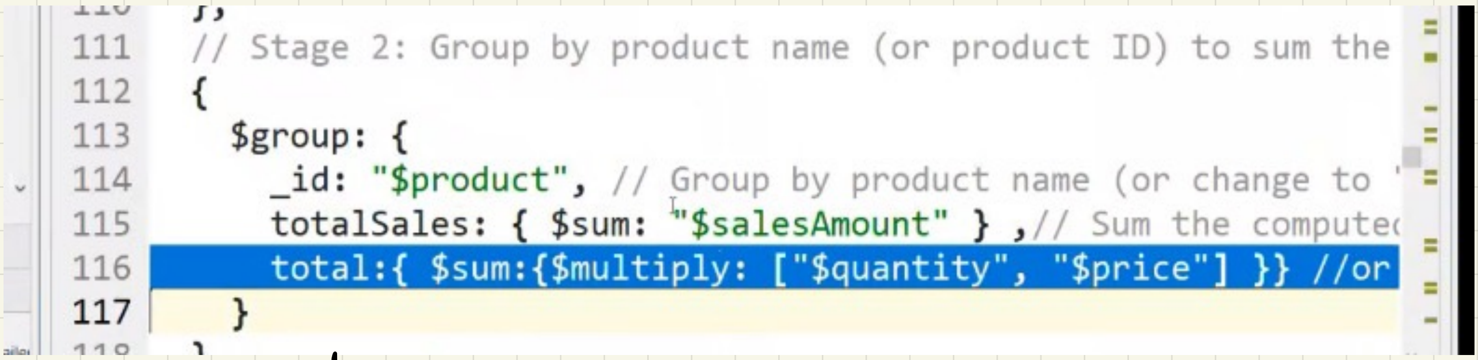
- countDocument() vs Count()

  ↓

  preffered because faster in retrival data and better performance

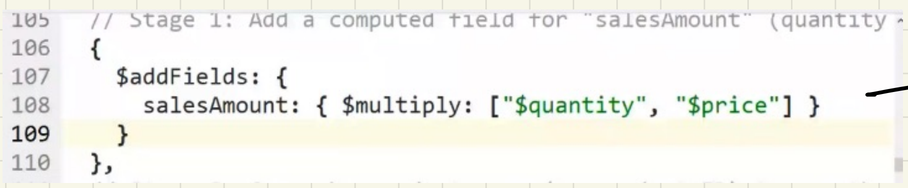
- create view Consists of some of pipelines ex → join
  → reshape

  → the output & alias of the pipeline is $\boxed{\text{as}}$



```
  },
  // Stage 2: Group by product name (or product ID) to sum the
  {
    $group: {
      _id: "$product", // Group by product name (or change to '
      totalSales: { $sum: "$salesAmount" } ,// Sum the computed
      total:{ $sum:{$multiply: ["$quantity", "$price"] }} //or
    }
```

→ solution for eq :- total = qty * price



```
// Stage 1: Add a computed field for "salesAmount" (quantity
{
  $addFields: {
    salesAmount: { $multiply: ["$quantity", "$price"] }
  }
},
```

→ another Solution

Recap Lec 1,2

structured $\Rightarrow$ key value , جداول , col $\Rightarrow$ type , row $\Rightarrow$ data

unstructured $\Rightarrow$ dynamic

Json $\Rightarrow$ semi structured

CAP theory