Adding Stages to a Collections Pipeline

Axel Sirota
MACHINE LEARNING ENGINEER
@AxelSirota

More Pipelines Stages: Grouping

$group Syntax

```
{
  $group:
    {
      _id: <expression>, // Group By Expression

      <field1>: { <accumulator1> : <expression1> },
      ...
    }
 }
```

- **# We define under which keys we want to group by!**
- **# The operations we want to do**

An Example Use of $group

```
db.rent.aggregate([
  {
     $group: {
        _id: {
           neighbourhood: "$neighbourhood_cleansed"
        },
        count: {
           $sum: 1
        },
        average_price: {
           $avg: {
              $toDouble: {
                 $reduce: {
                    input: {
                       $split: [{ $substr: [ "$price", 1, -1 ]}, ',']
                    },
                    initialValue: '',
                    in: {
                       $concat: ['$$value', '$$this']
}}}}}}}])
```

An Example Use of $group

```
db.rent.aggregate([
  {
    $group: {
      _id: {
        neighbourhood: "$neighbourhood_cleansed"
      },
      count: {
        $sum: 1
      },
      average_price: {
        $avg: {
          $toDouble: {
            $reduce: {
              input: {
                $split: [{ $substr: [ "$price", 1, -1 ]}, ',']
              },
              initialValue: '',
              in: {
                $concat: ['$$value', '$$this']
}}}}}}])
```

Equivalent SQL

```
SELECT
    neighbourhood,
    COUNT(*) AS count,
    AVERAGE(price) AS price
FROM RENT
GROUPBY neighbourhood
```

An Example Use of $group

{ "_id" : { "neighbourhood" : "Bayside" }, "count" : 71, "average_price" : 218.85915492957747 }
{ "_id" : { "neighbourhood" : "Tompkinsville" }, "count" : 44, "average_price" : 78 }
{ "_id" : { "neighbourhood" : "Tottenville" }, "count" : 2, "average_price" : 209.5 }
{ "_id" : { "neighbourhood" : "Inwood" }, "count" : 247, "average_price" : 85.69635627530364 }
{ "_id" : { "neighbourhood" : "SoHo" }, "count" : 378, "average_price" : 260.7777777777777 }
...,

$group by Multiple Fields

```
db.rent.aggregate([
  {
    $group: {
      _id: {
        neighbourhood:
"$neighbourhood_cleansed",
        room: "$room_type"
      },
      count: {
        $sum: 1
      }
    }
  },
  {
    $sort: {
      "_id.neighbourhood" : -1,
      "_id.room": -1
    }
  }
])
```

◀ **# We can group on multiple fields**

◀ **It is good practice to do $group: {_id: { field: “$field”}}**

◀ **# And sort by them**

Some Accumulators to Recall

sum	avg	addToSet
push	min/max	first/last

$push in Action

```
db.rent.aggregate([
  {
     $group: {
        _id: {
           neighbourhood:
"$neighbourhood_cleansed"
        },
        count: {
           $sum: 1
        },
        types: {
           $push: {property_type:
"$property_type"}
        }
     }
  }])
```

◀ **# We group by neighborhood**

◀ **# Create a types array with the property types!**

Makes sense, right?!

$push in Action

{ "_id" : { "neighbourhood" : "Upper East Side" }, "count" : 1796, **"types"** :
[**{ "property_type" : "Apartment" }**, { "property_type" : "Apartment" }, { "property_type" :
"Apartment" }, { "property_type" : "Apartment" }, { "property_type" : "Apartment" },
{ "property_type" : "Apartment" }, { "property_type" : "Apartment" }, { "property_type" :
"Apartment" }, { "property_type" : "Apartment" }, { "property_type" : "Apartment" },
{ "property_type" : "Apartment" }, { "property_type" : "Apartment" }, { "property_type" :
"Apartment" }, { "property_type" : "Apartment" }, { "property_type" : "Apartment" },
{ "property_type" : "Apartment" }, { "property_type" : "Apartment" }, { "property_type" :
"Apartment" }, { "property_type" : "Apartment" }, { "property_type" : "Apartment" },
{ "property_type" : "Apartment" }, { "property_type" : "Apartment" }, { "property_type" :
"Apartment" }, { "property_type" : "Apartment" }, { "property_type" : "Apartment" },
{ "property_type" : "Apartment" }, { "property_type" : "Apartment" }, { "property_type" :
"Apartment" }, { "property_type" : "Apartment" }, { "property_type" : "Apartment" },
{ "property_type" : "Apartment" }, ...,

Trying $addToSet

```
db.rent.aggregate([
  {
     $group: {
        _id: {
           neighbourhood:
"$neighbourhood_cleansed"
        },
        count: {
           $sum: 1
        },
        types: {
           $addToSet: {property_type:
"$property_type"}
        }
     }
  }])
```

◀ **# We group by neighborhood**

◀ **# Create a types array with the property types in a unique manner!**

Makes sense, right?!

Trying $addToSet


```
{ "_id" : { "neighbourhood" : "Williamsburg" }, "count" : 3844, "types" : [
    { "property_type" : "Condominium" },
    { "property_type" : "Yurt" },
    { "property_type" : "Cabin" },
    { "property_type" : "Townhouse" },
    { "property_type" : "Guest suite" },
    { "property_type" : "Apartment" },
    { "property_type" : "Guesthouse" },
    { "property_type" : "Serviced apartment" },
    { "property_type" : "Cottage" },
    { "property_type" : "House" },
    { "property_type" : "Bed and breakfast" },
    { "property_type" : "Other" },
    { "property_type" : "Camper/RV" },
    { "property_type" : "Hostel" },
    { "property_type" : "Loft" }
] }
```

Trying $addToSet


```
{ "_id" : { "neighbourhood" : "Williamsburg" }, "count" : 3844, "types" : [
   { "property_type" : "Condominium" },
   { "property_type" : "Yurt" },
   { "property_type" : "Cabin" },
   { "property_type" : "Townhouse" },
   { "property_type" : "Guest suite" },
   { "property_type" : "Apartment" },
   { "property_type" : "Guesthouse" },
   { "property_type" : "Serviced apartment" },
   { "property_type" : "Cottage" },
   { "property_type" : "House" },
   { "property_type" : "Bed and breakfast" },
   { "property_type" : "Other" },
   { "property_type" : "Camper/RV" },
   { "property_type" : "Hostel" },
   { "property_type" : "Loft" }
] }
```


“property_type” is repeated!

```
db.rent.aggregate([
  {
     $group: {
        _id: {
           neighbourhood:
"$neighbourhood_cleansed"
        },
        count: {
           $sum: 1
        },
        types: {
           $addToSet: {property_type:
"$property_type"}
        }
     }
  },
  {
     $project: {
        _id:1,
        count:1,
        types: "$types.property_type"
     }
  }
```

Trying $addToSet

◀ **# With $project we can get rid of the repeated field**

Trying $addToSet

{ "_id" : { "neighbourhood" : "Williamsburg" }, "count" : 3844, "types" : [
"Guesthouse",
"Serviced apartment",
"Cottage",
"House",
"Bed and breakfast",
"Other",
"Camper/RV",
"Hostel",
"Loft",
"Condominium",
"Yurt",
"Cabin",
"Townhouse",
"Guest suite",
“Apartment"
] }

```
{
    $group: {
        _id: {
            neighbourhood:
"$neighbourhood_cleansed"
        },
        count: {
            $sum: 1
        },
        maximum_price: {
            $max: "$num_price"
        },
        average_price: {
            $avg: "$num_price"
        },
        std_price: {
            $stdDevPop: "$num_price"
        },
        minimum_price: {
            $min: "$num_price"
        }
    }
}
```

Other Accumulators

- **# Calculates the maximum of the prices of that neighborhood**
- **# And we can calculate the average**
- **# Or the standard Deviation**
- **# And many more!**

Other Accumulators

{ "_id" : { "neighbourhood" : "Battery Park City" }, **"count" : 76, "maximum_price" : 2500, "average_price" : 241.42105263157896, "std_price" : 296.4213423294039, "minimum_price" : 55** }
{ "_id" : { "neighbourhood" : "Unionport" }, "count" : 10, "maximum_price" : 450, "average_price" : 153.6, "std_price" : 110.84962787488283, "minimum_price" : 60 }
{ "_id" : { "neighbourhood" : "Oakwood" }, "count" : 3, "maximum_price" : 100, "average_price" : 93.33333333333333, "std_price" : 6.236095644623236, "minimum_price" : 85 }
{ "_id" : { "neighbourhood" : "Middle Village" }, "count" : 35, "maximum_price" : 265, "average_price" : 116.28571428571429, "std_price" : 52.84185106986504, "minimum_price" : 44 }
{ "_id" : { "neighbourhood" : "Windsor Terrace" }, "count" : 146, "maximum_price" : 495, "average_price" : 139.1027397260274, "std_price" : 90.46604823305182, "minimum_price" : 30 }

Accumulators in $project

```
{
    $project: {
        average_field: {
            $avg: "$my_list"    # If my_list is [2,4,3]
        }
    }
}
```

◀ # Calculates at a document level

◀ # There is no grouping!

When We Have Arrays, We Unwind!

What Is the Most Common Amenity?


```
{
    "_id" : ObjectId("5e8936d06347c0057a059f11"),
    "name" : "SPECIOUS ONE BEDROOM IN THE HEART OF  CHELSEA",
    ...,
    "property_type" : "Apartment",
    "room_type" : "Private room",
    "accommodates" : 2,
    "bathrooms" : 1,
    "bedrooms" : 1,
    "beds" : 1,
    "bed_type" : "Real Bed",
    "amenities" : "{TV,Wifi,\"Air conditioning\",Heating,\"Family/kid
friendly\",Essentials,Shampoo,Hangers,\"Host greets you\"}",
    "square_feet" : "",
    "price" : "$190.00",
    ...,
}
```

Getting Our Array

```
{
    $addFields: {
        amenities: {
            $split: [
                {
                    $substr: [
                      "$amenities",
                      1,
                      {$subtract: [{$strLenCP: "$amenities" },2]}
                    ]
                },
                ','
            ]
        }
    }
}
```

- **# We split the string by ‘,’**
- **# After removing the starting and ending ‘{’, ‘}’**

Getting Our Array

```
{
    "_id" :
ObjectId("5e8936cd6347c0057a053363"),
    "name" : "Modern NYC",
    "amenities" : [
        "Internet",
        "Wifi"
    ],
    "neighbourhood" : "Washington Heights"
}
```

◀ **# Success!**

Getting Our Array

```
{
    $group: {
        _id: {
            neighbourhood: "$neighbourhood"
        },
        amenity: { $push: {
            amenity: "$amenities"
            }
        }
    }
}
```

◀ **# If we group by neighborhood and push the amenities...**

The Result Is Unmanageable!


```
{
   "_id" : {
      "neighbourhood" : "Lower East Side"
   },
   "amenity" : [
      {
         "amenity" : [
            ""
         ]
      },
      {
         "amenity" : [
            "\"Smoke detector\"",
            “Wifi”,
            “\”Dry Cleaning\””
             ...,
```

The Power of $unwind

```
{
    item: “tea”,
    types: [“herbal”,
“earl grey”, “cuban”]
}
```

```
{$unwind: “$types”}
```

The Power of $unwind

```
{
    item: “tea”,
    types: [“herbal”,
“earl grey”, “cuban”]
}
```

```
{$unwind: “$types”}
```

```
{item: “tea”, types:
“herbal”}
{item: “tea”, types: “earl
grey”}
{item: “tea”, types:
“cuban”}
```

The Power of $unwind

```
},
  {
    $unwind: "$amenities"
  },
  {
    $group: {
        _id: {
            neighbourhood: "$neighbourhood"
        },
        amenities: { $addToSet: "$amenities" }
    }
  }
```

◀ **# If we $unwind the amenities**

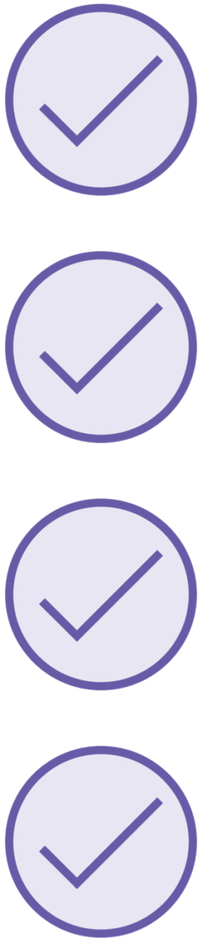
◀ **# And later we add to a new array in a unique manner**

The Power of $unwind


```
{
   "_id" : {
      "neighbourhood" : "Clifton"
   },
   "amenities" : [
      "Washer",
      "\"Dishes and silverware\"",
      "\"Smoking allowed\"",
      "\"Air conditioning\"",
      "\"Garden or backyard\"",
      "\"Pets allowed\"",
      "\"Laptop friendly workspace\"",
      "Shampoo",
      ...,
```

What Is the Most Common Amenity?

What Is the Most Common Amenity?



With **$project** and **$unwind** we get one document per amenity

What Is the Most Common Amenity?

With **$project** and **$unwind** we get one document per amenity

With **$match** we can later filter out empty ones

What Is the Most Common Amenity?

With **$project** and **$unwind** we get one document per amenity

With **$match** we can later filter out empty ones

We need to **$group** by neighborhood and amenities to get the count

What Is the Most Common Amenity?

With **$project** and **$unwind** we get one document per amenity

With **$match** we can later filter out empty ones

We need to **$group** by neighborhood and amenities to get the count

And then re-**$group** and **$sort** to get the final report

The Query up to Now

```
{$unwind: "$amenities"},
{$match: { amenities: {$ne: ‘’}}},
{$group: {
        _id: {
            neighbourhood: "$neighbourhood",
            amenity: "$amenities"
        },
        count: { $sum: 1 }
    }},
{$group : {
        _id :  "$_id.neighbourhood",
        amenities: {
            $push: {
                amenity:"$_id.amenity",
                count:"$count"
            }
        }
    }}
```

◀ **# Filter out empty amenities**

◀ **# Get count per amenity AND neighborhood**

◀ **# Remap to get per neighborhood the array of each amenity and its count!**

What Is the Most Common Amenity?


```
{
   "_id" : "Borough Park",
   "amenities" : [
      {
         "amenity" : "\"Pack 'n Play/travel crib\"",
         "count" : 1
      },
      {
         "amenity" : "\"Laptop friendly workspace\"",
         "count" : 70
      },
      {
         "amenity" : "Lockbox",
         "count" : 27
      },
...,
```

Adding $sort

```
{$unwind: "$amenities"},
{$match: { amenities: {$ne: ‘’}}},
{$group: {
        _id: {
            neighbourhood: "$neighbourhood",
            amenity: "$amenities"
        },
        count: { $sum: 1 }
    }},
{$sort: {count: -1}},
{$group : {
        _id :  "$_id.neighbourhood",
        amenities: {
            $push: {
                amenity:"$_id.amenity",
                count:"$count"
            }
        }
   }}
```

◀ **# If we sort by count here, then when grouping the amenities will already be sorted!**

What Is the Most Common Amenity?

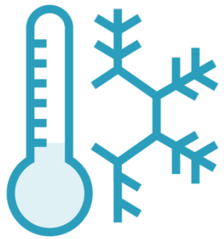

```
{
   "_id" : "Williamsburg",
   "amenities" : [
      {
         "amenity" : "Wifi",
         "count" : 3770
      },
      {
         "amenity" : "Kitchen",
         "count" : 3675
      },
      {
         "amenity" : "Heating",
         "count" : 3603
      },
      {
         "amenity" : "Essentials",
         "count" : 3572
      }, ...,
```

More Expressions: Handling Numeric Data!

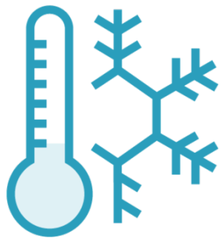
Temperature Measurements

```
db.temps.insertMany([
  { "_id" : 1, "city" : "Bakersfield", "tempsF" : [ 34.57, 81.96, 44.24 ] },
  { "_id" : 2, "city" : "Barstow", "tempsF" : [ 73.28, 9.67, 124.36 ] },
  { "_id" : 3, "city" : "San Bernadino", "tempsF" : [ 16.04, 3.25, 6.82 ] },
  { "_id" : 4, "city" : "San Francisco", "tempsF" : [ ] },
  { "_id" : 5, "city" : "New York", "tempsF" : ['12.$@#$%@', 'NaN' ] },
])
```

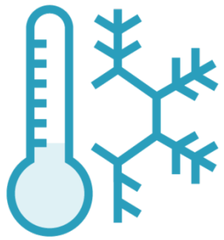
Some Things to Consider



For San Bernardino we need to add 3 degrees to the temperature



We need to truncate the values



We need to remove the faulty measurements

Filtering out Elements from an Array

```
{ $filter: {
        input: <array>,
        as: <string>,
        cond: <expression>
} }
```

input: Array field to filter elements
as: Temporary string to rename the field
cond: Condition to evaluate each element in array

Adding $filter to $project

```
db.temps.aggregate([
  {
     $project: {
        city:1,
        filteredTempsF: {
           $filter: {
              input: "$tempsF",
              as: "temp",
              cond: { $eq: [ {$type : "$$temp"} ,
"double"] }
           }
        }
     }
  }
])
```

◀ # We project a new field
◀ # That will be based on “tempsF” field

◀ # But filter non numbers!

Adding $filter to $project

```
{ "_id" : 1, "city" : "Bakersfield", "tempsF" :
[ 34.57, 81.96, 44.24 ] }
{ "_id" : 2, "city" : "Barstow", "tempsF" : [ 73.28,
9.67, 124.36 ] }
{ "_id" : 3, "city" : "San Bernadino", "tempsF" :
[ 16.04, 3.25, 6.82 ] }
{ "_id" : 4, "city" : "San Francisco", "tempsF" : [ ]
}
{ "_id" : 5, "city" : "New York", "tempsF" : [ ] }
```

◀ **# On numerical arrays is the same**

◀ **#But on “bad" arrays is a success!**

What is that **“$$”??!**

Understanding $$

{"city" : "Bakersfield",
"tempsF" : [34.57, 81.96,
44.24] }

```
filteredTempsF: {
      $filter: {
         input: "$tempsF",
         as: "temp",
         cond: { $eq: [ {$type : "$
$temp"} , "double"] }
       }
}
```

Understanding $$


```
filteredTempsF: {
      $filter: {
         input: "$tempsF",
         as: "temp",
         cond: { $eq: [ {$type : "$
$temp"} , "double"] }
       }
}
```

$ = $$CURRENT = {tempsF: [34.57, 81.96, 44.24], city: “Bakersfield”}

Understanding $$

```
{"city" : "Bakersfield",
"tempsF" : [ 34.57, 81.96,
44.24 ] }
```

```
filteredTempsF: {
      $filter: {
         input: "$tempsF",
         as: "temp",
         cond: { $eq: [ {$type : "$
$temp"} , "double"] }
       }
}
```

$ = $$CURRENT = {tempsF: [34.57, 81.96, 44.24], city: “Bakersfield”}

temp = $tempsF = [34.57, 81.96, 44.24]
INDEX = 0

Understanding $$

```
{"city" : "Bakersfield",
"tempsF" : [ 34.57, 81.96,
44.24 ] }
```

```
filteredTempsF: {
      $filter: {
         input: "$tempsF",
         as: "temp",
         cond: { $eq: [ {$type : "$
$temp"} , "double"] }
       }
}
```

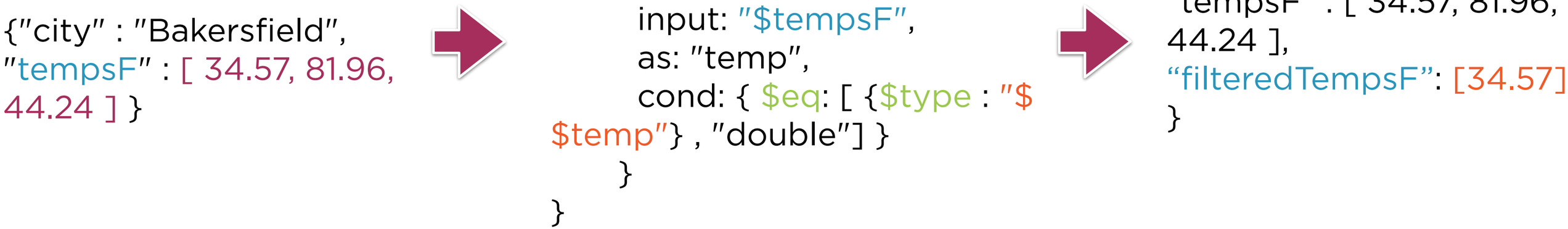
$ = $$CURRENT = {tempsF: [34.57, 81.96, 44.24], city: “Bakersfield”}

temp = $tempsF = [34.57, 81.96, 44.24]

INDEX = 0

$$temp = $$temp.$$INDEX = $temp.0 = $tempsF.0 = 34.57

Understanding $$



$ = $$CURRENT = {tempsF: [34.57, 81.96, 44.24], city: “Bakersfield”}

temp = $tempsF = [34.57, 81.96, 44.24]

INDEX = 0

$$temp = $$temp.$$INDEX = $temp.0 = $tempsF.0 = 34.57

Understanding $$


```
{"city" : "Bakersfield",
"tempsF" : [ 34.57, 81.96,
44.24 ] }
```


```
filteredTempsF: {
      $filter: {
         input: "$tempsF",
         as: "temp",
         cond: { $eq: [ {$type : "$
$temp"} , "double"] }
       }
}
```

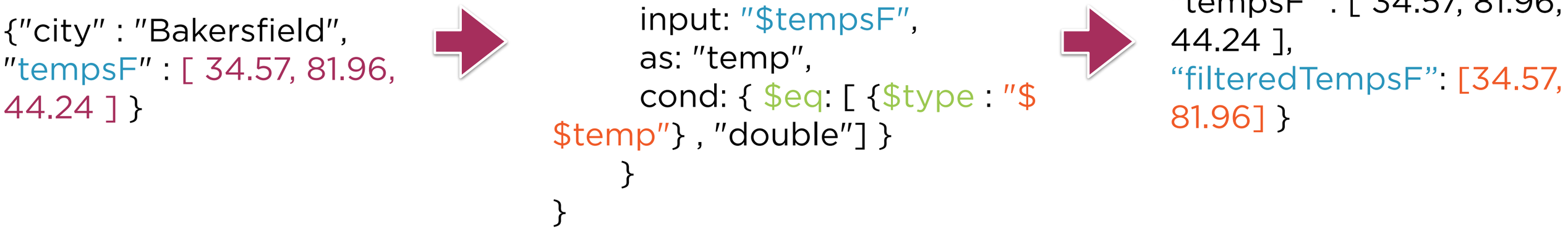
$ = $$CURRENT = {tempsF: [34.57, 81.96, 44.24], city: “Bakersfield”}

temp = $tempsF = [34.57, 81.96, 44.24]

INDEX = 1

$$temp = $$temp.$$INDEX = $temp.1 = $tempsF.1 = 81.96

Understanding $$



$ = $$CURRENT = {tempsF: [34.57, 81.96, 44.24], city: “Bakersfield”}

temp = $tempsF = [34.57, 81.96, 44.24]

INDEX = 1

$$temp = $$temp.$$INDEX = $temp.1 = $tempsF.1 = 81.96

Understanding $$

{"city" : "Bakersfield",
"tempsF" : [34.57, 81.96,
44.24] }
{ "_id" : 2, "city" :
"Barstow", "tempsF" :
[73.28, 9.67, 124.36] }

```
filteredTempsF: {
      $filter: {
         input: "$tempsF",
         as: "temp",
         cond: { $eq: [ {$type : "$
$temp"} , "double"] }
       }
}
```

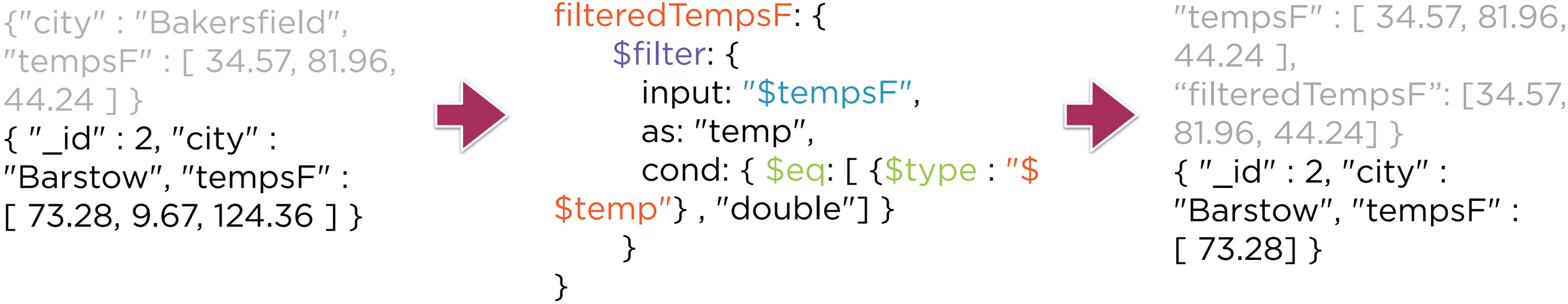
$ = $$CURRENT = {tempsF: [73.28, 9.67, 124.36], filteredTempsF: [73.28], city: "Barstow"}

temp = $tempsF = [73.28, 9.67, 124.36]

INDEX = 0

$$temp = $$temp.$$INDEX = $temp.0 = $tempsF.0 = 73.28

Understanding $$



$ = $$CURRENT = {tempsF: [73.28, 9.67, 124.36], filteredTempsF: [73.28], city: “Barstow”}

temp = $tempsF = [73.28, 9.67, 124.36]

INDEX = 0

$$temp = $$temp.$$INDEX = $temp.0 = $tempsF.0 = 73.28

Understanding $$

```
{ "_id" : 4, "city" : "San
Francisco", "tempsF" : [ ]
}
  { "_id" : 5, "city" : "New
York", "tempsF" :
['12.$@#$%@', 'NaN' ] },
```

```
filteredTempsF: {
      $filter: {
         input: "$tempsF",
         as: "temp",
         cond: { $eq: [ {$type : "$
$temp"} , "double"] }
      }
}
```

```
{ "_id" : 4, "city" : "San
Francisco", "tempsF" : [ ]
, "filteredTempsF": []}
```

$ = $$CURRENT = {tempsF: [12.$@#$%@', 'NaN'], filteredTempsF: [], city: ""New York"}

temp = $tempsF = ['12.$@#$%@', 'NaN']

INDEX = 0

$$temp = $$temp.$$INDEX = $temp.0 = $tempsF.0 = '12.$@#$%@'

Understanding $$

```
{ "_id" : 4, "city" : "San Francisco", "tempsF" : [ ] }
  { "_id" : 5, "city" : "New York", "tempsF" : ['12.$@#$%@', 'NaN' ] },
```

```
filteredTempsF: {
      $filter: {
         input: "$tempsF",
         as: "temp",
         cond: { $eq: [ {$type : "$$temp"} , "double"] }
      }
}
```

```
{ "_id" : 4, "city" : "San Francisco", "tempsF" : [ ] , "filteredTempsF": []}
  { "_id" : 5, "city" : "New York", "tempsF" : ['12.$@#$%@', 'NaN' ], filteredTempsF: [] },
```

$ = $$CURRENT = {tempsF: [12.$@#$%@', 'NaN'], filteredTempsF: [], city: ""New York"}

temp = $tempsF = ['12.$@#$%@', 'NaN']

INDEX = 0

$$temp = $$temp.$$INDEX = $temp.0 = $tempsF.0 = ~~'12.$@#$%@'~~

Filtering out Empty Arrays

```
{
    $addFields: {
        non_empty: {
            $gt: [ {$size: "$tempsF" }, 0 ]
        }
    }
},
{
    $match: {
        non_empty: true
    }
},
{
    $unset: "non_empty"
}
```

◀ **# We need a pivot field that marks if the array is empty**

◀ **# So later on we filter it out**

◀ **# And we clean up the field**

Mapping and Conditions

Mapping and Conditions

```
{ $cond: {
    if: <boolean-expression>,
    then: <true-case>,
    else: <false-case>
}}
```

```
$cond: { if: { $gte: [ "$price", 250 ] }, then:
30, else: 20 }
```

Mapping and Conditions

```
{ $cond: {
    if: <boolean-expression>,
    then: <true-case>,
    else: <false-case>
}}
```

```
$cond: { if: { $gte: [ "$price", 250 ] }, then:
30, else: 20 }
```

```
{ $map: {
    input: <expression>,
    as: <string>,
    in: <expression>
} }
```

```
{ $map: { input: “$tempsF”, as: ‘temp’, in:
{ $add: [ "$$temp", 3 ] } } }
```

Putting It Together

```
{
    $addFields: {
        adjustedtempsF: {
            $map: {
                input: "$tempsF",
                as: "temp",
                in: {
                    $cond: {
                        if: { $eq: [ "$city",  "San
Bernadino" ] },
                        then: { $add: [ "$$temp", 3 ] } ,
                        else: "$$temp"
                    }
                }
            }
        }
    }
}
```

- # For each element in tempsF array
- # We rename temporally to “temp”
- # If the city is San Bernardino
- # Add 3 to the temp, otherwise return as is

Converting to Celcius

```
{
    $project: {
        city:1,
        tempsC: {
            $map: {
                input: "$adjustedtempsF",
                as: "temps",
                in: { $trunc: { $multiply:
[ { $subtract: ["$$temps", 32 ] }, 5/9 ]}}
            }
        }
    }
}
```

```
{ "_id" : 1, "city" : "Bakersfield", "tempsC" : [ 1, 27, 6 ] }
{ "_id" : 2, "city" : "Barstow", "tempsC" : [ 22, -12, 51 ] }
{ "_id" : 3, "city" : "San Bernadino", "tempsC" : [ -7, -14, -12 ] }
```

- **# For each element in adjusted temps array**
- **# Truncate the result of applying the formula: C = 5/9 * (F - 32)**
- **# Success!!**

Some Statistics per City

```
{
    $project: {
        city:1,
        average_temperature: {
            $round: [ { $avg: "$tempsC"}, 2 ]
        },
        std_temperature: {
            $round: [ { $stdDevPop: "$tempsC"},
2 ]
        }
    }
  }
```

◀ **# As tempsC is an array, we can apply accumulators on $project, that work at document level!**

Temperature Measurements

{ "_id" : 1, "city" : "Bakersfield", "average_temperature" : 11.33, "std_temperature" : 11.26 }
{ "_id" : 2, "city" : "Barstow", "average_temperature" : 20.33, "std_temperature" : 25.75 }
{ "_id" : 3, "city" : "San Bernadino", "average_temperature" : -11, "std_temperature" : 2.94}

Demo

Demo

Demo

Get the biggest and smallest cities on each state

Demo

Get the biggest and smallest cities on each state

Delve into $group, $project and $sort!

Demo

Get the biggest and smallest cities on each state

Delve into $group, $project and $sort!

Handle numeric and array expressions

Summary

Summary

Summary

Learned about grouping in MongoDB

Summary

Learned about grouping in MongoDB

Used **$unwind** and array expressions for manipulating arrays

Summary

Learned about grouping in MongoDB

Used $unwind and array expressions for manipulating arrays

Handled numeric data