WELCOME

Aggregation Pipelines
Compute inside the database

Andrew Morgan
Senior Staff Developer Advocate
clusterdb.com

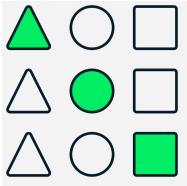
This is a very technical presentation for engineers.

```
{ Mostly : "it is examples of code " }
```

The important syntax is white, the rest is green

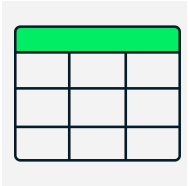
You won't remember *how* to do these things, but I want you to know you *can* do these things.

Create aggregate documents



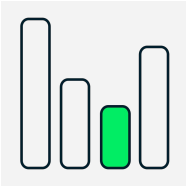
GROUP by values

Equivalent to SQL GROUP BY



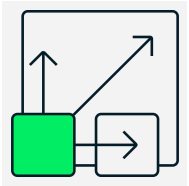
Flatten arrays

Make records more tabular and return multiple record per database record



Determine cohorts

Calculate the cohorts in a range of values

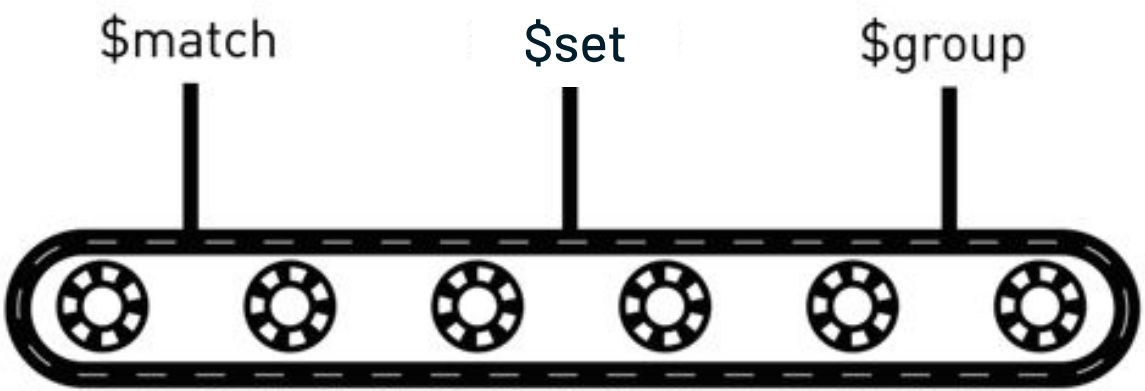


JOIN records

Perform SQL style JOINS

Each aggregation is a list of simple transformations

- Each transformation is a single step known as a stage.
- Compared to one huge SQL style statement this is
  - Easier to understand
  - Easier to debug
  - Easier for MongoDB to rewrite and optimize



Basic aggregation stages

Aggregation stage	Query API equivalent
`{$match: query}`	`find(query)`
`{$project: projection}` `{$set: {projection}}`	`find({}, projection)`
`{$sort: {order}}`	`find().sort(order)`

Database expressions

$abs
$accumulator
$acos
$acosh
$add
$addToSet
$allElementsTrue
$and
$anyElementTrue
$arrayElemAt
$arrayToObject
$asin
$asinh
$atan
$atan2
$atanh
$avg
$binarySize
$bitAnd
$bitNot
$bitOr
$bitXor
$bottom
$bottomN
$bsonSize
$ceil
$cmp
$concat
$concatArrays
$cond
$convert
$cos
$cosh
$count
$covariancePop
$covarianceSamp
$dateAdd
$dateDiff
$dateFromParts
$dateFromString
$dateSubtract
$dateToParts
$dateToString
$dateTrunc
$dayOfMonth
$dayOfWeek
$dayOfYear
$degreesToRadians
$denseRank
$derivative
$divide
$documentNumber
$eq
$exp
$expMovingAvg
$filter
$first
$firstN
$floor
$function
$getField
$gt
$gte
$hour
$ifNull
$in
$indexOfArray
$indexOfBytes
$indexOfCP
$integral
$isArray
$isNumber
$isoDayOfWeek
$isoWeek
$isoWeekYear
$last
$lastN
$let
$linearFill
$literal
$ln
$locf
$log
$log10
< Many Removed >
$trim
$trunc
$type
$unsetField
$week
$year
$zip

Expression syntax

```
 format_prices
 = {
  $project: {
      basic_price : "$base_price",
      cleaning_fee: "$cleaning_fee",
      total_price : { $add : [ "$base_price",
                          "$cleaning_fee"] }
  }
}
```

```
properties.aggregate([
  format_prices
])
```

```
{
      basic_price: 100,
      cleaning_fee: 20,
      total_price: 120
}
```

Working through an example

Aggregation in practice

For non-shared properties with warm weather, what are the typical amenities?

- Filter just Villas and Houses so amenities are likely private
- JOIN the current temperature for the city of the rental
- Filter on cities which are warm
- List the most common amenities

Properties

```
{
  name: "Parkside",
  city: "New York",
  property_type: "House",
  amenities: [
    "Laundry",
    "Pool",
    "Gym"
  ]
}
```

Temperatures

```
{
  location: "New York",
  temperatureC: 30,
  temperatureF: 86
}
```

Villas and houses

```
notShared = {
  $match: {
    property_type: {$in: [ "House", "Villa" ]}
  }
}
```

```
properties.aggregate([
  notShared
])
```

Properties

```
{
  name: "Parkside",
  city: "New York",
  property_type: "House",
  amenities: [
    "Laundry",
    "Pool",
    "Gym"
  ]
}
```

Temperatures

```
{
  location: "New York",
  temperatureC: 30,
  temperatureF: 86
}
```

Villas and houses

```
notShared = {
  $match: {
    property_type: {$in: [ "House", "Villa" ]}
  }
}
```

```
properties.aggregate([
  notShared
])
```

```
{ name: "Parkside",
  city: "New York",
  property_type: "House",
  amenities: [
    "Laundry",
    "Pool",
    "Gym"
  ]
}
```

```
{ name: "Halftime Mews",
  city: "London",
  property_type: "House",
  amenities: [
    "Pool",
    "Bike Storage",
    "Kettle"
  ]
}
```

```
{ name: "Azure Villa",
  city: "Nice",
  property_type: "Villa",
  amenities: [
    "Laundry",
    "Air Conditioning",
    "Pool"
  ]
}
```

Temperature of cities

```
joinCityTemperature = {
  $lookup: {
    from: "temperatures",
    localField: "city",
    foreignField: "location",
    as: "cityTemperature"
  }
}

properties.aggregate([
  notShared,
  joinCityTemperature
])
```

Temperatures

```
{
  location: "New York",
  temperatureC: 30,
  temperatureF: 86
}
```

```
{ name: "Parkside",
  city: "New York",
  property_type: "House",
  amenities: [
    "Laundry",
    "Pool",
    "Gym"
  ]
}
```

```
{ name: "Halftime Mews",
  city: "London",
  property_type: "House",
  amenities: [
    "Pool",
    "Bike Storage",
    "Kettle"
  ]
}
```

```
{ name: "Azure Villa",
  city: "Nice",
  property_type: "Villa",
  amenities: [
    "Laundry",
    "Air Conditioning",
    "Pool"
  ]
}
```

Temperature of cities

```
joinCityTemperature = {
  $lookup: {
    from: "temperatures",
    localField: "city",
    foreignField: "location",
    as: "cityTemperature"
  }
}

properties.aggregate([
  notShared,
  joinCityTemperature
])
```

```
{ name: "Parkside",
  city: "New York",
  property_type: "House",
  amenities: [
    "Laundry",
    "Pool",
    "Gym"
  ],
  cityTemperature: [
    { temperatureC: 30,
      temperatureF: 86,
      location: "New York"
    }
  ]
}
```

```
{ name: "Halftime Mews",
  city: "London",
  property_type: "House",
  amenities: [
    "Pool",
    "Bike Storage",
    "Kettle"
  ],
  cityTemperature: [
    {
      temperatureC: 15,
      temperatureF: 59,
      location: "London"
    }
  ]
}
```

Keep first joined doc

```
joinCityTemperature = {
  $lookup: {
    from: "temperatures",
    localField: "city",
    foreignField: "location",
    as: "cityTemperature"
  }
}

firstElement = {
  $set: {
    cityTemperature: { $first: "$cityTemperature" }
}}

properties.aggregate([
  notShared,
  joinCityTemperature,
  firstElement
])
```

```
{ name: "Parkside",
  city: "New York",
  property_type: "House",
  amenities: [
    "Laundry",
    "Pool",
    "Gym"
  ],
  cityTemperature: [
    { temperatureC: 30,
      temperatureF: 86,
      location: "New York"
    }
  ]
}
```

```
{ name: "Halftime Mews",
  city: "London",
  property_type: "House",
  amenities: [
    "Pool",
    "Bike Storage",
    "Kettle"
  ],
  cityTemperature: [
    {
      temperatureC: 15,
      temperatureF: 59,
      location: "London"
    }
  ]
}
```

Keep first joined doc

```
joinCityTemperature = {
  $lookup: {
    from: "temperatures",
    localField: "city",
    foreignField: "location",
    as: "cityTemperature"
  }
}

firstElement = {
  $set: {
    cityTemperature: { $first: "$cityTemperature" }
}}

properties.aggregate([
  notShared,
  joinCityTemperature,
  firstElement
])
```

```
{ name: "Parkside",
  city: "New York",
  property_type: "House",
  amenities: [
    "Laundry",
    "Pool",
    "Gym"
  ],
  cityTemperature: {
    temperatureC: 30,
    temperatureF: 86,
    location: "New York"
  }
}
```

```
{ name: "Halftime Mews",
  city: "London"
  a
  ]
  c
  }
}
```

```
{ name: "Azure Villa",
  city: "Paris",
  amenities: [
    "Laundry",
    "Air Conditioning",
    "Pool"
  ],
  cityTemperature: {
    temperatureC: 25,
    temperatureF: 77,
    location: "Paris"
  }
}
```

Filter on temperature

```
isWarm = {
  $match: {
    "cityTemperature.temperatureC" : { $gte: 25 }
  }
}
```

```
properties.aggregate([
  hasPool,
  joinCityTemperature,
  firstElement,
  isWarm
])
```

```
{ name: "Parkside",
  city: "New York",
  property_type: "House",
  amenities: [
    "Laundry",
    "Pool",
    "Gym"
  ],
  cityTemperature: {
    temperatureC: 30,
    temperatureF: 86,
    location: "New York"
  }
}
```

```
{ name: "Halftime Mews",
  city: "London"
  a
  ]
  c
  }
}
```

```
{ name: "Azure Villa",
  city: "Paris",
  amenities: [
    "Laundry",
    "Air Conditioning",
    "Pool"
  ],
  cityTemperature: {
    temperatureC: 25,
    temperatureF: 77,
    location: "Paris"
  }
}
```

Filter on temperature

```
isWarm = {
  $match: {
    "cityTemperature.temperatureC" : { $gte: 25 }
  }
}
```

```
properties.aggregate([
  hasPool,
  joinCityTemperature,
  firstElement,
  isWarm
])
```

```
{ name: "Parkside",
  city: "New York",
  amenities: [
    "Laundry",
    "Pool",
    "Gym"
  ],
  cityTemperature:{
    temperatureC: 30,
    temperatureF: 86,
    location: "New York"
  }
}
```

```
{ name: "Azure Villa",
  city: "Paris",
  amenities: [
    "Laundry",
    "Air Conditioning",
    "Pool"
  ],
  cityTemperature: {
    temperatureC: 25,
    temperatureF: 77,
    location: "Paris"
  }
}
```

Separate amenities

```
unwindAmenities = {
  $unwind: {
    "$amenities"
  }
}
```

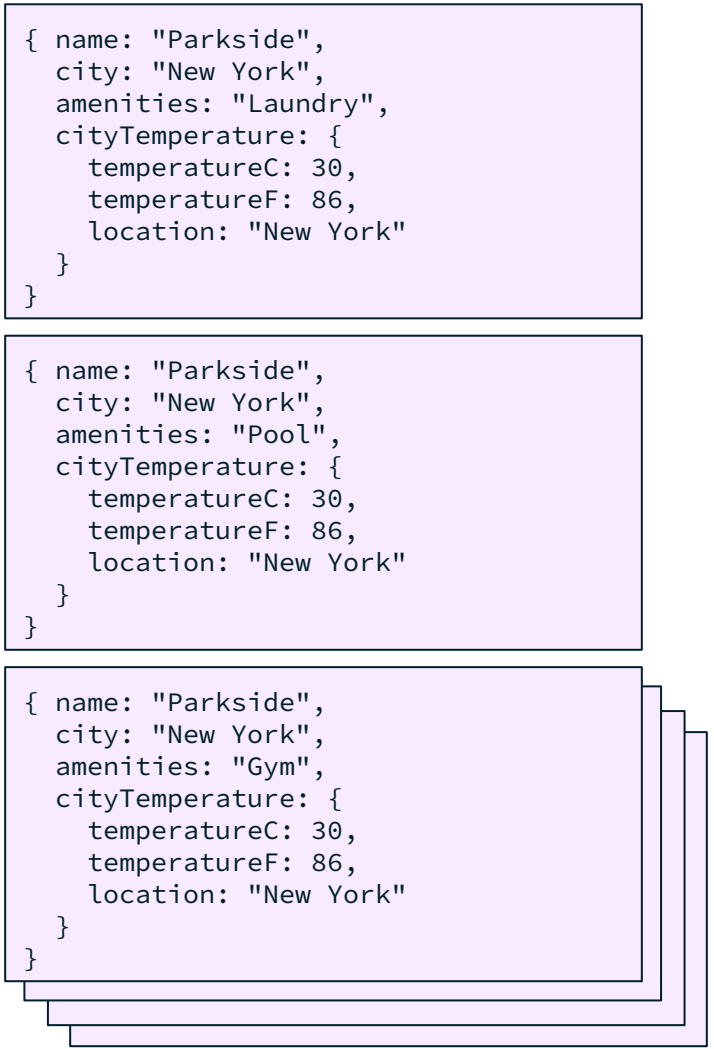
```
properties.aggregate([
  hasPool,
  joinCityTemperature,
  firstElement,
  isWarm,
  unwindAmenities
])
```

```
{ name: "Parkside",
  city: "New York",
  amenities: [
    "Laundry",
    "Pool",
    "Gym"
  ],
  cityTemperature:{
    temperatureC: 30,
    temperatureF: 86,
    location: "New York"
  }
}
```

```
{ name: "Azure Villa",
  city: "Paris",
  amenities: [
    "Laundry",
    "Air Conditioning",
    "Pool"
  ],
  cityTemperature: {
    temperatureC: 25,
    temperatureF: 77,
    location: "Paris"
  }
}
```

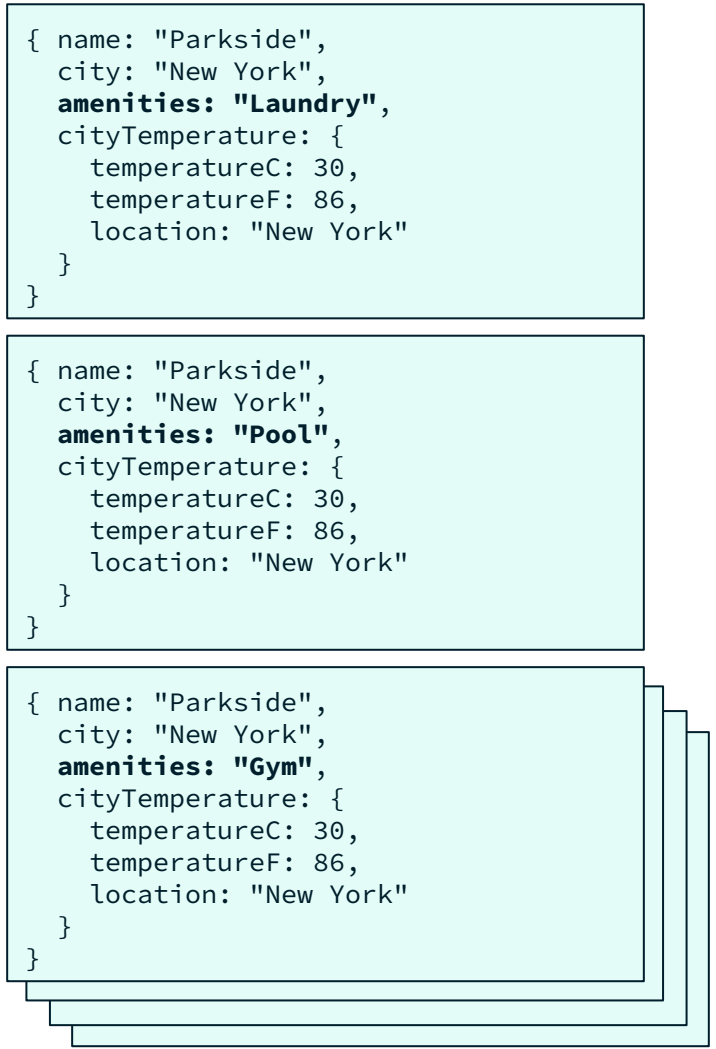
Separate amenities

```
unwindAmenities = {
  $unwind: {
    "$amenities"
  }
}


properties.aggregate([
  hasPool,
  joinCityTemperature,
  firstElement,
  isWarm,
  unwindAmenities
])
```



Most common amenities

```
mostCommonAmenity = { $sortByCount: "$amenities"  }
}

properties.aggregate([
  hasPool,
  joinCityTemperature,
  remove array,
  niceWeather,
  unwindAmenities,
  mostCommonAmenity
])
```



Most common amenities

```
mostCommonAmenity = { $sortByCount: "$amenities"  }
}

properties.aggregate([
  hasPool,
  joinCityTemperature,
  remove array,
  niceWeather,
  unwindAmenities,
  mostCommonAmenity
])
```

```
{ _id: "Laundry",
  count: 2,
}
```

```
{ _id: "Pool",
  count: 2,
}
```

```
{ _id: "Gym",
  count: 1,
}
```

```
{ _id:"Air Conditioning",
  count: 1,
}
```

Aggregation in practice

For properties with pool and decent weather, what are the popular amenities?

1. Find the properties that aren't apartments or shared rooms
2. Get the current temperature for the city of the rental
3. Filter on cities which are warm
4. Group by amenity, count and sort them

```
properties.aggregate([
  hasPool,                  // 1.
  joinCityTemperature,      // 2.
  remove array,
  niceWeather,              // 3.
  unwindAmenities,          // 4a.
  mostCommonAmenity         // 4b.
])
```

Aggregation can do anything!

> 40 Stages
> 200 Expressions

Turing Complete Functional Language

Things you can do

- Write games
- Build simulators
- Generate fractals
- Mine Bitcoin

Or just analyse the business data you need to

Grouping similar data

The stage **$group** is a very common stage similar to GROUP BY in SQL

```
groupStage = {
  $group: {
    _id: "$address.country",
    cities: { $addToSet: "$address.city" },
    average_price: { $avg: "$price" },
    total_properties: { $count: {} },
    total_beds: { $sum: "$beds" },
    total_pools: {
       $sum: {
          $cond: {
             if:
                { $in: ["Pool", "$amenities"] },
                  then: 1,
                  else: 0 }
         }
     }
 }
db.listingsAndReviews.aggregate([groupStage])
```

```
{
  _id: "United Kingdom",
  cities: [ "London",
            "Edinburgh",
            "Leeds" ],
  average_price: 180.234,
  total_properties: 654,
  total_beds: 2172,
  total_pools: 17
}
```

```
{
  _id: "USA",
  cities: [ "Atlanta",
            "New York",
            "Las Vegas",
            "Los Angeles" ],
  average_price: 207.598,
  total_properties: 2764,
  total_beds: 8651,
  total_pools: 1401
}
```

Finding Cohorts

The stage **$bucketAuto** divides the data into equal sized groups. Suppose we want *Budget*, *Standard* and *Premium* being equal sized groups based on price

```
bucketByPrice = {
  $bucketAuto: {
    groupBy: "$price",
    buckets: 3,
    output: {
      average_bedrooms: { $avg: "$bedrooms" },
      total_properties: { $count: {} }
    }
  }
}

db.listingsAndReviews.aggregate([bucketByPrice])
```

```
{   _id: { min: 9, max: 86 },
    average_bedrooms: 1.153
    total_properties: 1884
}
```

```
{   _id: { min: 86, max: 204 },
    average_bedrooms: 1.334
    total_properties: 1852
}
```

```
{  _id: { min: 204, max: 48842 },
   average_bedrooms: 1.758
   total_properties: 1819
}
```

Window functions

Looking at the temperature for a single day may give wide fluctuations

Let's take the average temperature for the day and the previous 3 days

```
window = { documents: [-3, 0] }

avgTemp = { $avg: "$temperatureC", window }

windowStage = {
  $setWindowFields: {
    partitionBy: "$location",
    sortBy: { date: 1 },
    output: { avgTemp }
  }
}

db.temperature.aggregate( [ windowStage ] )
```

```
{ location: "Paris",
  temperatureC: 19,
  date: "2024-05-24",
  avgTemp: 20.75
}
```

```
{ location: "Paris",
  temperatureC: 20,
  date: "2024-05-23",
  avgTemp: 21
}
```

```
{ location: "Paris",
  temperatureC: 25,
  date: "2024-05-22",
  avgTemp: 23
}
```

```
{ location: "Paris",
  temperatureC: 19,
  date: "2024-05-21",
  avgTemp: 19.5
}
```

Window functions

Looking at the temperature for a single day may give wide fluctuations

Let's take the average temperature for the day and the previous 3 days

```
window = { documents: [-3, 0] }

avgTemp = { $avg: "$temperatureC", window }

windowStage = {
  $setWindowFields: {
    partitionBy: "$location",
    sortBy: { date: 1 },
    output: { avgTemp }
  }
}

db.temperature.aggregate( [ windowStage ] )
```

```
{ location: "Paris",
  temperatureC: 25,
  date: "2024-05-25",
  avgTemp: 22.25
}
```

```
{ location: "Paris",
  temperatureC: 19,
  date: "2024-05-24",
  avgTemp: 20.75
}
```

```
{ location: "Paris",
  temperatureC: 20,
  date: "2024-05-23",
  avgTemp: 21
}
```

```
{ location: "Paris",
  temperatureC: 25,
  date: "2024-05-22",
  avgTemp: 23
}
```

```
{ location: "Paris",
  temperatureC: 19,
  date: "2024-05-21",
  avgTemp: 19.5
}
```

Combining result sets

The stage **$unionWith** lets you run multiple pipelines and combine the results

```
selectProperty = {$match: { propertyId: 12912 }}

flagAsArchived = { $set: { archived: true }}

queryArchivedToo  = { $unionWith: {
        col: "archived_bookings",
        pipeline: [ selectProperty,
                    flagAsArchived ] }}

projection = {$project: {
            propertyId: 1, bookingId: 1,
            fromDate: 1, toDate: 1,
            archived: 1 }}

db.bookings.aggregate([selectProperty,
                       queryArchivedToo,
                       projection])
```

```
{
  propertyId: 12912,
  bookingId: 66254,
  fromDate: "2024-03-07",
  toDate: "2024-03-14",
}
```

```
{
  propertyId: 12912,
  bookingId: 65513,
  fromDate: "2024-02-01",
  toDate: "2024-02-14",
}
```

```
{
  propertyId: 12912,
  bookingId: 39711,
  fromDate: "2022-06-07",
  toDate: "2024-06-14",
  archived: true
}
```

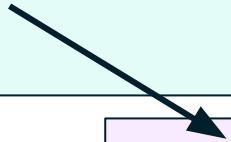
Materialized views

The stage **$merge** lets you convert a pipeline result to many updates to another collection

```
today = new Date(); today.setHours(0, 0, 0, 0);

todaysBookings = { $match: {bookingDate: {$gt: today}}}

valueOnly = {$project: { _id: "$propertyId",
                         bookings_value: "$price_paid"}}

updateStats = { $merge: {
       into: "propertyStats",
       on: "_id",
       let: { new_value: "$bookings_value" },
       whenMatched: [ { $set: { bookings_value:
                                  { $sum: [ "$$new_value",
                                       "$bookings_value" ]}}}],
       whenNotMatched: "insert"
     }}

db.bookings.aggregate([todaysBookings, valueOnly, updateStats])
```

```
{
  propertyId: 12912,
  bookingId: 65513,
  bookingDate: "2023-12-28",
  fromDate: "2024-02-01",
  toDate: "2024-02-14",
  price_paid: 123
}
```

```
{
  _id: 12912,
  bookings_value: 1000
}
```



```
{
  _id: 12912,
  bookings_value: 1123
}
```

Conclusion

It pays to know more than most.

Aggregation is very powerful

If you take it stage by stage it's easy to write

No need to extract data to process it elsewhere