# NO SQL :

```
SELECT v.VAccineName, v.AssiciatedPathology, t.month,

SUM(a.NumVaccinations),
100*SUM(a.NumVaccinations) / SUM(SUM(a.NumVaccinations)) OVER (PARTITION BY v.AssiciatedPathology, t.month),
RANK() OVER (PARTITION BY v.VaxID, t.month ORDER BY SUM(a.NumVaccinations))

FROM time t, administrations a, Vaccine v
WHERE t.TimeID = a.TimeID AND v.VAxID = a.VaxID AND t.year = 2022
GROUP BY v.VAccineName, v.AssiciatedPathology, t.month
```

FOR EACH %

↳ SEPARATELY FOR

- PARTITION :
  - OVER
  - ORDER BY
  - ROWS

PHYSICAL LEVEL : ROW COUNT
- ROWS UNBOUNDED PRECEDING
- ROWS 2 PRECEDING
- ROWS 1 PRECEDING AND 1 FOLLOWING

RANGE 2 MONTH PRECEDING

RANGE BETWEEN '<n>' DAY PRECEDING AND CURRENT ROW

LOGICAL LEVEL : IT CHECK THE LOGIC USEFUL IF MONTH MISSING

- rank() : COMPUTES THE RANK, LEAVING EMPTY SLOT AFTER A TIE
  ex. A : RANK 1, B : RANK 1, C : RANK 3
- denserank() : COMPUTES RANK, DOESN'T LEAVE EMPTY SLOT AFTER A TIE
  ex. A : RANK 1, B : RANK 1, C : RANK 2

```
SELECT Type, Weight,
  ROW_NUMBER() OVER (
    PARTITION BY Type
    ORDER BY Weight
  ) AS RowNumberWeight
FROM Item;
```

| RowNumberWeight | |
|---|---|
| 1 | Partition 1 |
| 1 | Partition 2 |
| 1 | Partition 3 |
| 1 | Subgroup 1 |
| 1 | |
| 1 | |
| 2 | |
| 2 | Subgroup 2 |
| 2 | |
| 3 | |
| 3 | Subgroup 3 |
| 3 | |

- ROW NUMBER () : IN EACH PARTITION IT ASSIGNES A PROGRESSIVE NUMBER ∀ ROW
- CUME_DIST () : ∀ PARTITION, IT ASSIGN A WEIGHT IN [0 ; 1] ACCORDINGLY TO THE
  ↳ SQL : AS ROWNUMBER N° OF VALUES THAT PRECEDE IT, CONSIDERING THE SORTING IN THE PARTITION
- NTILE (X) : ALLOWS SPLITTING EACH PARTITION IN X SUBGROUPS CONTAINING THE SAME AMOUNT OF ROWS

# MONGO DB :

## AGGREGATION :

```
db.invoices.aggregate([
    { $match: {year: {$lt: 2021}}},
    { $unwind: "$usage" },
    { $group: {
      _id: {
        service: { "$usage.service" },
        cust_city: { "$customer.city" }
      },
      max_usage_amt: { $max: "$usage.qty" }
    }},
    { $sort: {_id.service: 1, max_usage_amt: -1}}
  ])
```

VAR ON WHICH COMPUTE AGGREGATION

FIELD NAME

## $AND/OR : [{...}, {...}]

WORKS FOR LISTS TOO

```
db.movie.aggregate([
{ $match: { category: 'comedy',
            releases.contry: 'Italy' }},
{ $unwind: '$directors'},
{ $group:
{
  _id: '$directors.nationality',
  avg_score: {$avg: '$review_score'},
  n_movies: {$sum: 1}
}
},
{ $sort: {avg_score: -1}}
])
```

↳ COUNT()

## FIND :

- cursor.sort()
- cursor.count()
- cursor.forEach()
- cursor.limit()
- cursor.max()
- cursor.min()
- cursor.pretty()

```
db.movie.find(
{ // filter
categories: 'Fantasy',
directors: { ~> LIST : EXACT MATCH WORKS ON LIST
$elemMatch: {name: 'George, surname: 'Lucas'}
}
review_score: {$gt: 4}
},
{ // projection
title: 1, keywords: 1, _id: 0 // ir removes the id
}
)
```

BOOLEAN 0/1

FILTER CHECK "ITALY" IN 'COUNTRIES' LIST

```
countries: {$in : ["Italy"]}
```

## PIPELINE STAGES :

```
db.<collection>.aggregate( [
   {
     $addFields: { // add new fields to document
       totalHomework: { $sum: "$homework" } ,
       totalQuiz: { $sum: "$quiz" }
     }
```

```
db.<collection>.aggregate(
  [
    { < other aggregationpatterns >},
    {
      $count: "passing_scores"
    } // OUTPUT: { "passing_scores" : 4 }
```

```
db.<collection>.aggregate([
    // only 5 document to next stage
   { $limit : 5 }
]);
```

| Name | Description |
|---|---|
| $eq or : | 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, **including documents that do not contain the field.** |
| $nin | Matches none of the values specified in an array |

## CONDITIONAL OPERATORS :

| MySQL | MongoDB | Description |
|---|---|---|
| AND | , | Both verified |
| OR | $or | At least one verified |

# DESIGN :

## EXCHANGE

```
{
_id: ObjectId(),
stock: {
        _id: ObjectId(),
        symbol: <string>,
        market: <string>
}
day: <datetime>,
sold: {
      count: <number>,
        avg_qty: <number>,
        avg_price: <number>
        },
purchased: {
      count: <number>,
        avg_qty: <number>,
        avg_price: <number>
        },
events: [
        { ts: <datetime>,
          type: <string>,
          qty: <string>,
          price: <number>,
          trader: ObjectId()
        }
]
}
```

## LIST OF OBJECT :

< NAME > : [< TYPE >]

**STOCK**

```
{_id: ObjectId(),
symbol: <string>,
market: <string>,
category: <string>,
currency: <string>
last_price" : <number>,
}
}
```

GEO COORD

```
{
_id: ObjectId(),
name: <string>,
address: {
        street: <string>,
        number: <string|number>,
        postal_code: <number>,
        city: <string>,
        province: <string>,
        geo_ref: {type: <string>, coordinates: [ <number> ]}
}
tel: <string>,
website: <string> // optional
}
```

**Patterns used:**

- **Bucket** pattern to track events
- **Computed** pattern for the sold and purchased stats in events collection, and stats attributes in stock collection.
- **Extended reference** for the events collection to show the stock information.

- Query an Array with **Compound Filter Conditions** on the Array Elements

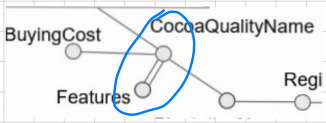
```
db.inventory.find( { dim_cm: { $gt: 15, $lt: 20 } } )
```

- Query for an Array Element that **Meets Multiple Criteria**

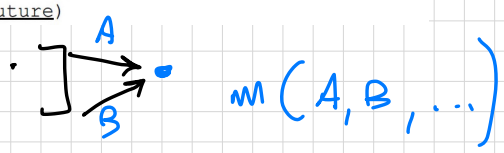
```
db.inventory.find( { dim_cm: { $elemMatch: { $gt: 15, $lt: 20 } } } )
```

Attention:

- **Compound filter**: one element of the array can satisfy the greater than 15 condition and another element can satisfy the less than 20 condition, or alternatively a single element can satisfy both
- **elemMatch**: one single element of the array **must** satisfy **both**




→ CocoaQuality(QualityId, QualityName, BuyingCost, Continent, State, Region, PlantationName)
Featured(QualityId, Feauture)




## INSERTION / UPDATE (MONGO DB):

```
db.events.updateOne(
  // filter
  { 'sensor.id': 5, 'start': new Date("2021-12-01T00:00:00.000Z")},
  // update
  {
    $push: {measures: {'ts': new Date("2021-12-01T80:00:00.000Z"), 'temperature': 19}},
    $inc: { n: 1, sum_temp: 19}}
  }
)
```

INSERT ELEMENT IN LIST
↳ n = n + 1

```
db.inventory.updateMany(
    { "qty": { $lt: 50 } },
    {
      $set: { "size.uom": "in", status: "P" },
      $currentDate: { lastModified: true }
    }
)
```

```
db.inventory.find( { item: null } )
db.inventory.find( { item : { $exists: false } } )
db.inventory.find( { item : { $stype: 10 } } )
```

→ DOC THAT NOT CONTAIN "ITEM" FIELD
→ CHECK TYPE CONTAINED IN "ITEM"

ADD "NOSQL" TO 'TAG' LIST

```
db.book.updateMany({tag: "mongodb" }, { $addToSet: {tag: "NoSQL"} })
```

## MEDIAN PRICE

All 3

```
db.listingsAndReviews.aggregate ([
{$match: {"minimum_nights": {$gte: 3 }}},
{$sort: {price: 1}},
{$group: {
  '_id': null,
  'value': {$push: '$price'}
}},
{$project: {
  _id: 1,
  "50": {
    $arrayElemAt:
    ["$value", {$floor: {$multiply: [0.50, {$size: "$value"}]}}]
  }
}}])
```

FIND IN ARRAY 'VALUE', AT INDEX {$floor...}
→ $arrayElemAt: [<LIST>, <INDEX>]
↳ round()

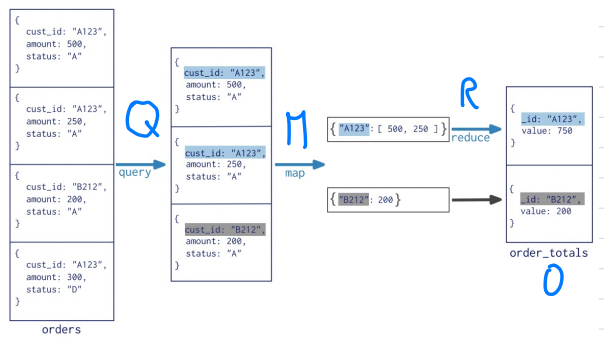
```
db.measures.aggregate([
  {$match: {
        "sensor.country": "Italy",
        "start": {$gte: new Date ("2021-01-01")},
        "start": {$lte: new Date ("2021-01-31")},
  {$addFields: {
        avg: { $divide: ["$tot", "$n" ]}
  }},
  { $match : {avg: {$gte: 15}}},
  {$project: { "sensor._id": 1, "sensor.city": 1, start: 1}}
])
```

## MAP REDUCE

```
db.<nameDB>.mapReduce(
  // map
  function(){
    emit(this.<field>, this.<field>)
  }
  // reduce
  function(key, value){
    return f(value)
  }
  // query:
  query: {<query>}
  // output
  output: <output name>
  )
```

Collection

```
db.orders.mapReduce(
    M  map     ──→  function() { emit( this.cust_id, this.amount ); },
    R  reduce  ──→  function(key, values) { return Array.sum( values ) },
                    {
    Q  query   ──→    query: { status: "A" },
    O  output  ──→    out: "order_totals"
                    }
                )
```




## INDEX CHOICE:

- INDEXING FOR DIMENSIONS
  - IN PREDICATE, ATTRIBUTE FREQ. INVOLVED
  - DOMAIN HIGH CARDINALITY: B-TREE INDEX
  - DOMAIN LOW CARDINALITY: BITMAP INDEX
- INDEXES FOR JOIN: BITMAPPED JOIN INDEXES RECOMMENDED
- INDEXES FOR GROUP BY: USE MATERIALIZED VIEW

# • DESIGN PATTERNS :

1. **APPROXIMATION** : APPROXIMATION OF VALUE UPDATE : +1 NOT USEFUL
   - ex. POPULATION COUNTER, MOVIE WEBSITE COUNTER
2. **ATTRIBUTE** : IF DOCUMENT SHARE SIMILAR FIELD ⟶ CREATE COLLECTION
   - ex. PRODUCT CATALOGUE
3. **BUCKET**: STREAM OF DATA "BUCKET" TOGETHER
   - ex. IoT, TIME SERIES
4. **COMPUTED** : STORE PRECOMPUTED DATA IN DB IN ORDER TO NOT COMPUTE VALUES EACH TIME
   - ex. REVENUE OR VIEWER, TIME SERIES DATA, PRODUCT CATALOGS
5. **DOCUMENT VERSIONING** : ~> 2+ VERSION OF SAME DOCUMENT
   IT IS USEFUL TO STORE PREVIOUS VERSIONS/SCHEMA OF DBs
   - ex. FINANCIAL INDUSTRIES, HEALTHCARE INDUSTRIES
6. **EXTENDED REFERENCE** : INSTEAD OF EMBED FULL DOCUMENT IN ANOTHER, ONLY MOST FREQUENT ACCESSED FIELD
7. **OUTLIERS** : MANAGE SITUATION WHERE ∃ SOME SPECIAL DOCUMENTS
   - ex. BESTSELLER BOOK WITH TRACKED BUYERS -> TOO BIG IN DIMENSIONS
   - SOL : FLAG DOCUMENT AS OUTLIER AND MOVE EXTRA INFO IN ANOTHER DOCUMENT
   - ex. SOCIAL MEDIA RELATIONSHIP (INFLUENCER), BOOK SALES, MOVIE REVIEWS
8. **PRE-ALLOCATION** : FIND DATA STRUCTURES WHICH BETTER ALLOW ACCESS TO DATA
   - ex. 2-DIMENSIONAL STRUCTURES, RESERVATION SYSTEM
9. **POLYMORPHIC** : GROUP OBJECT THAT ARE SIMILAR TOGETHER, BASED ON THE QUERIES TO RUN
   - ex. SINGLE VIEW APPLICATION, CROSS-COMPANY OR CROSS-UNIT USE CASE, WIDE PRODUCT CATALOG
10. **SCHEMA VERSIONING** : HAVING AN "id" FOR EACH SCHEMA VERSION A DOCUMENT
    - ex. CUSTOMER PROFILE CAN HELP THE DB HOW TO HANDLE THE DOCUMENT AND MAKE CHANGES EASILY MANAGEABLE
11. **SUBSET** : USEFUL TO CREATE SUBSET TO REDUCE RAM WORKLOAD
    - A COLLECTION CAN BE SPLITTED IN 2 :
      MOST FREQUENT DOCUMENTS /
      LEAST FREQUENT DOCUMENTS
    - ex. REVIEW OF A PRODUCT (COLLECTION SPLIT : ONLY MOST RECENT REVIEWS)
12. **TREE** : TREE HIERARCHY STRUCTURE MANAGEMENT
    - INSTEAD OF STORING ∀ NODE -> PARENT / CHILDREN
      L> STORE ∀ DOCUMENT ITS PATH

| Patterns | Catalog | Content Management | Internet of Things | Mobile | Personalization | Real-Time Analytics | Single View |
|---|---|---|---|---|---|---|---|
| Approximation | ✓ | | ✓ | ✓ | | ✓ | |
| Attribute | ✓ | ✓ | | | | | ✓ |
| Bucket | | | ✓ | | | ✓ | |
| Computed | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Document Versioning | ✓ | ✓ | | | ✓ | | ✓ |
| Extended Reference | ✓ | | | ✓ | | ✓ | |
| Outlier | | | ✓ | ✓ | ✓ | | |
| Preallocated | | | ✓ | | | ✓ | |
| Polymorphic | ✓ | ✓ | | ✓ | | | ✓ |
| Schema Versioning | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Subset | ✓ | ✓ | | ✓ | ✓ | | |
| Tree and Graph | ✓ | ✓ | | | | | |

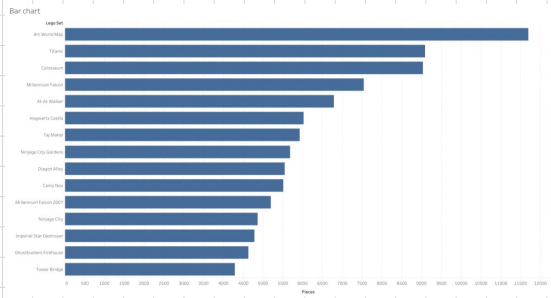
- **DATA QUALITY:**
  - ACCURACY: UNIT-MEASURE RIGHT? REASONABLE? -> DATA VS REALITY
  - COMPLETENESS: ARE ALL DATA CONSIDERED OR A SUBSET? COMPLETENESS ON X/Y AXIS?
  - CONSISTENCY: CONTRADICTION IN DATA, TIMESPANS/RANGE ON X-AXIS, ALL DATA COMPUTED SAME WAY
  - CURRENCY: ARE DATA UP TO DATE?
  - CREDIBILITY: IS THERE A SOURCE? IS THE SOURCE TRUSTWORTHY?
  - UNDERSTENDABILITY: MEANING OF DATA, DATA EXTIMATED? HOW DATA COLLECTED?
  - PRECISION: DECIMAL DIGIT ACCURATE? -> HOW NUMBER IS REPRESENTED

- **DESIGN DATA:**

| Field | Dim./Measure | Description |
|---|---|---|
| PIECES | Measure | The number of pieces in the Lego set |
| LEGO_SET | Dimension | The name of the Lego set |
| COLOR | Dimension | No real meaning |

- **DESIGN SCHEMA:**

| Schema | Details |
|---|---|
| Columns: | SUM(PIECES) |
| Rows: | LEGO_SET |
| Graph type: | Bar |
| Color: | Default |
| Size: | Default |
| Label: | Default |




STEVEN'S LAW: AREA VS LENGTH VS VOLUME COMPARISON

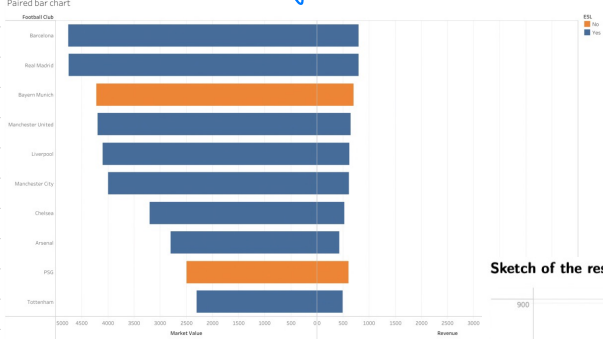
- **VISUAL**
  - PROPORTIONALITY: LF, DATA PROPORTIONAL TO GRAPH? (Y-AXIS START FROM 0?), MEASURE COMPARISON
  - UTILITY: ALL ELEMENTS CONVEY USEFUL INFO? (COLORS, SHAPES, PICTURES)
  - CLARITY: DATA IN GRAPH IDENTIFIABLE AND UNDERSTANDABLE? (LEGENDS TITLE, COLOR-BLIND, DOUBLE SCALE, COLOR MEANING, SCALE/GRAPH CONNECTION)
    - ALL ELEMENTS EFFECTIVE
    - -> } AXIS?
    - -> } NOT CLEAR ELEMENTS?

**Design schema**

| Schema | Details |
|---|---|
| Columns: | SUM(CURRENT_VALUE), SUM(REVENUE) |
| Rows: | CLUB_NAME |
| Graph type: | Bar with an inverted axis |
| Color: | SUPER_LEAGUE |
| Size: | Default |
| Label: | Default |

**Sketch of the resulting graph**

Paired bar chart



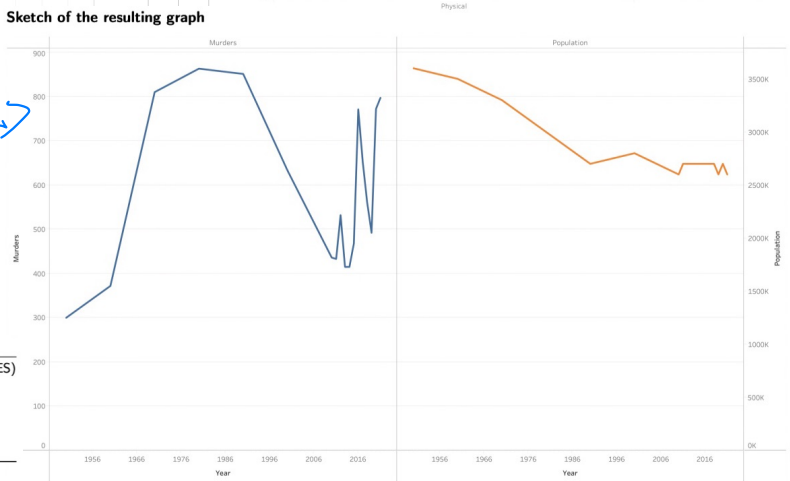

| Schema | Details |
|---|---|
| Columns: | Measure Names |
| Rows: | Measure Values [SUM(Physical), SUM(Political)] |
| Graph type: | Line |
| Color: | Continent |
| Size: | Default |
| Label: | ATTR(Continent) |

**Sketch of the resulting graph**

Slope chart

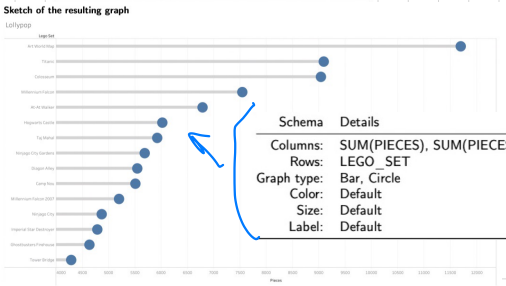

| Schema | Details |
|---|---|
| Columns: | Measure Names, YEAR(YEAR) |
| Rows: | SUM(MURDERS), SUM(POPULATION) |
| Graph type: | Line |
| Color: | Measure Names |
| Size: | Default |
| Label: | Default |

**Sketch of the resulting graph**




**Sketch of the resulting graph**

Lollypop




| Schema | Details |
|---|---|
| Columns: | SUM(PIECES), SUM(PIECES) |
| Rows: | LEGO_SET |
| Graph type: | Bar, Circle |
| Color: | Default |
| Size: | Default |
| Label: | Default |