Module 2 - Week 2

NoSQL - Part 2

TimeSeries Team

Ngày 15 tháng 7 năm 2025

I. Aggregation Framework

Nếu Mongo Query Language (MQL) - ngôn ngữ dùng để tương tác với dữ liệu trong MongoDB - là công cụ phù hợp cho các truy vấn cơ bản thì Aggregation Framework mở rộng khả năng xử lý dữ liệu lên mức nâng cao, hỗ trợ các thao tác phức tạp một cách hiệu quả và linh hoạt.

Aggregation Pipeline là một phần quan trọng trong framework đó — nó cho phép bạn truyền dữ liệu qua nhiều "stage" nối tiếp nhau, mỗi stage sẽ thực hiện một phép xử lý cụ thể.

Giới thiệu về Aggregation Pipeline

Thành phần chính trong một aggregation pipeline bao gồm các "stage" chính sau:

- Stage 1:**$match** - Lọc và giữ lại các document thỏa mãn điều kiện nhất định.
- Stage 2: **$project** - Chọn các trường cụ thể, tính toán thêm các trường mới, hoặc loại bỏ các trường không cần thiết.
- Stage 3: **$group** – Nhóm các document dựa theo một trường cụ thể để thực hiện tính toán tổng hợp như tính tổng, trung bình, số lượng,...

Lưu ý: Các stage cần được thực hiện theo thứ tự logic. Ví dụ, ta không thể thực hiện tính toán trên các trường mà trước đó ta đã thực hiện bước lọc hoặc loại bỏ.

1.1. $match

Toán tử $match được dùng để lọc các document theo phạm vi hoặc danh sách giá trị cụ thể. Giả sử chúng ta có một bộ dữ liệu về các công ty, và mục tiêu là lấy ra những công ty được thành lập trong khoảng thời gian từ năm 2005 đến năm 2010.

Cách đầu tiên, sử dụng $gte và $lte để lọc theo khoảng giá trị:

Ví dụ sử dụng $gte và $lte

```
db.companies.aggregate([
{ $match: { "founded_year": { $gte: 2005, $lte: 2010 } } }
])
```

Trong đoạn lệnh trên:

- $gte (greater than or equal): lớn hơn hoặc bằng 2005
- $lte (less than or equal): nhỏ hơn hoặc bằng 2010

Cách thứ hai, sử dụng $in để lọc theo danh sách giá trị cụ thể:

Ví dụ sử dụng $gte và $lte

```
db.companies.aggregate([
{ $match: { "founded_year": { $in: [2005, 2006, 2007, 2008, 2009, 2010] } } }
])
```

Trong trường hợp này:

Toán tử $in sẽ kiểm tra xem giá trị của `founded_year` có thuộc danh sách các năm đã cung cấp hay không, sau đó nó sẽ lọc ra các giá trị thuộc những năm trên.

1.2. $project

Toán tử $project được dùng để:

- Chọn ra những trường (fields) cần giữ lại trong mỗi document.
- Ẩn hoặc loại bỏ những trường không cần thiết.
- Tạo ra trường mới dựa trên trường có sẵn hoặc biểu thức.

Ví dụ 1: Giữ lại một số trường cụ thể

```
db.companies.aggregate(
    [{ $project: { "founded_year": 1, "category_code": 1 }
    }]
)
```

Lệnh trên sẽ chỉ giữ lại hai trường: `founded_year` và `category_code`, đồng thời ẩn tất cả các trường khác.

Ví dụ 2: Kết hợp $match và $project

```
db.companies.aggregate(
    [{ $match: { "founded_year": { $gte: 2005, $lte: 2010 }}},
    { $project: { "founded_year": 1, category_code: 1}
    }]
)
```

Lọc ra các công ty thành lập từ 2005–2010 và chỉ giữ lại hai trường cần thiết. Điều này tương tự như câu lệnh `SELECT` trong SQL

Ví dụ 3: Tạo trường mới bằng cách đổi tên

```
db.companies.aggregate(
    [{ $project:
            { "number_of_employees": 1,
            "no_of_employees": "$number_of_employees"}
    }]
)
```

Ở đây, `no_of_employees` là một trường mới được tạo, có giá trị bằng với `number_of_employees`. Đây là cách thường dùng để đổi tên trường hoặc tạo alias, tương tự như cách dùng câu lệnh `AS` trong SQL.

1.3. Arthmetic Expression Operators

MongoDB cung cấp một loạt toán tử số học cho phép bạn thực hiện các phép tính trực tiếp trong các stage như $project, $addFields, hoặc $group. Một số toán tử thường dùng gồm:

Bảng 1: Một số toán tử số học trong Aggregation Framework

Toán tử	**Mô tả**
`$add`	Cộng các biểu thức
`$subtract`	Trừ hai biểu thức
`$multiply`	Nhân các biểu thức
`$divide`	Chia hai biểu thức
`$round`	Làm tròn số theo số chữ số thập phân
`$abs`	Trị tuyệt đối
`$ceil`	Làm tròn lên (ceil)
`$floor`	Làm tròn xuống (floor)
`$mod`	Lấy phần dư (modulo)
`$pow`	Lũy thừa
`$sqrt`	Căn bậc hai
`$exp`	Lũy thừa cơ số e
`$ln`	Logarit tự nhiên
`$log`	Logarit theo cơ số tùy chọn
`$trunc`	Cắt phần thập phân (truncate)

Ví dụ 1: Sử dụng $divide để chuyển phút thành giờ

```
db.companies.aggregate(
    [{ $project:
            { "tripduration": 1,
            "tripduration_hrs": {$divide: ["$tripduration",60]}}
    }]
)
```

Ví dụ 2: Kết hợp $divide và $round để làm tròn đến 1 chữ số

```
db.companies.aggregate(
    [{ $project:
            { "tripduration": 1,
            "tripduration_hrs": {$round: [${$divide: ["$tripduration",60]},1]}}
    }]
)
```

So sánh hai cách dùng biểu thức toán học trong $match

✗ Cách sai – không dùng `$expr`

```
db.companies.aggregate(
    [{ $match:
            { "birth_year": {$gt: {$multiply: ["$tripduration",3]}}}
    }]
)
```

MongoDB sẽ không báo lỗi, tuy nhiên cũng không in ra gì vì nó không tự thực hiện được các phép tính toán expression trên **$match**

✓ Cách đúng – dùng `$expr` để so sánh biểu thức

```
db.companies.aggregate(
    [{ $match:
            {$expr:
                {$gt:
                    [{$multiply: ["$tripduration",3]},
                    "birth_year"]}
    }}]
)
```

$expr cho phép bạn nhúng các phép toán và biểu thức logic ngay trong $match.

1.4. String Expression Operators

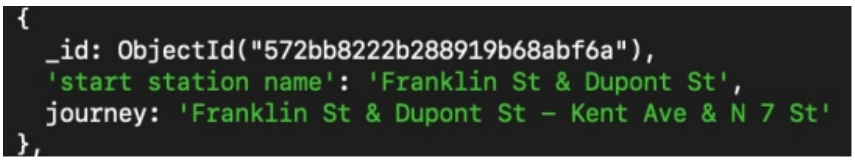
MongoDB cung cấp các toán tử xử lý chuỗi giúp thao tác, định dạng và trích xuất thông tin từ các trường văn bản. Dưới đây là một số toán tử chuỗi phổ biến:

Bảng 2: Một số toán tử chuỗi trong Aggregation Framework

Toán tử	**Mô tả**
`$concat`	Nối nhiều chuỗi lại với nhau
`$toUpper`	Chuyển chuỗi sang chữ in hoa
`$toLower`	Chuyển chuỗi sang chữ thường
`$split`	Tách chuỗi thành mảng theo ký tự phân cách
`$indexOfBytes`	Vị trí xuất hiện của chuỗi con
`$substrBytes`	Trích xuất chuỗi con từ vị trí và độ dài
`$ltrim / $rtrim`	Loại bỏ khoảng trắng đầu/cuối chuỗi
`$regexMatch`	Kiểm tra chuỗi có khớp với regex không
`$regexFind`	Trả về kết quả đầu tiên khớp với regex
`$toString`	Chuyển đổi sang kiểu chuỗi
`$dateToString`	Chuyển kiểu ngày sang chuỗi định dạng tùy chọn
`$dateFromString`	Chuyển chuỗi định dạng ngày sang kiểu ngày (ISODate)

Ví dụ sử dụng $concat

```
db.trips.aggregate(
    [{$project:
            {"start station name": 1,
            "journey": {$concat: ["$start station name",
                                  " - ",
                                  "$end station name"]
}}}])
```



Hình 1: Output của toán tử $concat

1.5. Date Expression Operators

MongoDB hỗ trợ nhiều toán tử xử lý ngày tháng để trích xuất, chuyển đổi và thao tác với dữ liệu thời gian. Dưới đây là một số toán tử ngày tháng thường dùng trong Aggregation Framework:

Bảng 3: Một số toán tử ngày tháng trong Aggregation Framework

Toán tử	Mô tả
`$dateAdd`	Cộng thêm thời gian vào một ngày cụ thể
`$dateDiff`	Tính khoảng cách giữa hai thời điểm theo đơn vị cụ thể
`$dateFromParts`	Tạo ngày từ các thành phần (năm, tháng, ngày, giờ...)
`$dateFromString`	Chuyển chuỗi ngày sang định dạng ngày ISO
`$dateToString`	Chuyển kiểu ngày sang chuỗi theo định dạng tùy chỉnh
`$isoWeek`	Trả về số tuần ISO (theo chuẩn quốc tế)
`$month`	Trích xuất số tháng từ trường ngày
`$dayOfMonth`	Trích xuất ngày trong tháng
`$dayOfWeek`	Trích xuất thứ trong tuần
`$hour`, `$second`	Trích xuất giờ, giây từ thời gian

Ví dụ sử dụng $month

```
db.trips.aggregate(
    [{$project:
        {"start time": 1,
        "month_no": { $month: "$start time" }}
  }])
```

1.6. Comparison Expression Operators

Các toán tử so sánh được sử dụng để đánh giá biểu thức logic giữa hai giá trị hoặc hai trường dữ liệu. Chúng có thể được dùng trong các stage như `$match`, `$project`, hoặc kết hợp trong `$expr` để biểu diễn biểu thức động.

Bảng 4: Một số toán tử so sánh trong Aggregation Framework

Toán tử	Mô tả
`$eq`	So sánh bằng (equal)
`$ne`	So sánh khác (not equal)
`$gt`	Lớn hơn (greater than)
`$lt`	Nhỏ hơn (less than)
`$gte`	Lớn hơn hoặc bằng
`$lte`	Nhỏ hơn hoặc bằng
`$cmp`	Trả về -1, 0, 1 tương ứng với so sánh nhỏ hơn, bằng, lớn hơn

field:[operator, value]

Ví dụ 1: So sánh với giá trị trong $match

```
db.trips.aggregate([
  { $match: { "tripduration": { $gt: 100 }}}]
)
```

operator:[exp1, exp2]

Ví dụ 2: So sánh với biểu thức trong $expr

```
db.trips.aggregate([
  { $match:
        { $expr: { $gt: ["$tripduration", 100] } }
}])
```

Ví dụ 3: So sánh trong $project để tạo trường logic

```
db.trips.aggregate([
  {
    $project:
        {"tripduration": 1,
        "over100flag": { $gt: ["$tripduration", 100] }}
}])
```

Ouput trả về cho `over100flag` là True/ False (True đối với `tripduration` lớn hơn 100, False đối với `tripduration` nhỏ hơn 100

1.7. Array Expression Operators

MongoDB cung cấp nhiều toán tử để thao tác với dữ liệu dạng mảng. Những toán tử này giúp kiểm tra, truy xuất, nối, lọc hoặc xử lý các phần tử trong mảng một cách linh hoạt trong Aggregation Pipeline.

Bảng 5: Một số toán tử xử lý mảng trong Aggregation Framework

Toán tử	**Mô tả**
`$isArray`	Kiểm tra xem giá trị có phải là mảng không (trả về true/-false)
`$arrayElemAt`	Truy xuất phần tử tại chỉ số cụ thể trong mảng
`$first`	Lấy phần tử đầu tiên của mảng
`$last`	Lấy phần tử cuối cùng của mảng
`$size`	Trả về số lượng phần tử trong mảng
`$in`	Kiểm tra giá trị có tồn tại trong mảng không
`$concatArrays`	Nối nhiều mảng lại thành một mảng lớn
`$range`	Tạo mảng số nguyên theo khoảng
`$slice`	Cắt ra một phần của mảng
`$reverseArray`	Đảo ngược thứ tự phần tử trong mảng
`$reduce`	Rút gọn mảng theo biểu thức
`$zip`	Gộp các mảng thành mảng các cặp
`$indexOfArray`	Trả về vị trí phần tử trong mảng
`$objectToArray`	Chuyển đổi tượng thành mảng các cặp key-value

Ví dụ 1: sử dụng $isArray

```
db.grades.aggregate([
  {$project: {
      "scores": 1,
      "is_array": { $isArray: "$scores" }}
}])
```

Trường `is_array` sẽ trả về True nếu $score là mảng, ngược lại là False.

Ví dụ 2: sử dụng $first

```
db.grades.aggregate([
  {$project: {
      "first_element": { $first: "$scores" }}
}])
```

Trường first_element sẽ chứa giá trị đầu tiên trong mảng $scores.

1.8. Conditional Expression Operators

Các toán tử điều kiện trong Aggregation Pipeline cho phép kiểm tra và phân nhánh logic tương tự như câu lệnh `if - else` trong các ngôn ngữ lập trình. Chúng rất hữu ích khi muốn xử lý dữ liệu tùy theo điều kiện.

Bảng 6: Một số toán tử điều kiện trong Aggregation Framework

Toán tử	**Mô tả**
`$cond`	Cấu trúc if-then-else để thực hiện phân nhánh logic
`$ifNull`	Trả về giá trị thay thế nếu biểu thức là `null` hoặc không tồn tại
`$switch`	Cho phép kiểm tra nhiều điều kiện phân nhánh phức tạp hơn

Ví dụ 1: Dùng `$ifNull` để gán giá trị mặc định nếu trường bị null

```
db.companies.aggregate(
    [{$project:
        {"number_of_employees":
                {$ifNull: ["$number_of_employees", 0]}}
}])
```

Nếu `number_of_employees` không có giá trị hoặc bị null, MongoDB sẽ thay thế bằng 0.

Ví dụ 2: Dùng `$cond` để phân loại công ty theo quy mô nhân sự

```
db.companies.aggregate(
    [{$project:
        {"number_of_employees": 1,
        "size_employees":
        {$cond:
            {if: { $gt: ["$number_of_employees", 1000] },
            then: "large",
            else: "not large"}}}
}])
```

Trường `size_employees` sẽ chứa *"large"* nếu số nhân viên > 1000, ngược lại là *"not large"*.

1.9. $addFields

Toán tử `$addFields` được dùng để thêm hoặc tính toán các trường mới trong document mà không loại bỏ các trường cũ — điều này khác biệt với `$project` vì nếu ta thực hiện tính toán tạo một trường mới **với `$project`, các trường khác sẽ bị loại bỏ trừ khi bạn ghi rõ giữ lại nó.**

`$addFields` có thể sử dụng trực tiếp các toán tử biểu thức như `$divide`, `$round`, v.v. mà không cần bao bên ngoài bởi `$expr` như trong `$match`.

Ví dụ: Tạo trường `tripduration_hrs` từ `tripduration` bằng cách chia cho 60

```
db.trips.aggregate(
    [{$addFields:
        {"tripduration_hrs": { $divide: ["$tripduration", 60] }}
}])
```

Trường mới `tripduration_hrs` được thêm vào mỗi document, chứa thời lượng chuyến đi tính theo giờ, trong khi các trường cũ vẫn được giữ nguyên.

1.10. Cursor Stages

Cursor Stages là các stage chuyên dùng để thao tác với kết quả đầu ra của pipeline như sắp xếp, đếm, phân trang hoặc giới hạn số lượng document trả về. Đây là những bước thường xuất hiện ở cuối pipeline để chuẩn bị dữ liệu cho việc hiển thị hoặc xuất ra giao diện người dùng.

Một số toán tử cursor thường dùng gồm:

- `$sort` – Sắp xếp kết quả theo một hoặc nhiều trường.
- `$limit` – Giới hạn số lượng document trả về.
- `$skip` – Bỏ qua một số lượng document đầu tiên.
- `$count` – Đếm số lượng document còn lại sau các bước trước.

Các stage này thường được kết hợp để thực hiện phân trang, truy vấn hiệu suất cao, hoặc hiển thị top-N bản ghi.

Ví dụ 1: $sort sắp xếp các chuyến đi theo thời lượng giảm dần:

```
db.trips.aggregate(
    [{ $sort: { "tripduration": -1 }
}])
```

Ví dụ 2: $sort + $limit lấy 2 chuyến đi có thời lượng dài nhất:

```
db.trips.aggregate([
  { $sort: { "tripduration": -1 } },
  { $limit: 2 }
])
```

Ví dụ 3: $sort + $limit + $skip lấy 2 chuyến đi dài nhất

```
db.trips.aggregate([
  { $sort: { "tripduration": -1 } },
  { $limit: 2 },
  { $skip: 1 }
])
```

Ví dụ 3: $match + $count đếm tổng số chuyến có thời lượng lớn hơn 100 phút:

```
db.trips.aggregate([
  { $match: { "tripduration": { $gt: 100 } } },
  { $count: "total" }
])
```

Lưu ý: `$count` sẽ trả về đúng một document duy nhất dưới dạng `{"total": số lượng}`, còn `$limit` và `$skip` thường dùng kết hợp để phân trang dữ liệu.

1.11. $group

Stage `$group` trong Aggregation Pipeline có chức năng tương tự như câu lệnh `GROUP BY` trong SQL. Nó dùng để nhóm các document theo một trường nhất định và tính toán các giá trị tổng hợp trên từng nhóm.

- `_id`: Biểu thức định nghĩa điều kiện nhóm — thường là một trường cụ thể như `"$category_code"`.
- Các trường còn lại là những biểu thức tính toán (accumulators) như: `$sum`, `$avg`, `$min`, `$max`, `$push`, `$addToSet`, v.v.

Template `$group`

```
{
  $group: {
    "_id": <group_key_expression>,
    "field1": { <accumulator1>: <expression1> },
    "field2": { <accumulator2>: <expression2> },
    ...
  }
}
```

Ví dụ 1: Nhóm theo category

```
db.companies.aggregate([
  { $group: { "_id": "$category_code" } }
])
```

Nhóm tất cả các document theo trường `category_code`, nhưng chưa có tính toán nào khác.

Ví dụ 2: Tính trung bình nhân viên mỗi nhóm

```
db.companies.aggregate([
  {
    $group: {
      "_id": "$category_code",
      "avg employees": { $avg: "$number_of_employees" }
    }
  }
])
```

Với mỗi nhóm theo `category_code`, tính số nhân viên trung bình.

Ví dụ 3: Thêm bước sắp xếp theo số nhân viên tăng dần

```
db.companies.aggregate([
  {
    $group: {
      "_id": "$category_code",
      "avg employees": { $avg: "$number_of_employees" }
    }
  },
  { $sort: { "avg employees": 1 } }
])
```

Sau khi nhóm và tính trung bình, kết quả được sắp xếp theo số nhân viên trung bình tăng dần.

Lưu ý: Trường `_id` là bắt buộc trong `$group`, vì nó đại diện cho mỗi nhóm. Nếu bạn muốn gộp toàn bộ dataset thành 1 nhóm duy nhất, chỉ cần đặt `"_id": null`.

1.12. $bucket and $bucketAuto

`$bucket` và `$bucketAuto` là hai stage trong Aggregation Pipeline dùng để phân loại dữ liệu liên tục thành các "khoảng nhóm" (buckets), tương tự như cách bạn chia dữ liệu thành các bins trong biểu đồ histogram.

`$bucket`

Toán tử `$bucket` cho phép bạn tự định nghĩa các "ranh giới" phân nhóm cụ thể.

Template `$bucket`

```
$bucket: {
  groupBy: <expression>,
  boundaries: [lowerBound1, lowerBound2, ..., upperBound],
  default: <"default bucket label">,
  output: {
    <outputField1>: { <accumulator>: <expression> },
    <outputField2>: { <accumulator>: <expression> },
    ...
  }
}
```

Ví dụ: Chia `tripduration` thành các bucket theo boundaries

```
db.trips.aggregate([
  {$bucket: {
      groupBy: "$tripduration",
      boundaries: [0, 100, 1000, 10000],
      default: "other",
      output: {
        "avg duration": { $avg: "$tripduration" },
        "count": { $sum: 1 }}}
}])
```

Giải thích:

- `groupBy`: Trường dùng để nhóm dữ liệu theo khoảng.
- `boundaries`: Danh sách các giá trị ranh giới cho các bucket.
- `default`: (tùy chọn) – Tên bucket dành cho các giá trị nằm ngoài khoảng.
- `output`: (tùy chọn) – Cho phép tính toán như `$avg`, `$sum` trong từng bucket.

`$bucketAuto`

Khác với `$bucket`, toán tử `$bucketAuto` sẽ tự động xác định các khoảng bucket sao cho phân phối dữ liệu tương đối đều (equal distribution of documents).

Template `$bucketAuto`

```
$bucketAuto: {
  groupBy: <expression>,
  buckets: <number>,
  output: {
    <outputField1>: { <accumulator>: <expression> },
    <outputField2>: { <accumulator>: <expression> },
    ...
  }
}
```

Ví dụ: Chia `tripduration` thành số bucket mong muốn

```
db.trips.aggregate([
  {$bucketAuto: {
      groupBy: "$tripduration",
      buckets: 5,
      output: {
        "avg duration": { $avg: "$tripduration" },
        "count": { $sum: 1 }}}
}])
```

Giải thích:

- `buckets`: Số lượng bucket bạn muốn tạo.
- MongoDB sẽ chia dữ liệu thành số bucket tương ứng sao cho mỗi bucket có số lượng document gần bằng nhau.
- "count": $sum: 1 ở đây nghĩa là mỗi document ta cộng thêm 1 vào giá trị $sum

1.13. $facet

Toán tử `$facet` trong Aggregation Pipeline cho phép chạy nhiều pipeline nhỏ (sub-pipelines) song song trên cùng một tập dữ liệu đầu vào. Kết quả sẽ trả về dưới dạng một document, trong đó mỗi trường tương ứng với kết quả từ một pipeline con. Dùng khi bạn muốn thực hiện nhiều phân tích đồng thời trên cùng dữ liệu — ví dụ như vừa nhóm thủ công bằng `$bucket`, vừa nhóm tự động bằng `$bucketAuto`. Hoặc khi bạn muốn tạo ra các pipeline song song độc lập nhau, pipeline 1 tính sum, pipeline 2 tính bình phương, pipeline 3 tính căn bậc hai

Template `$facet`

```
$facet: {
  subPipeline1: [ <stage1>, <stage2>, ... ],
  subPipeline2: [ <stage1>, <stage2>, ... ],
  ...
}
```

Ví dụ 1: Phân loại thủ công và tự động đồng thời

```
db.trips.aggregate([
  {$facet: {
      "BucketManual": [
        {$bucket: {
            groupBy: "$tripduration",
            boundaries: [0, 100, 1000],
            default: "other"
          }
        }
      ],
      "BucketAuto": [
        {$bucketAuto: {
            groupBy: "$tripduration",
            buckets: 5 }}]}
}])
```

Giải thích:

- `BucketManual`: Pipeline con dùng `$bucket` để phân loại dựa trên khoảng thủ công.
- `BucketAuto`: Pipeline con dùng `$bucketAuto` để phân chia thành 5 nhóm tự động.

Kết quả trả về là một document có hai trường: `BucketManual` và `BucketAuto`, mỗi trường chứa mảng kết quả của pipeline tương ứng.

1.14. $sortByCount

Toán tử `$sortByCount` là một shortcut tiện lợi trong Aggregation Pipeline, tương đương với việc kết hợp `$group` và `$sort` để đếm số lượng và sắp xếp giảm dần theo tần suất xuất hiện của một trường cụ thể.

Template `$sortByCount`

```
$sortByCount: <expression>
($group = $sort)
```

Ví dụ 1: `$sortByCount` đếm số lượng người dùng theo loại tài khoản:

```
db.trips.aggregate([
  { $sortByCount: "$usertype" }
])
```

Kết quả trả về là danh sách các nhóm theo trường `usertype`, kèm theo số lượng "count", và được sắp xếp giảm dần theo số lượng.

Cách viết tương đương bằng `$group` + `$sort`:

Viết tương đương thủ công

```
db.trips.aggregate([
  {$group: {
      "_id": "$usertype",
      "count": { $sum: 1 }}},
  { $sort: { "count": -1 } }
])
```

Lưu ý:

- `$sortByCount` chỉ dùng được trực tiếp trong pipeline, không dùng được trong các stage con như trong `$facet`.
- Nếu cần tùy biến phức tạp hơn, nên dùng kết hợp `$group` + `$sort`.

1.15. User Variable

Trong MongoDB Aggregation, bạn có thể gán từng *stage* của pipeline vào các biến riêng biệt trước khi thực thi truy vấn. Cách làm này giúp đoạn mã dễ đọc, dễ bảo trì, và dễ mở rộng nếu cần thêm stage mới sau này.

Tình huống ví dụ: Truy vấn yêu cầu:

- Tìm tất cả công ty được thành lập từ 2005 đến 2010.

- Giữ lại trường `category_code`.
- Đếm số lượng công ty theo từng loại ngành (`category_code`).

Cách viết trực tiếp:

Truy vấn liền mạch

```
db.companies.aggregate([
  { $match: { "founded_year": { $gte: 2005, $lte: 2010 } } },
  { $project: { "category_code": 1 } },
  { $sortByCount: "$category_code" }
])
```

Cách viết tách biến - sử dụng user-defined stage variables:

Gán từng stage vào biến riêng

```
stage1 = { $match: { "founded_year": { $gte: 2005, $lte: 2010 } } };
stage2 = { $project: { "category_code": 1 } };
stage3 = { $sortByCount: "$category_code" };

db.companies.aggregate([ stage1, stage2, stage3 ]);
```

Lợi ích:

- Dễ đọc và hiểu từng bước logic của pipeline.
- Dễ tái sử dụng và mở rộng — nếu cần thêm `$group` hay `$match` mới, chỉ cần định nghĩa thêm biến.

1.16. System Variable

MongoDB cung cấp một số **biến hệ thống** (system variables) dùng trong Aggregation Pipeline để lấy thông tin hệ thống tại thời điểm truy vấn như thời gian, timestamp đồng bộ theo cluster, hoặc truy cập root document.

Một số biến hệ thống phổ biến:

- `$$NOW`: Ngày giờ hiện tại tại thời điểm thực hiện truy vấn.
- `$$CLUSTER_TIME`: Thời điểm đồng bộ của cluster (phù hợp trong môi trường phân tán).
- `$$ROOT`: Tham chiếu toàn bộ document gốc tại thời điểm hiện tại trong pipeline.

Gắn thời gian hệ thống vào kết quả

```
db.grades.aggregate([
  {$project: {
      "student_id": 1,
      "scores": 1,
      "datetime": "$$NOW",
      "timestamp": "$$CLUSTER_TIME"}
}])
```

Ứng dụng thực tế:

- Theo dõi thời điểm xử lý pipeline.
- Kết hợp với các điều kiện lọc ngày (`$gt: $$NOW`) trong các hệ thống thời gian thực.
- Truy xuất dữ liệu gốc khi dùng `$replaceRoot` hoặc `$mergeObjects`.

II. Indexes

2.1. Indexes

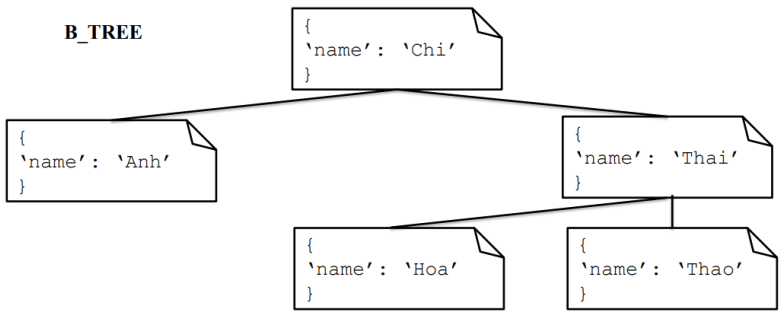
Trong MongoDB, **Index** (chỉ mục) là một cấu trúc dữ liệu đặc biệt giúp tăng hiệu suất truy vấn bằng cách cho phép truy cập nhanh hơn vào các document trong collection. Nếu không có index, MongoDB phải thực hiện `COLLSCAN` — tức là quét toàn bộ collection — để tìm các document phù hợp.

Truy vấn đơn giản không có index tìm document có trường `name` là "Thai":

```
db.collection.find({ "name": "Thai" })
```

Nếu không có index trên trường `name`, MongoDB sẽ quét từng document một để tìm giá trị tương ứng.

Cấu trúc dữ liệu chỉ mục: MongoDB sử dụng cấu trúc cây B-Tree để lưu trữ index. B-Tree là viết tắt của Balanced Tree. Tương tự như thuật toán cây cân bằng, cấu trúc này giúp MongoDB tối ưu hóa tìm kiếm. Dưới đây là minh họa đơn giản:



Hình 2: Cấu trúc chỉ mục dạng B-Tree trong MongoDB

Phân bố theo cấu trúc cây cân bằng giúp MongoDB chỉ rẽ nhánh đến những giá trị nó cần tìm, tiết kiệm thời gian truy vấn và xử lí dữ liệu.

Tạo chỉ mục:

Tạo và xoá chỉ mục

```
db.collection.createIndex({
  "first_name": 1,
  "last_name": -1
})

db.collection.dropIndex("last_name")
```

Kiểm tra hiệu suất với chỉ mục: Bạn có thể kiểm tra quá trình thực thi và hiệu suất truy vấn bằng '.explain("executionStats")':

Truy vấn không tạo index

```
db.companies.aggregate([
  { $match: { number_of_employees: { $gt: 1000 } } }
]).explain("executionStats")
```

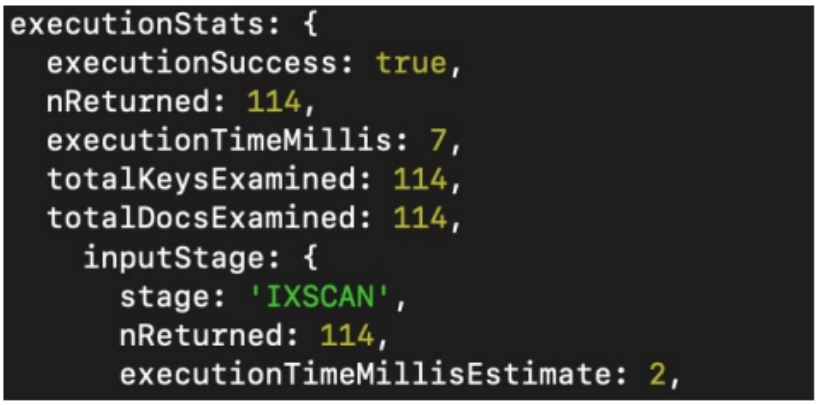
Not use Index

```
executionStats: {
  executionSuccess: true,
  nReturned: 114,
  executionTimeMillis: 87,
  totalKeysExamined: 0,
  totalDocsExamined: 9500,
  executionStages: {
    stage: 'COLLSCAN',
    filter: { number_of_employees: { '$gt': 1000 } },
    nReturned: 114,
```

Hình 3: Output của '.explain("executionStats") khi không tạo index

Truy vấn tạo index

```
db.companies.createIndex({ number_of_employees: 1 })
```



Hình 4: Output của '.explain("executionStats") khi tạo index

Bạn thấy thời gian execution của không sử dụng index từ 87 mili giây giảm còn 7 mili giây khi ta sử dụng index.

Lợi ích chính của Index:

- Tăng tốc độ truy vấn đáng kể.
- Giảm chi phí tính toán khi xử lý các truy vấn phức tạp.
- Tránh `COLLSCAN`, giúp hệ thống hoạt động hiệu quả hơn.

2.2. Compound Indexes

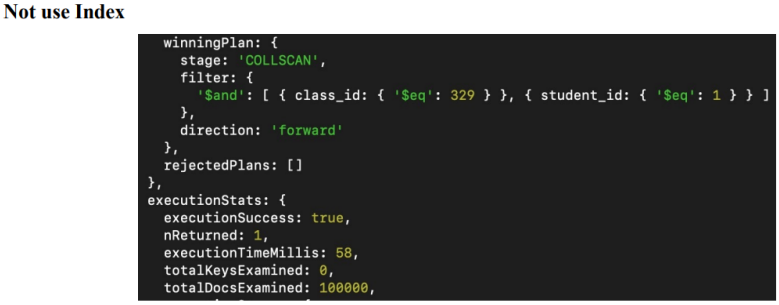
Compound Index là loại chỉ mục bao gồm nhiều trường trong một index duy nhất. Điều này rất hữu ích trong các truy vấn mà bạn lọc theo nhiều trường cùng lúc.

Ví dụ: Giả sử bạn thường xuyên thực hiện truy vấn theo cặp `student_id` và `class_id`:

Truy vấn không có compound index

```
db.grades.find({ "student_id": 1, "class_id": 329 })
  .explain("executionStats")
```

Kết quả sẽ sử dụng `COLLSCAN` nếu không có index phù hợp.



Hình 5: Output của '.explain("executionStats") khi không tạo index

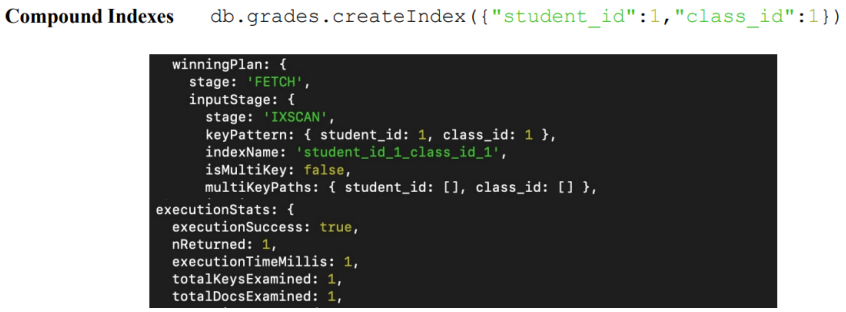
Thời gian execution là 58 mili giây

Tạo Compound Index:

Tạo compound index và thực hiện truy vấn

```
db.grades.createIndex({ "student_id": 1, "class_id": 1 })

db.grades.find({ "student_id": 1, "class_id": 329 })
  .explain("executionStats")
```



Hình 6: Output của '.explain("executionStats") khi tạo index

Thời gian execution là 1 mili giây

Lưu ý: Compound index hoạt động hiệu quả theo thứ tự các trường được định nghĩa. Ví dụ:

- Truy vấn theo `student_id` — sử dụng được index.
- Truy vấn theo `class_id` — không sử dụng được index nếu nó không phải trường đầu tiên tạo compound index.

Kiểm tra hiệu suất khi gọi `student_id`

```
db.grades.find({ "student_id": 1 }).explain("executionStats")
```

```
db.grades.find({"student_id":1})
.explain("executionStats")
```

```
  winningPlan: {
    stage: 'FETCH',
    inputStage: {
      stage: 'IXSCAN',
      keyPattern: { student_id: 1, class_id: 1 },
      indexName: 'student_id_1_class_id_1',
,,
executionStats: {
  executionSuccess: true,
  nReturned: 10,
  executionTimeMillis: 0,
  totalKeysExamined: 10,
  totalDocsExamined: 10,
```

Hình 7: Output của '.explain("executionStats") khi gọi `student_id`

Kiểm tra hiệu suất khi gọi `class_id`

```
db.grades.find({ "class_id": 329 }).explain("executionStats")
```

```
db.grades.find({"class_id":329})
.explain("executionStats")
```

```
  winningPlan: {
    stage: 'COLLSCAN',
    filter: { class_id: { '$eq': 329 } },
    direction: 'forward'
  },
  rejectedPlans: []
},
executionStats: {
  executionSuccess: true,
  nReturned: 203,
  executionTimeMillis: 57,
  totalKeysExamined: 0,
  totalDocsExamined: 100000,
```

Hình 8: Output của '.explain("executionStats") khi gọi `class_id`

Kết quả sử dụng `COLLSCAN` vì `class_id` không phải là trường đầu tiên khi ta tạo compound index

Tóm lại: Compound index rất hiệu quả trong các truy vấn nhiều trường, nhưng nên cẩn thận với thứ tự trường vì nó ảnh hưởng đến khả năng tối ưu hoá truy vấn.

2.3. Partial Indexes

Partial Index (chỉ mục một phần) là loại index chỉ áp dụng cho các document thỏa mãn điều kiện nhất định (được xác định bởi `partialFilterExpression`). Điều này giúp tiết kiệm bộ nhớ và tối ưu truy vấn khi bạn chỉ quan tâm tới một phần dữ liệu cụ thể.

Template tạo Partial Index

```
db.collection.createIndex(
  { field1: 1, field2: -1, ... },
  {
    partialFilterExpression: {
      fieldX: { <toán tử>: <giá trị> }
    }
  }
)
```

Chỉ tạo index cho các chuyến đi có thời lượng trên 100 phút.

```
db.trips.createIndex(
  { "tripduration": 1 },
  { partialFilterExpression: { "tripduration": { $gt: 100 } } }
)
```

Truy vấn thỏa điều kiện partial index — sử dụng index

```
db.trips.find({ "tripduration": { $gt: 150 } })
  .explain("executionStats")
```

```
stage: 'FETCH',
inputStage: {
  stage: 'IXSCAN',
  keyPattern: { tripduration: 1 },
  indexName: 'tripduration_1',
```

Hình 9: Output của '.explain("executionStats") khi gọi trong phạm vi tạo Partial Index

Ta tìm các chuyến đi có thời lượng trên 150 phút, cũng là các chuyến đi được tạo index nên kết quả sử dụng `IXSCAN`

Truy vấn ngoài điều kiện — KHÔNG sử dụng index

```
db.trips.find({ "tripduration": { $lt: 100 } })
  .explain("executionStats")
```

```
winningPlan: {
  stage: 'COLLSCAN',
  filter: { tripduration: { '$lt': 100 } },
  direction: 'forward'
},
```

Hình 10: Output của '.explain("executionStats") khi gọi ngoài phạm vi tạo Partial Index

Ta tìm các chuyến đi có thời lượng dưới 100 phút, không thuộc nhóm đã tạo Partial Index nên kết quả sử dụng `COLLSCAN`

Lợi ích của Partial Index:

- Giảm kích thước index, tiết kiệm bộ nhớ.
- Tăng tốc độ truy vấn khi bạn chỉ cần quan tâm đến một phân khúc dữ liệu cụ thể.
- Hữu ích cho các truy vấn có điều kiện (ví dụ: dữ liệu "active", "non-null", "> threshold").

III: MongoDB Drivers

3.1. PyMongo Overview

PyMongo là thư viện chính thức được MongoDB cung cấp để kết nối cơ sở dữ liệu MongoDB bằng ngôn ngữ lập trình Python. Thư viện này cho phép thực hiện đầy đủ các thao tác truy vấn và quản lý dữ liệu như:

- **C**: Create – Tạo tài liệu (insert).
- **R**: Read – Truy vấn dữ liệu (find, find_one).
- **U**: Update – Cập nhật nội dung tài liệu.
- **D**: Delete – Xoá tài liệu khỏi collection.

Ứng dụng chính:

- Kết nối ứng dụng Python với MongoDB Server (local hoặc cloud).
- Thao tác CRUD với dữ liệu một cách thuận tiện.
- Hỗ trợ Aggregation Framework, Indexes, Transactions, v.v.

Cài đặt PyMongo:

Cài đặt thư viện PyMongo

```
pip install pymongo
```

3.2. Connecting to MongoDB

Để bắt đầu thao tác với MongoDB từ Python, ta cần tạo một kết nối đến cơ sở dữ liệu thông qua PyMongo.

Ví dụ kết nối đến MongoDB local:

Kết nối MongoDB local qua PyMongo

```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client["mydatabase"]
collection = db["mycollection"]
```

Ghi chú:

- "localhost:27017" là địa chỉ mặc định cho MongoDB server chạy local.
- "mydatabase" và "mycollection" là tên cơ sở dữ liệu và collection bạn muốn sử dụng.

Nếu bạn sử dụng MongoDB Atlas (cloud), thì cần thay "localhost" bằng URI kết nối được cung cấp.