

# MongoDB

---

A DOCUMENT-ORIENTED DATABASE

# Framework di aggregazione

---

AGGREGATE

# Framework di aggregazione

---

Il framework di aggregazione permette di applicare **trasformazioni e aggregazioni** sui documenti di una collezione

E' costituito da una serie di **operatori di pipeline**, *mattoni* che possono essere liberamente combinati tra loro (**anche più volte ed in qualunque ordine**) per dar vita ad interrogazioni più o meno complesse

- Match, Project, Group, Unwind, Sort, Limit, Skip

Non solo aggregazioni: l'elevata espressività del framework consente di formulare **interrogazioni che non si potevano fare col Find**

- Applicare trasformazioni sulle date
- Concatenare due o più campi
- Confrontare i valori di due campi
- Restituire un singolo elemento di un array invece dell'array intero

# Framework di aggregazione

---

Un esempio: in una collezione di riviste, voglio sapere quali sono gli autori che hanno venduto più di tutti

- **Project**: estraggo da ogni documento l'autore della rivista
- **Group**: raggruppo per autore, contando il numero di occorrenze di ciascuno
- **Sort**: ordino in maniera decrescente sul numero di occorrenze
- **Limit**: mantengo solo i primi 5 risultati

La query:

- ```
db.articles.aggregate([
  {"$project" : {"author" : 1}},
  {"$group" : {"_id" : "$author", "count" : {"$sum" : 1}}},
  {"$sort" : {"count" : -1}},
  {"$limit" : 5}
])
```

# Operatore \$match

---

L'operatore **\$match** permette di **filtrare i documenti**

- Opera sostanzialmente come una query di Find
- Unica eccezione: non supporta operatori geospaziali

E' buona norma utilizzare l'operatore il prima possibile

- Riduce il numero di documenti delle operazioni successive
- Può sfruttare gli indici (in fasi successive potrebbero non essere utilizzabili)

Un esempio

- `db.restaurants.aggregate([{$match: {cuisine: "Hamburger"}}])`
- `db.restaurants.find({cuisine: "Hamburger"})`

# Operatore \$project

---

L'operatore **\$project** permette di **effettuare una proiezione dei campi**

- E' **molto più potente** della proiezione nel comando Find
- Permette di estrarre campi da oggetti innestati e di applicare trasformazioni

```
db.articles.aggregate([{"$project" : {"author" : 1, "_id" : 0}}])
```

- Restituisce l'autore di un articolo ed esclude il campo \_id

```
db.users.aggregate([{"$project" : {"userId" : "$_id", "_id" : 0}}])
```

- Rinomina il campo \_id in userId
- In pratica, introduce un nuovo campo userId il cui valore corrisponde al valore di \_id
- NB: l'utilizzo dell'operatore \$ in "\$\_id" permette di indicare il riferimento ad un campo; altrimenti, "\_id" verrebbe interpretato come un semplice valore

# Operatore \$project – espressioni matematiche

---

## **Espressioni su 1 o più valori:** \$add, \$multiply

- "\$add" : [expr1, expr2, ..., exprN]

## **Espressioni su 2 valori:** \$subtract, \$divide, \$mod

- "\$subtract" : [expr1, expr2]
- "\$divide": divide il primo valore per il secondo
- "\$mod": divide il primo valore per il secondo e restituisce il resto

## **Un esempio**

- `db.employees.aggregate([ { "$project" : { "totalPay" : { "$subtract" : [ { "$add" : [ "$salary", "$bonus" ] }, "$401k" ] } } } ] )`
- Restituisce un campo calcolato: `totalPay = (salary + bonus) - 401k`

# Operatore \$project – espressioni sulle date

---

Ci sono diverse espressioni che permettono di **estrarre una specifica informazione a partire da una data**

- "\$year", "\$month", "\$week"
- "\$dayOfYear", "\$dayOfMonth", "\$dayOfWeek"
- "\$hour", "\$minute", "\$second"

## Esempi

- `db.employees.aggregate([{"$project": {"hiredIn": {"$month": "$hireDate"}} }])`
- Restituisce il mese in cui gli impiegati sono stati assunti
- `db.employees.aggregate([{"$project": {"tenure": {"$subtract": [{"$year": new Date()}, {"$year": "$hireDate"}] } } ]])`
- Restituisce il numero di anni trascorsi dall'assunzione degli impiegati
- Un'operazione aritmetica tra due date restituisce un risultato in millisecondi



# Operatore \$project – espressioni sulle stringhe

---

**"\$substr"** : [*expr*, *startOffset*, *numToReturn*]

- Restituisce una sottostringa della stringa passata come primo parametro; parte da *startOffset* e restituisce *numToReturn* byt
- Attenzione alla codifica: un byte potrebbe non corrispondere ad un carattere

**"\$concat"** : [*expr1*[, *expr2*, ..., *exprN*]]

- Concatena le stringhe passate come parametri

**"\$toLower"**, **"\$toUpperCase"**

- Restituiscono la stringa passata come parametro in tutte minuscole o maiuscolo.

Esempio

- `db.employees.aggregate([{"$project" : {"email" : {"$concat" : [{"$substr" : ["$firstName", 0, 1]}, ".", "$lastName", "@example.com"] } } }])`
- Restituisce una stringa come e.gallinucci@example.com

# Operatore \$project – espressioni logiche

---

## Espressioni di confronto

- **"\$cmp"** : [expr1, expr2]  
Confronta expr1 con expr2. Ritorna 0 se sono uguali, un numero negativo se  $\text{expr1} < \text{expr2}$ , un numero positivo se  $\text{expr1} > \text{expr2}$ .
- **"\$strcasecmp"** : [string1, string2]  
Confronto case-insensitive tra due stringhe
- **"\$eq"/"\$ne"/"\$gt"/"\$gte"/"\$lt"/"\$lte"** : [expr1, expr2]  
Confronta expr1 con expr2 e ritorna true o false

## Espressioni booleane

- **"\$and", "\$or"** : [expr1[, expr2, ..., exprN]]  
Ritorna vero se tutte (\$and) o almeno una (\$or) delle espressioni è vera
- **"\$not"** : *expr*  
Ritorna il valore booleano opposto di *expr*

# Operatore \$project – espressioni logiche

---

## Espressioni di controllo

- "\$cond" : [*booleanExpr*, *trueExpr*, *falseExpr*]  
Se l'espressione *booleanExpr* è vera, ritorna *trueExpr*, altrimenti *falseExpr*
- "\$ifNull" : [*expr*, *replacementExpr*]  
Se *expr* vale null, ritorna *replacementExpr*, altrimenti ritorna *expr*

Un esempio: gli studenti vengono valutati per il 10% sulla presenza, 30% sulle interrogazioni, 60% sulle verifiche; ma prendono 100 se sono “cocchi”

- `db.students.aggregate([{"$project" : {"grade" : {"$cond" : ["$teachersPet", 100, {"$add" : [{"$multiply" : [.1, "$attendanceAvg"]}, {"$multiply" : [.3, "$quizzAvg"]}, {"$multiply" : [.6, "$testAvg"]}]}]} } ]])`

# Operatore \$group

---

L'operatore **\$group** permette di **raggruppare i documenti** sulla base di determinate chiavi e di **calcolare dei valori aggregati**. Alcuni esempi:

- Contesto: misurazioni meteo minuto-per-minuto.  
Query: umidità media per giorno
- Contesto: collezione di studenti  
Query: raggruppare gli studenti per voto
- Contesto: collezione di utenti  
Query: raggruppare gli utenti per città e stato

I campi su cui si vuole raggruppare costituiscono le chiavi del gruppo

- {"\$group" : {"\_id" : "\$day"}}
- {"\$group" : {"\_id" : "\$grade"}}
- {"\$group" : {"\_id" : {"state" : "\$state", "city" : "\$city"}}}

# Operatore \$group ed operatori aritmetici

---

Oltre a specificare le chiavi su cui raggruppare è possibile indicare una o più **operazioni per calcolare valori aggregati**.

Gli operatori aritmetici sono due:

- **"\$sum"** : value  
Produce la somma dei valori
- **"\$avg"** : value  
Produce la media dei valori

Un esempio completo

- `db.sales.aggregate([{"$group" : {  
 "_id" : "$country",  
 "totalRevenue" : {"$avg" : "$revenue"},  
 "numSales" : {"$sum" : 1}  
} ]])`

# Operatore \$group ed operatori su estremi

---

Ci sono quattro operatori per ottenere gli "**estremi**" del dataset:

- "\$max" : *expr* ; "\$min" : *expr*  
Esaminano tutti i documenti e restituiscono rispettivamente il massimo ed il minimo valore trovato
- "\$first" : *expr* ; "\$last" : *expr*  
Esaminano solo il primo e l'ultimo documento per restituire il valore trovato

Due esempi

- `db.scores.aggregate([{"$group" : {  
 "_id" : "$grade",  
 "lowestScore" : {"$min" : "$score"},  
 "highestScore" : {"$max" : "$score"}  
} ]])`

```
db.scores.aggregate([  
  {"$sort" : {"score" : 1 }},  
  {"$group" : {  
    "_id" : "$grade",  
    "lowestScore" : {"$first" : "$score"},  
    "highestScore" : {"$last" : "$score"}  
  }  
] ]])
```

# Operatore \$group ed operatori di collezione

---

Ci sono due operatori che consentono di costruire un array con i valori riscontrati in ciascun gruppo

- **"\$addToSet"** : *expr*  
Costruisce un array con tutti i valori distinti
- **"\$push"**: *expr*  
Costruisce un array con tutti i valori trovati, anche duplicati

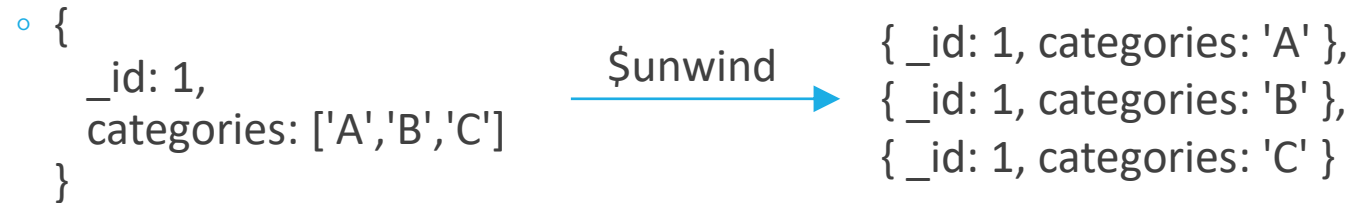
Un esempio

- `db.sales.aggregate([{"$group" : {  
 "_id" : { day : { $dayOfYear: "$date" }, year: { $year: "$date" } },  
 "itemsSold" : { $addToSet: "$item" }  
} ]])`  
Restituisce l'elenco distinto dei prodotti venduti in ciascun giorno

# Operatore \$unwind

---

L'operatore **\$unwind** permette di *appiattare un array*, costruendo tanti documenti quanti sono gli elementi dell'array



Ciò torna utile per effettuare proiezioni e aggregazioni sugli elementi interni degli array

- ```
db.blog.aggregate([  
  {"$project" : {"comments" : "$comments"}},  
  {"$unwind" : "$comments"},  
  {"$match" : {"comments.author" : "Mark"}}  
])
```

Restituisce i commenti di Mark



# Operatore \$unwind

---

Un altro esempio

- { \_id: 1, nome: "Enrico", città: "Cesena", voti: [1, 2, 3],  
 { \_id: 2, nome: "Lorenzo", città: "Cesena", voti: [4, 5, 6] },  
 { \_id: 3, nome: "Matteo", città: "Trieste", voti: [7, 8, 9] } }
- `db.col.aggregate([  
 {"$unwind" : "$voti"},  
 {"$group" : {"_id" : "$città", "mediaVoti": { "$avg" : "$voti" } } },  
])`

Restituisce la media dei voti per città

Se l'array contiene un altro array, è possibile applicare l'operatore `$unwind` in cascata (prima sull'array esterno, poi su quello interno)

# Operatore \$unwind

---

L'operatore \$unwind può essere dichiarato anche come un oggetto, in cui indicare (oltre al campo da appiattare) alcuni parametri opzionali

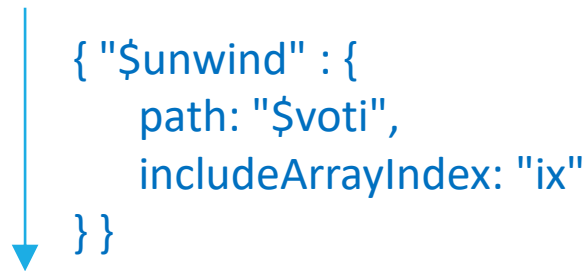
- `$unwind: {`
  - `path: <field path>,`
  - `includeArrayIndex: <string>,`
  - `preserveNullAndEmptyArrays: <boolean>``}`
- **path** è il percorso dell'array (come nella versione semplice)
- **includeArrayIndex** è il nome di un nuovo campo in cui si vuole estrarre l'indice posizionale dell'array
- **preserveNullAndEmptyArrays**, se impostato a true, permette di restituire un documento anche se l'array indicato non esiste (oppure è null o vuoto)

# Operatore \$unwind

---

Un esempio con la versione estesa di \$unwind

- { \_id: 1, nome: "Enrico", città: "Cesena", voti: [1, 2, 3],  
 { \_id: 1, nome: "Lorenzo", città: "Cesena", voti: [4, 5, 4] }



- { \_id: 1, nome: "Enrico", città: "Cesena", voti: 1, ix: 0 },  
 { \_id: 1, nome: "Enrico", città: "Cesena", voti: 2, ix: 1 },  
 { \_id: 1, nome: "Enrico", città: "Cesena", voti: 3, ix: 2 },  
 { \_id: 2, nome: "Lorenzo", città: "Cesena", voti: 4, ix: 0 },  
 { \_id: 2, nome: "Lorenzo", città: "Cesena", voti: 5, ix: 1 },  
 { \_id: 2, nome: "Lorenzo", città: "Cesena", voti: 6, ix: 2 }

# Operatori \$sort, \$limit e \$skip

---

Gli operatori **\$sort**, **\$limit** e **\$skip** funzionano come nella formulazione delle interrogazioni semplici

- Se si vuole ordinare un grande numero di documenti, è buona norma fare l'ordinamento il prima possibile lungo la pipeline e avere un indice sul campo

Un esempio

- `db.employees.aggregate([`  
    `{ "$project" : {`  
        `"compensation" : { "$add" : ["$salary", "$bonus"] },`  
        `"name" : 1`  
    `}},`  
    `{ "$sort" : {"compensation" : -1, "name" : 1} }`  
    `])`  
◦ E' possibile ordinare anche sui campi creati lungo la pipeline

# Lookup

---

# Operatore \$lookup

---

Introdotta a partire dalla versione 3.2

L'operatore **\$lookup** permette di eseguire il **left outer join** tra collezioni residenti nello **stesso database**

- Nella collezione «primaria» viene creato un nuovo campo di tipo *array*, contenente gli eventuali documenti corrispondenti nella collezione «secondaria»

La sintassi:

- **\$lookup**: {  
  **from**: <collection to join>,  
  **localField**: <field from the input documents>,  
  **foreignField**: <field from the documents of the "from" collection>,  
  **as**: <output array field>  
}

# Operatore \$lookup

---

## Collezione **orders**

- { "\_id" : 1, "item" : "abc", "price" : 12, "quantity" : 2 }
- { "\_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1 }
- { "\_id" : 3 }

## Collezione **inventory**

- { "\_id" : 1, "sku" : "abc", "description" : "product 1", "instock" : 120 }
- { "\_id" : 2, "sku" : "def", "description" : "product 2", "instock" : 80 }
- { "\_id" : 3, "sku" : "ghi", "description" : "product 3", "instock" : 60 }
- { "\_id" : 4, "sku" : "jkl", "description" : "product 4", "instock" : 70 }
- { "\_id" : 5, "sku" : null, "description" : "Incomplete" }
- { "\_id" : 6 }

# Operatore \$lookup

---

Esempio di lookup

```
◦ db.orders.aggregate([  
  $lookup: {  
    from: "inventory",  
    localField: "item",  
    foreignField: "sku",  
    as: "inventory_docs"  
  }  
])
```

Risultato →

```
{  
  "_id" : 1,  
  "item" : "abc",  
  "price" : 12,  
  "quantity" : 2,  
  "inventory_docs" : [  
    { "_id" : 1, "sku" : "abc",  
      description: "product 1", "instock" : 120 }  
  ]  
}  
..  
{  
  "_id" : 3,  
  "inventory_docs" : [  
    { "_id" : 5, "sku" : null, "description" : "Incomplete" },  
    { "_id" : 6 }  
  ]  
}
```



# Nuove collection

---

## Statistiche NBA

- <http://www.mediafire.com/file/ju52cn1eadiydz6/NBA2016.json>
- 1 documento contenente statistiche relative alla stagione 2016/17 per tutti i giocatori e tutte le squadre
- Modificato per separare le statistiche su giocatori e squadre in due file

## Statistiche città USA

- <https://gist.githubusercontent.com/Miserlou/c5cd8364bf9b2420bb29/raw/2bf258763cddddd704f8ffd3ea9a3e81d25e2c6f6/cities.json>
- 1000 documenti con statistiche sulle città più popolose negli Stati Uniti d'America

```
mongoimport --collection nba2016players --db test  
C:\nba2016players.json --jsonArray
```