

# MongoDB

---

A DOCUMENT-ORIENTED DATABASE

# Interrogazioni semplici

---

FIND, FINDONE, COUNT, DISTINCT

# Il comando Find

---

È il comando che permette di eseguire interrogazioni (query) sul DB

La forma di base è:

```
db.nomeCollezione.find([[objSel], [objProj]])
```

Dove

- **nomeCollezione** va sostituito col nome della collezione da interrogare; corrispettivo SQL: FROM (ma limitato ad un'unica collezione)
- **[objSel]** è un (eventuale) oggetto che contiene i criteri di ricerca; corrispettivo SQL: WHERE
- **[objProj]** è un (eventuale) oggetto che contiene i criteri di ricerca; corrispettivo SQL: SELECT

# Il comando Find

---

Document-based

```
db
  .nomeCollezione
  .find({objSel}, {objProj})
  .sort({attrs})
```

Relational

```
select objProj
from nomeCollezione
where objSel
order by attrs
```

# Il comando Find

---

## Alcuni esempi

`db.restaurant.find()`

- Restituisce tutti i documenti

`db.restaurant.findOne()`

- Restituisce solo il primo documento

`db.restaurant.find({cuisine: "Hamburgers"})`

- Restituisce i documenti in cui l'attributo cuisine (se presente) è valorizzato con la stringa "Hamburgers"

`db.restaurant.find({}, {cuisine: 1})`

- Restituisce tutti i documenti, ma proiettando solamente l'attributo cuisine (oltre all'\_id, che viene restituito di default)

`db.restaurant.find({cuisine: "Hamburgers"}, {cuisine: 1})`

- La combinazione di selezione e proiezione

# Find - proiezione

---

In caso di proiezione non specificata, vengono restituiti tutti gli attributi di tutti i documenti

Se si indica una proiezione, vengono mantenuti solo i campi indicati – ad eccezione del campo `_id`, che viene mantenuto ugualmente

- E' comunque possibile escludere il campo

## Sintassi

- `nomeChiave: [0, 1]`

## Dove

- `nomeChiave` è il nome di un attributo
- `1` va indicato se si vuole mantenere il campo
- `0` va indicato se, invece, si vuole escludere il campo (e.g., per l'`_id`)

# Find – selezione semplice

---

Una prima modalità di selezione avviene attraverso il match esatto del valore dell'attributo con un valore specificato

Esempi:

- `db.users.find({"age": 27})`
- `db.users.find({"username": "joe"})`
- `db.users.find({"username": "joe", "age": 27})`

Come esprimere condizioni più complesse?

- ~~`db.restaurant.find({restaurant_id > 40367790})`~~
- Non è possibile, perché bisogna rispettare la sintassi degli oggetti Javascript

# Find – selezione complessa

---

L'espressione di condizioni di selezione complesse avviene attraverso l'incapsulamento di nuovi oggetti

## Sintassi

- nomeChiave: { operatore: valore }

## Dove

- operatore corrisponde ad un operatore di confronto secondo la sintassi di MongoDB (e.g., "\$gte", acronimo di "Greater Than or Equal to")
- valore corrisponde ad un valore semplice (e.g., un numero o una stringa)
- Alcuni operatori richiedono che valore sia a sua volta un oggetto, composto da un'altra coppia operatore: valore



# Find – operatori di confronto

---

Quali sono gli operatori

- `$gte`, `$gt` – corrispondono a  $\geq$  e  $>$
- `$lte`, `$lt` – corrispondono a  $\leq$  e  $<$
- `$ne` – corrisponde a  $\neq$

Esempi

- `db.users.find({"age": {"$gte": 18}})`
- `db.users.find({"age": {"$gte": 18, "$lte": 30}})`
- `db.users.find({"registered": {"$lt": new Date("2007-01-01")}})`
  - Il formato della data dipende dalla localizzazione
- `db.users.find({"username": {"$ne": "joe"}})`

# Find – condizioni multiple

---

Quali sono gli operatori

- \$in, \$nin – equivalenti alle clausole IN e NOT IN di SQL
- \$or, \$nor, \$and – equivalenti ai rispettivi operatori logici

Esempi

- `db.users.find({"user_id": {"$in": [12345, "joe"]}})`
- `db.raffle.find({"ticket_no": {"$nin": [725, 542, 390]}})`
- `db.raffle.find({"$or": [{"ticket_no": 725}, {"winner": true}]})`
- `db.raffle.find({"$or": [{"ticket_no": {"$in": [725, 542, 390]}}, {"winner": true}]})`
- `db.raffle.find({"$nor": [{"ticket_no": 725}, {"winner": true}]})`
- `db.raffle.find({"$and": [{"ticket_no": 725}, {"winner": true}]})`

# Find – condizioni multiple

---

Possono esserci modi diversi per esprimere lo stesso criterio, più o meno ottimizzati

## Esempi

- `db.users.find({"$and": [{"x": {"$lt": 5}}, {"x": 1}]})`
- `db.users.find({"x": {"$lt": 5, "$in": [1]}})`

L'ottimizzatore fa più fatica in presenza di operatori `$and` e `$or`; se possibile, è meglio evitare di usarli

# Find – negazione

---

Quali sono gli operatori

- `$not` – permette di negare un determinato criterio

Esempi

- `db.users.find({"id_num": {"$not": {"$mod": [5, 1]}}})`

# Find – esistenza e campi nulli

---

Alcuni attributi possono avere null come valore.

Il comando

- `db.c.find({"y": null})`

restituisce sia i documenti in cui la chiave y esiste ed è valorizzata a null, sia i documenti in cui la chiave y non esiste.

Per avere solo i documenti in cui la chiave y esiste ed è valorizzata a null, bisogna verificare anche l'esistenza della chiave stessa:

- `db.c.find({"y": {"$in": [null], "$exists": true}})`

# Lab time!

---

# Find – interrogare array

---

Contesto: collezione food con 3 documenti:

- {"\_id": 1, "fruit": ["apple", "banana", "peach"]}
- {"\_id": 2, "fruit": ["apple", "kumquat", "orange"]}
- {"\_id": 3, "fruit": ["cherry", "banana", "apple"]}

## Comandi

- db.food.find({"fruit": "banana"})  
match se l'array contiene banana (restituisce: 1 e 3)
- db.food.find({fruit: {\$all: ["apple", "banana"]}})  
match se l'array contiene sia apple che banana (restituisce: 1 e 3)
- db.food.find({fruit: {\$in: ["apple", "banana"]}})  
match se l'array contiene apple o banana (restituisce: 1, 2 e 3)

# Find – interrogare array

---

Contesto: collezione food con 3 documenti:

- {"\_id": 1, "fruit": ["apple", "banana", "peach"]}
- {"\_id": 2, "fruit": ["apple", "kumquat", "orange"]}
- {"\_id": 3, "fruit": ["cherry", "banana", "apple"]}

## Comandi

- db.food.find({"fruit": ["banana", "apple", "peach"]})  
match se l'array corrisponde esattamente a quello indicato (restituisce: nulla)
- db.food.find({"fruit.2": "peach"})  
match se l'array contiene peach in posizione 2 0-based (restituisce: 1)
- db.food.find({"fruit": {"\$size": 3}})  
match se l'array contiene 3 elementi (restituisce: 1, 2 e 3)



# Find – interrogare array

---

In fase di proiezione è possibile limitare il numero di elementi dell'array che vengono restituiti dalla query

Contesto: un doc che contiene il post di un blog ed i relativi commenti

- `db.blog.posts.find(criteria, {"comments": {"$slice": 10}})`  
restituisce i primi 10 commenti
- `db.blog.posts.find(criteria, {"comments": {"$slice": -10}})`  
restituisce gli ultimi 10 commenti
- `db.blog.posts.find(criteria, {"comments": {"$slice": [23,10]}})`  
salta i primi 23 documenti e restituisce i 10 successivi (dal 24° al 33°)
- `db.blog.posts.find(criteria, {"comments.$": 1})`  
restituisce i commenti che rispondono ai criteri di selezione indicati

Attenzione: se `$slice` è l'unico operatore utilizzato nella proiezione, tutti i campi dei documenti vengono restituiti

# Find – interrogare array

---

Quando si pone una selezione con più criteri su un **attributo con valore semplice** (e.g., una stringa o un numero), i criteri sono valutati in **AND**

- `db.test.find({"x": {"$gt":10, "$lt":20}})`

**Il valore di x deve essere maggiore di 10 e minore di 20**

Se l'**attributo è un array**, i criteri sono valutati in **OR** per ogni elemento: se ce n'è almeno uno che corrisponde, il documento viene restituito

- `db.test.find({"x": {"$gt":10, "$lt":20}})`

**Gli elementi dell'array x devono essere o maggiori di 10, o minori di 20**

Per imporre i due vincoli in **AND** sugli elementi di un'array, bisogna utilizzare l'operatore **\$elemMatch**

- `db.test.find({"x": {"$elemMatch": {"$gt":10, "$lt":20}})`

# Find – interrogare oggetti

---

## Contesto

- `{"name": {"first": "Joe", "middle": "K", "last": "Schmoe"}}`

## Esistono due modalità

- `db.people.find({"name": {"first": "Joe", "last": "Schmoe"}})`  
Match esatto: l'oggetto cercato deve essere uguale a quello specificato (in questo caso, non restituisce nulla)
- `db.people.find({"name.first": "Joe", "name.last": "Schmoe"})`  
In alternativa, si può usare la dot notation per referenziare i singoli campi (in questo caso, restituisce il documento)

# Find – interrogare oggetti dentro ad array

---

Obiettivo: cercare i commenti di Joe con un punteggio di almeno 5

- `db.blog.find({"comments": {"author": "joe", "score": {"$gte": 5}}})`

Sbagliato: cerca il match esatto

- `db.blog.find({"comments.author": "joe", "comments.score": {"$gte": 5}})`

Sbagliato: restituisce entrambi i commenti, perché le condizioni sono valutate in OR

- `db.blog.find({"comments": {"$elemMatch": {"author": "joe", "score": {"$gte": 5}}}})`

Corretto

Contesto:

```
{
  "content": "...",
  "comments": [{
    "author": "joe",
    "score": 3,
    "comment": "nice post"
  }, {
    "author": "mary",
    "score": 6,
    "comment": "terrible post"
  }]
}
```

# Find – Javascript scripts

---

L'espressività delle query tramite coppie chiave-valore è limitata

Per interrogazioni particolarmente complesse è possibile utilizzare l'operatore `$where`, che consente di eseguire uno script Javascript

- `db.mycoll.find({$where: function() { return this.date.getMonth() == 11} })`
- La complessità dello script è liberamente definita dall'utente

Tramite script è possibile fare praticamente qualunque tipo di operazione

- Per questioni di sicurezza, però, è fortemente sconsigliato l'utilizzo dell'operatore `$where`
- In generale, agli utenti finali non dovrebbe MAI essere concesso di eseguire questo tipo di interrogazioni

# Limit, skip & sort

---

Al comando find possono essere applicati in cascata ulteriori comandi, al fine di applicare alcune trasformazioni al risultato ottenuto

Limit: restituisce solo i primi n documenti

- `db.c.find().limit(3)`

Skip: salta i primi n documenti e restituisci i successivi

- `db.c.find().skip(3)`

Sort: ordina i risultati sulla base di uno o più attributi

- `db.c.find().sort({username: 1, age: -1})`
- L'ordinamento può essere crescente (1) o decrescente (-1)

# Limit, skip & sort

---

Questi comandi possono essere combinati

Un'applicazione spesso utilizzata è quella della paginazione dei risultati

- Contesto: negozio di e-commerce
- L'utente cerca i prodotti di tipo mp3 in ordine decrescente di prezzo
- `db.stock.find({"desc": "mp3"}).limit(50).sort({"price": -1})`
- L'utente vuole vedere più risultati e clicca per accedere alla pagina successiva
- `db.stock.find({"desc": "mp3"}).limit(50).skip(50).sort({"price": -1})`

Il comando `sort` può essere specificato prima o dopo `limit` e `skip`, ma la sua esecuzione è sempre antecedente agli altri

Nota: a fini prestazionali, è bene evitare valori troppo elevati di `skip`

- In tal caso, gestire la paginazione lato applicazione può risultare più efficiente

# Count

---

Count è il comando per contare il numero di documenti restituiti da una query

- `db.foo.count()`
- `db.foo.count({"x": 1})`
- Sostanzialmente simile al Find, con l'eccezione dell'assenza dell'oggetto di selezione



# Distinct

---

Distinct è il comando per restituire i valori distinti di un campo a partire dai documenti che corrispondono ai criteri indicati

- `db.inventory.distinct( "item.sku", { dept: "A" } )`
- Restituisce i valori distinti del campo `item.sku` nei documenti in cui il dipartimento è A
- Se `item.sku` è un array, vengono restituiti i valori distinti anche rispetto all'array di un singolo documento