Basi Dati NoSQL

Introduzione a MongoDB

MongoDB: Introduzione

- MongoDB è il sistema di database più utilizzato tra quelli basate su documenti.
- Funzioni aggiuntive oltre alle standard di NoSQL:
  - Alte prestazioni
  - Disponibilità
  - Scalabilità nativa
  - Alta flessibilità
  - Open source

Terminologia – Concetti a confronto

Basi dati relazionali	Mongo DB
Tabella	Collezione
Record	Documento
Colonna	Campo

MongoDB: design dei documenti

- Rappresentazione dei dati ad alto livello:
  - I record sono memorizzati sotto forma di documenti
    - Formati da coppie chiave-valore
    - Simili a oggetti JSON.
    - Possono essere nidificati.

MongoDB: design dei documenti

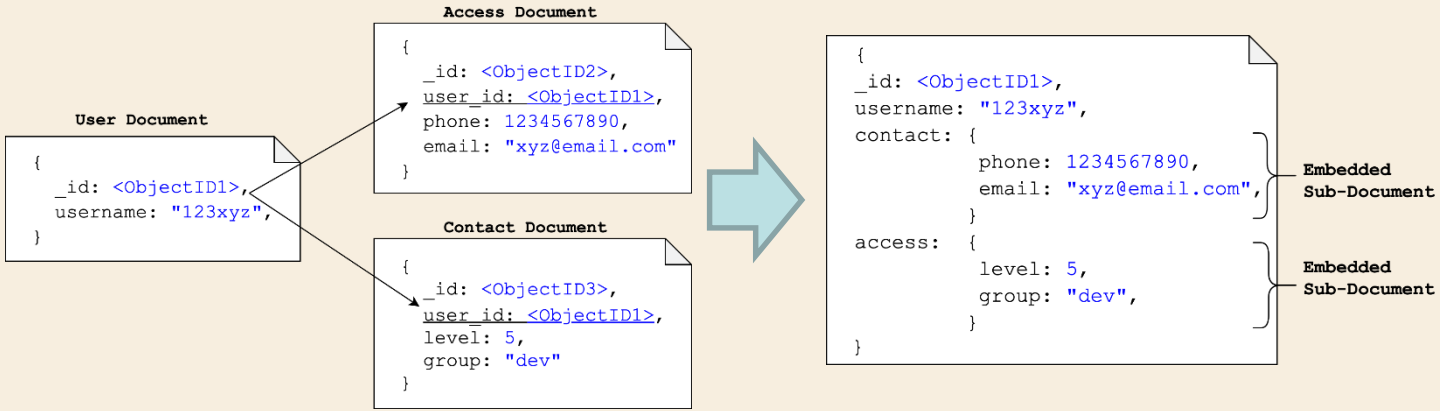
- Flessibile e con una ricca sintassi. Si adatta alla maggior parte dei casi d'uso.

- Permette il mapping dei tipi in oggetti dei principali linguaggi di programmazione:
  - anno, mese, giorno, timestamp,
  - liste, sotto-documenti, etc.

MongoDB: design dei documenti

Attenzione!

- Le relazioni tra documenti sono inefficienti.
  - Il riferimento viene fatto tramite l'uso dell'Object(ID). **Non** esiste l'operatore di **join nativo.**



MongoDB: Caratteristiche principali

- Linguaggio di query ricco di funzionalità:
  - I documenti possono essere creati, letti, aggiornati e cancellati.
  - Il linguaggio SQL non è supportato.
  - Sono disponibili delle interfacce di comunicazione per i principali linguaggi di programmazione:
    - JavaScript, PHP, Python, Java, C#, ..

MongoDB: Caratteristiche principali

- **Di default**, MongoDB non supporta le transazioni multi-documento.
  - Le proprietà **ACID** sono soddisfatte solo a livello di **singolo documento**.
- Da **MongoDB 4.0**, le transazioni multi-documento sono supportate
  - Questa caratteristica impatta in modo rilevante sulle performance.

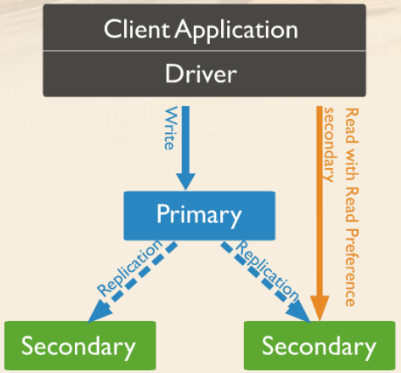
MongoDB: Caratteristiche principali

- Scalabilità orizzontale attraverso l'uso di **tecniche di sharding**
  - Ogni shard contiene un sottoinsieme di documenti.
  - Prestare attenzione **all'attributo** di **sharding**
    - **Può avere un impatto significativo sulle performance delle query.**
- **Indici**
  - Velocizzano le query
  - Diversi tipi di indici (Single Field, Multi-key, Geo spaziale, testuali…)
  - Di default, un indice viene creato sull'ID del documento.

MongoDB: Repliche

- Un **replica set** è un gruppo di istanze di MongoDB che contengono gli stessi dati
  - Replica sets = Copie multiple dei dati
- La replicazione fornisce **ridondanza** e **aumenta la disponibilità** dei dati**.**
  - Tolleranza ai guasti contro la perdita di un singolo server
- La replicazione può fornire un **aumento** nella **capacità di lettura** (i dati possono essere letti da diversi server).
  - **Non è il comportamento di default** in MongoDB

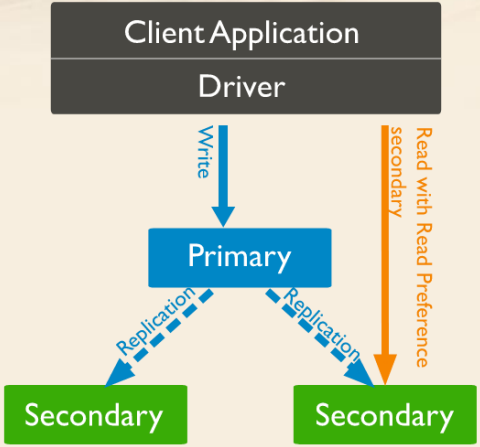
MongoDB: Repliche



- Replica set
  - Nodo principale
    - Riceve tutte le operazioni di scrittura e aggiornamento
  - Nodi secondari
    - Replicano le stesse operazioni del nodo principale nei propri set di dati.
- Replicazione asincrona
- Failover automatico
  - Quando il nodo principale smette di funzionare, uno di quelli secondari inizia la procedura di sostituzione.

MongoDB: Repliche

- Operazioni di lettura
  - Tutti i nodi nel replica set possono accettare operazioni in lettura
  - Le repliche in MongoDB si basano sulla replica **asincrona**. → Letture da **nodi secondari potrebbero** restituire dati che **non riflettono lo stato del nodo principale.**



- Di **default**, un applicazione dirige le **richieste di lettura verso il nodo principale.**
  - Per evitare incoerenza di dati

Casi d'uso: MongoDB vs Oracle

- I casi d'uso più comuni di MongoDB includono:
  - Internet of Things, Mobile, Analisi Real-Time, Personalizzazione, Dati geo spaziali.
- Oracle è ritenuto più adatto per:
  - Applicazioni che richiedono molte transazioni complesse (ad esempio: un sistema di gestione di partite doppie).

From https://www.mongodb.com/compare/mongodb-oracle

Casi d'uso: MongoDB + Oracle

- I sistemi di prenotazione che gestiscono un sistema di prenotazione viaggi.
  - La parte principale del sistema di prenotazione dovrebbe utilizzare Oracle.
  - Quelle parti dell'applicazione che interagiscono con l'utente finale – pubblicano contenuti, si integrano ai social network, gestiscono le sessioni – sarebbe meglio gestirli con MongoDB.

From https://www.mongodb.com/compare/mongodb-oracle

MongoDB

Operatori per selezionare i dati

MongoDB: query language

- La maggior parte delle operazioni disponibili in SQL può essere espressa nel linguaggio usato da MongoDB.

MySQL	MongoDB
`SELECT`	`find()`

`SELECT` `*` `FROM people`	`db.people.`**`find()`**

MongoDB: operatore find()

MySQL	MongoDB
SELECT	find()

```
SELECT id,
       user_id,
       status
FROM people
```

```
db.people.find(
    { },
    { user_id: 1,
      status: 1
    }
)
```

MongoDB: operatore find()

MySQL	MongoDB
SELECT	find()

```
SELECT id,
       user_id,
       status
FROM people
```

```
db.people.find(
    { },
    { user_id: 1,
      status: 1
    }
)
```

Condizioni (WHERE)

Selezione (SELECT)

MongoDB: operatore find()

MySQL	MongoDB
SELECT	find()
WHERE	find({<WHERE CONDITIONS>})

SELECT * FROM people WHERE status = "A"	db.people.find(**{ status: "A" }**)

Condizioni (WHERE)

MongoDB: operatore find()

MySQL	MongoDB
SELECT	find()
WHERE	find({<WHERE CONDITIONS>})

```
SELECT user_id, status
FROM people
WHERE status = "A"
```

```
db.people.find(
    { status: "A" },
    { user_id: 1,
      status: 1,
      _id: 0
    }
)
```

Condizioni (WHERE)

Selezione (SELECT)

Di default, il campo `_id` viene sempre mostrato.
Per escludero dalla visualizzatione bisogna usare: `_id: 0`

MongoDB: operatori di confronto

- Nel linguaggio SQL, gli operatori di confronto sono essenziali per esprimere condizioni sui dati.
- Nel linguaggio usato da MongoDB sono disponibili con una sintassi differente.

MySQL	MongoDB	Descrizione
>	$gt	Maggiore di
>=	$gte	Maggiore o uguale a
<	$lt	Minore di
<=	$lte	Minore o uguale a
=	$eq	Uguale a
<>	$neq	Diverso da

MongoDB: operatori di confronto (>)

MySQL	MongoDB	Descrizione
>	$gt	Maggiore di

```
SELECT *
FROM people
WHERE age > 25
```

```
db.people.find(
    { age: { $gt: 25 } }
)
```

MongoDB: operatori di confronto (>=)

MySQL	MongoDB	Descrizione
>	$gt	Maggiore di
>=	**$gte**	**Maggiore o uguale a**

```
SELECT *
FROM people
WHERE age >= 25
```

```
db.people.find(
    { age: { $gte: 25 } }
)
```

MongoDB: operatori di confronto (<)

MySQL	MongoDB	Descrizione
>	$gt	Maggiore di
>=	$gte	Maggiore o uguale a
<	**$lt**	**Minore di**

```
SELECT *
FROM people
WHERE age < 25
```

```
db.people.find(
    { age: { $lt: 25 } }
)
```

MongoDB: operatori di confronto (<=)

MySQL	MongoDB	Descrizione
>	$gt	Maggiore di
>=	$gte	Maggiore o uguale a
<	$lt	Minore di
<=	**$lte**	**Minore o uguale a**

```
SELECT *
FROM people
WHERE age <= 25
```

```
db.people.find(
    { age: { $lte: 25 } }
)
```

MongoDB: operatori di confronto (=)

MySQL	MongoDB	Descrizione
>	$gt	Maggiore di
>=	$gte	Maggiore o uguale a
<	$lt	Minore di
<=	$lte	Minore o uguale a
=	**$eq**	**Uguale a**

```
SELECT *
FROM people
WHERE age = 25
```

```
db.people.find(
    { age: { $eq: 25 } }
)
```

MongoDB: operatori di confronto (!=)

MySQL	MongoDB	Descrizione
>	$gt	Maggiore di
>=	$gte	Maggiore o uguale a
<	$lt	Minore di
<=	$lte	Minore o uguale a
=	$eq	Uguale a
<>	**$neq**	**Diverso da**

SELECT * FROM people WHERE **age <> 25**	db.people.find({ **age: { $neq: 25 }** })

MongoDB: operatori condizionali

- Per specificare condizioni multiple, **gli operatori condizionali** sono usati per affermare se una o entrambe le condizioni devono essere soddisfatte.
- Anche in questo caso MongoDB offre le stesse funzionalità di SQL con una sintassi diversa.

MySQL	MongoDB	Descrizione
AND	,	Entrambe soddisfatte
OR	$or	Almeno una soddisfatta

MongoDB: operatori condizionali (AND)

MySQL	MongoDB	Descrizione
AND	**,**	**Entrambe soddisfatte**

```
SELECT *
FROM people
WHERE status = "A"
AND age = 50
```

```
db.people.find(
    { status: "A",
      age: 50 }
)
```

MongoDB: operatori condizionali (OR)

MySQL	MongoDB	Descrizione
AND	,	Entrambe soddisfatte
OR	**$or**	**Almeno una soddisfatta**

```
SELECT *
FROM people
WHERE status = "A"
OR age = 50
```

```
db.people.find(
{ $or:
   [ { status: "A" } ,
     { age: 50 }
   ]
}
)
```

MongoDB: operatore count()

MySQL	MongoDB
COUNT	count()or find().count()

SELECT COUNT(*) FROM people	db.people.count() oppure db.people.find().count()

MongoDB: operatore count()

MySQL	MongoDB
COUNT	count()or find().count()

- Analogamente all'operatore find(), count() può avere come argomento gli operatori condizionali.

```
SELECT COUNT(*)
FROM people
WHERE age > 30
```

```
db.people.count(
{ age: { $gt: 30 } }
)
```

MongoDB: ordinare i dati

- Per ordinare i dati rispetto a un attributo specifico bisogna utilizzare l'operatore sort().

MySQL	MongoDB
ORDER BY	sort()

```
SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id ASC
```

```
db.people.find(
  { status: "A" }
).sort( { user_id: 1 } )
```

MongoDB: ordinare i dati

- Per ordinare i dati rispetto a un attributo specifico bisogna utilizzare l'operatore sort().

MySQL	MongoDB
ORDER BY	sort()

SELECT * FROM people WHERE status = "A" **ORDER BY user_id ASC**	db.people.find({ status: "A" }).**sort({ user_id: 1 })**
SELECT * FROM people WHERE status = "A" **ORDER BY user_id DESC**	db.people.find({ status: "A" }).**sort({ user_id: -1 })**

MongoDB

Inserire, aggiornare e cancellare documenti

MongoDB: inserire nuovi documenti

- Mongo DB permette di inserire nuovi documenti nella base dati. Ogni tupla SQL corrisponde a un documento in MongoDB.

- La chiave primaria `_id` viene automaticamente aggiunta se il campo `_id` non è specificato.

MySQL	MongoDB
`INSERT INTO`	`insertOne()`

MongoDB: inserire nuovi documenti

MySQL	MongoDB
INSERT INTO	insertOne()

```
INSERT INTO
people(user_id,
       age,
       status)
VALUES ("bcd001",
        45,
        "A")
```

```
db.people.insertOne(
   {
     user_id: "bcd001",
     age: 45,
     status: "A"
   }
)
```

MongoDB: inserire nuovi documenti

- In MongoDB è possibile inserire più documenti con un singolo comando usando l'operatore `insertMany()`.

```
db.products.insertMany( [
      { user_id: "abc123", age: 30, status: "A"},
      { user_id: "abc456", age: 40, status: "A"},
      { user_id: "abc789", age: 50, status: "B"}
   ] );
```

MongoDB: aggiornare documenti esistenti

- I dati esistenti possono essere modificati a seconda delle necessità.
- Aggiornare le tuple richiede la loro selezione tramite delle condizioni di «WHERE»

MySQL	MongoDB
UPDATE <table> SET <statement> WHERE <condition>	db.<table>.updateMany({ <condition> }, { $set: {<statement>} })

MongoDB: aggiornare documenti esistenti

MySQL	MongoDB
UPDATE <table> SET <statement> WHERE <condition>	db.<table>.updateMany({ <condition> }, { $set: {<statement>} })

UPDATE people SET status = "C" WHERE age > 25	db.people.updateMany({ age: { $gt: 25 } }, { $set: { status: "C" }})

MongoDB: aggiornare documenti esistenti

MySQL	MongoDB
UPDATE <table> SET <statement> WHERE <condition>	db.<table>.updateMany({ <condition> }, { $set: {<statement>} })

UPDATE people SET status = "C" WHERE age > 25	db.people.updateMany({ age: { $gt: 25 } }, { $set: { status: "C" }})
UPDATE people SET age = **age + 3** WHERE status = "A"	db.people.updateMany({ status: "A" } , { **$inc: { age: 3 }** })

L'operatore `$inc` incrementa il valore di un campo.

MongoDB: cancellare documenti

- Cancellare dati esistenti, in MongoDB corrisponde alla cancellazione del documento associato.
- In maniera simile a SQL, più documenti possono essere cancellati con un singolo comando.

MySQL	MongoDB
`DELETE FROM`	`deleteMany()`

MongoDB: cancellare documenti

MySQL clause	MongoDB operator
`DELETE FROM`	`deleteMany()`

`DELETE FROM people`
`WHERE status = "D"`
	`db.people.deleteMany(`
	`  { status: "D" }`
	`)`

MongoDB: cancellare documenti

MySQL clause	MongoDB operator
`DELETE FROM`	`deleteMany()`

`DELETE FROM people`
`WHERE status = "D"`
	`db.people.deleteMany(`
	`  { status: "D" }`
	`)`

`DELETE FROM people`	`db.people.deleteMany({})`

MongoDB

Indici

MongoDB: Indici

- Gli indici sono strutture dati che memorizzano una porzione della base dati in una struttura ottimizzata.
- Gli indici memorizzano, per un attributo specifico, i valori ordinati.
- Questo permette loro di applicare in modo efficiente condizioni di uguaglianza (=, <>), condizioni di ordine (>, <, …) e operazioni di ordinamento (sort).

MongoDB: Indici

- MongoDB fornisce diversi tipi di indici:
  - Indici Single field (su un singolo attributo)
  - Indici Compound field (su più attributi)
  - Indici Multikey (se l'attributo è un array)
  - Indici Geo spaziali (su coordinate spaziali)
  - Indici di campi di tipo testuale
  - Indici di tipo Hash

MongoDB: Creare nuovi indici

- Creare un indice

```
db.collection.createIndex(<index keys>, <options>)
```

  - Per versioni precedent alla v. 3.0 bisogna usare `db.collection.ensureIndex()`

- Le opzioni includono: `name`, `unique` (se bisogna accettare o meno l'inserimento di documenti con chiavi duplicate), `background`, `dropDups`, ..

MongoDB: indici

- Indici single field
  - Supportano il verso di ordinamento (ascendente/discendente) sul campo indicizzato.
- E.g.,
  - `db.orders.createIndex( {orderDate: 1} )`
- Indici Compound field
  - Supportano l'indicizzazione su più attributi
- E.g.,
  - `db.orders.createIndex( {orderDate: 1, zipcode: -1} )`

MongoDB: indici

- MongoDB supporta interrogazioni efficenti su dati geo spaziali.
- I dati geo spaziali sono memorizzati come:
  - Oggetti GeoJSON : documenti incorporati { <type>, <coordinate> }
    - E.g., location: `{type: "Point", coordinates: [-73.856, 40.848]}`
  - Coppie di coordinate: array o documenti incorporati
    - `point: [-73.856, 40.848]`

MongoDB: dati geo spaziali

- Indici geospaziali
  - MongoDB fornisce due tipi di indici geospaziali: `2d` e `2dsphere`
- Un indice `2dsphere` supporta interrogazioni che calcolano distanze su una superficie sferica.
- Bisogna usare un indice `2d` per dati memorizzati come punti su un piano bidimensionale.
- Esempio,
  - `db.places.createIndex( {location: "2dsphere"} )`
- Operatori geo spaziali:
  - $geoIntersects, $geoWithin, $near, $nearSphere

MongoDB: operatori geo spaziali

- Sintassi di `$near`:

```
{
   <location field>: {
     $near: {
       $geometry: {
          type: "Point" ,
          coordinates: [ <longitude> , <latitude> ]
       },
       $maxDistance: <distance in meters>,
       $minDistance: <distance in meters>
     }
   }
}
```

MongoDB: operatori geo spaziali

- E.g.,
  - `db.places.createIndex( {location: "2dsphere"} )`
- Operatori geo spaziali:
  - $geoIntersects, $geoWithin, $near, $nearSphere
- Operatori geo spaziali nelle funzioni di aggregazione:
  - $near

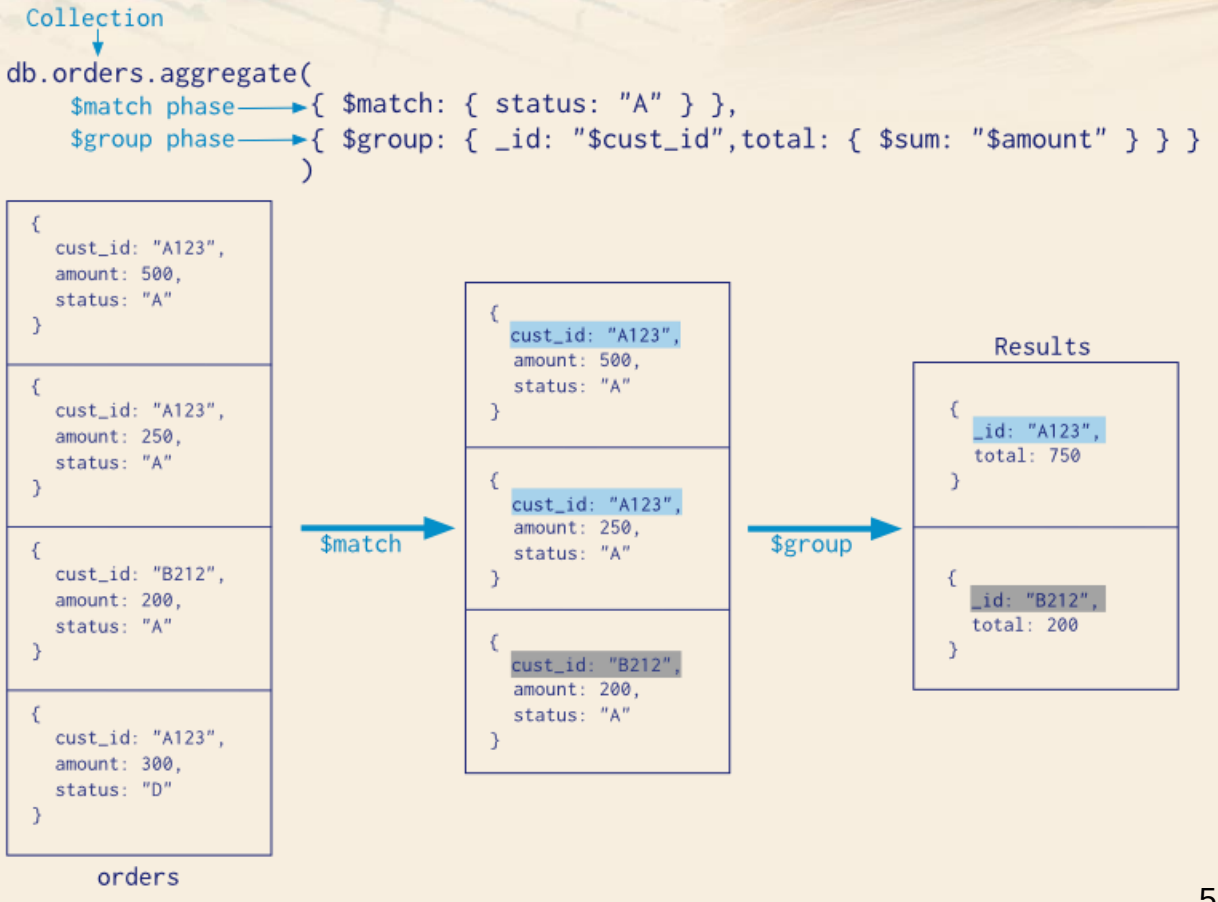
MongoDB

Operatori di aggregazione

Aggregazione su MongoDB

- Gli operatori di aggregazione processano i dati in input e ritornano il risultato delle operazioni applicate.
- I documenti entrano in una pipeline che consiste di più fasi che trasforma i documenti in risultati aggregati.

Aggregazione su MongoDB



Aggregazione su MongoDB: Group By

MySQL	MongoDB
GROUP BY	aggregate($group)

```
SELECT status,
       SUM(age) AS total
FROM people
GROUP BY status
```

```
db.orders.aggregate( [
   {
     $group: {
        _id: "$status",
        total: { $sum: "$age" }
     }
   }
] )
```

Aggregazione su MongoDB: Group By

MySQL	MongoDB
GROUP BY	aggregate($group)

```
SELECT status,
       SUM(age) AS total
FROM people
GROUP BY status
```

```
db.orders.aggregate( [
   {
     $group: {
        _id: "$status",
        total: { $sum: "$age" }
     }
   }
] )
```

Campo usato per l'aggregazione

Aggregazione su MongoDB: Group By

MySQL	MongoDB
GROUP BY	aggregate($group)

```
SELECT status,
       SUM(age) AS total
FROM people
GROUP BY status
```

```
db.orders.aggregate( [
   {
     $group: {
        _id: "$status",
        total: { $sum: "$age" }
     }
   }
] )
```

Campo usato per l'aggregazione

Funzione di aggregazione

Aggregazione su MongoDB: Group By

MySQL	MongoDB
HAVING	aggregate($group, $match)

```
SELECT status,
       SUM(age) AS total
FROM people
GROUP BY status
HAVING total > 1000
```

```
db.orders.aggregate( [
   {
     $group: {
        _id: "$status",
        total: { $sum: "$age" }
     }
   },
  { $match: { total: { $gt: 1000 } } }
] )
```

Aggregazione su MongoDB: Group By

MySQL	MongoDB
HAVING	aggregate($group, $match)

```
SELECT status,
       SUM(age) AS total
FROM people
GROUP BY status
HAVING total > 1000
```

```
db.orders.aggregate( [
    {
      $group: {
         _id: "$status",
         total: { $sum: "$age" }
      }
    },
   { $match: { total: { $gt: 1000 } } }
] )
```

Fase di aggregazione: Specificare l'attributo e la funzione applicate durante il raggruppamento.

Aggregazione su MongoDB: Group By

SQL	MongoDB
HAVING	aggregate($group, $match)

```
SELECT status,
       SUM(age) AS total
FROM people
GROUP BY status
HAVING total > 1000
```

```
db.orders.aggregate( [
   {
     $group: {
        _id: "$status",
        total: { $sum: "$age" }
     }
   },
   { $match: { total: { $gt: 1000
] )
```

Fase di aggregazione: Specificare l'attributo e la funzione applicate durante il raggruppamento.

Condizioni: specificare le condizioni come nel campo `HAVING`