



Politecnico
di Torino



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

MongoDB: Why

Why MongoDB	
<i>What</i>	Why
<i>JSON</i>	End To End
<i>No Schema</i>	“No DBA”, Just Serialize
<i>Write</i>	10K Insert/sec on virtual machine
<i>Read</i>	Similar to MySQL
<i>HA</i>	10 min to setup a cluster
<i>Sharding</i>	Nativo
<i>LBS</i>	Great for that
<i>Buzz</i>	Trend with NoSQL

<http://blogs.microsoft.co.il/blogs/vprnd>

<http://top-performance.blogspot.com>

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.

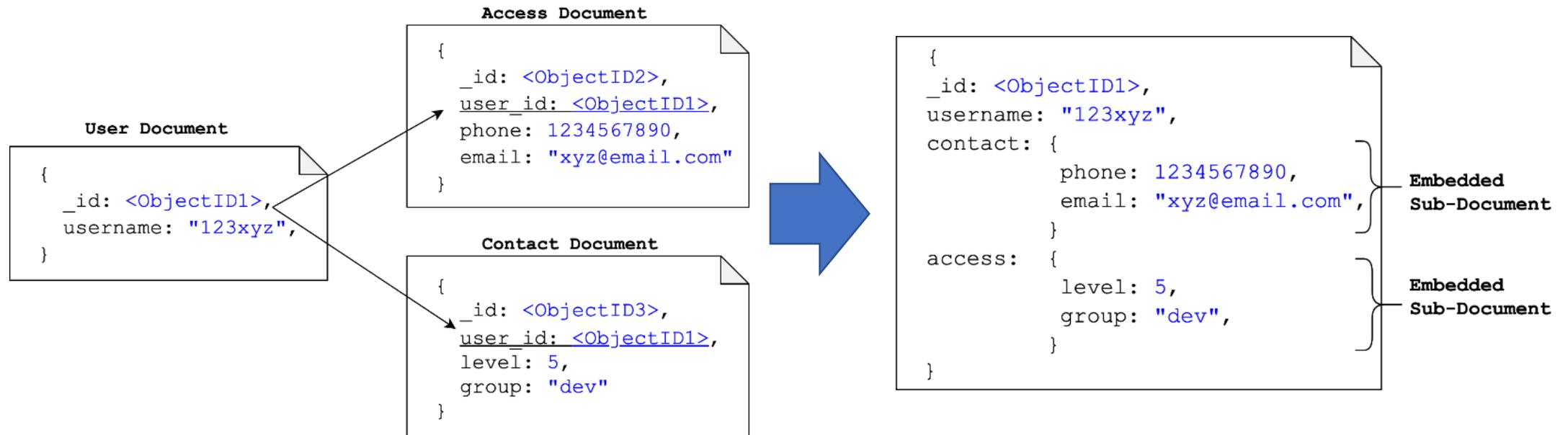


MongoDB: design dei documenti

- 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

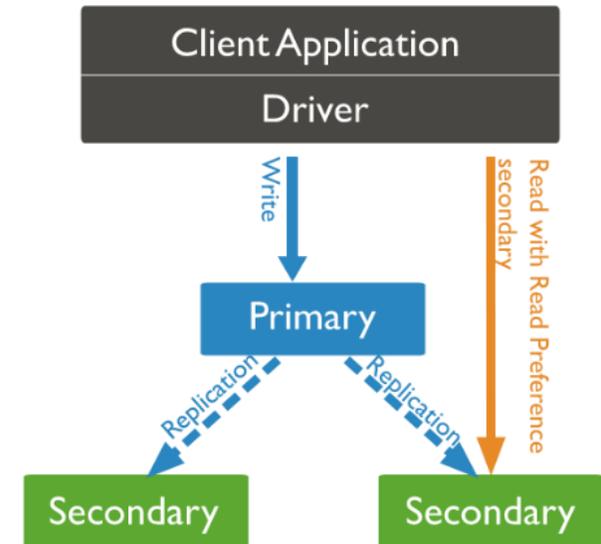
- Scalabilità orizzontale attraverso l'uso di **tecniche di sharding**
 - Ogni shard contiene un sottoinsieme di documenti.
 - Prestare attenzione **all'attributo di sharding: può avere un impatto significativo sulle performance delle query.**
- **Indici**
 - Velocizzano le query
 - Diversi tipi di indici (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

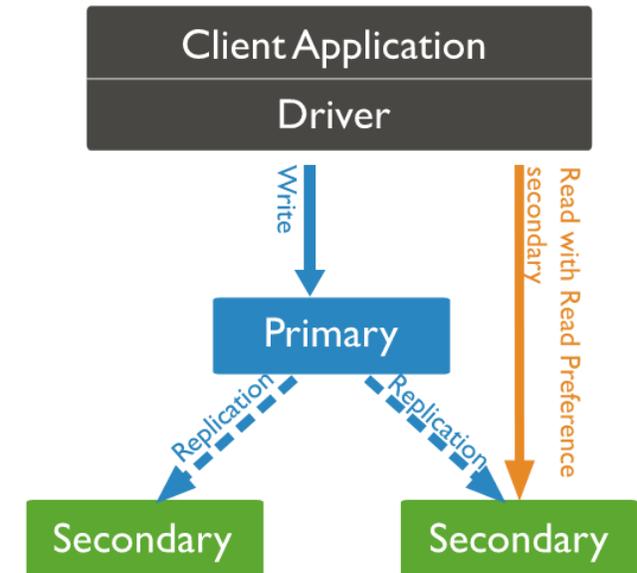
MongoDB: Repliche

- 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>



Operatori per selezionare i dati

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: query language

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

MySQL	MongoDB
SELECT	<code>find()</code>

SELECT * FROM people	<code>db.people.find()</code>
--------------------------------	--------------------------------------

MongoDB: operatore find()

MySQL	MongoDB
SELECT	find()

<pre>SELECT id, user_id, status FROM people</pre>	<pre>db.people.find({ }, { user_id: 1, status: 1 })</pre>
--	---

MongoDB: operatore find()

MySQL	MongoDB
SELECT	find()

<pre>SELECT id, user_id, status FROM people</pre>	<pre>db.people.find({ }, { user_id: 1, status: 1 })</pre>
---	---

Condizioni
(WHERE)

Selezione (SELECT)

MongoDB: operatore find()

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

<pre>SELECT * FROM people WHERE status = "A"</pre>	<pre>db.people.find({ status: "A" })</pre>
--	---

Condizioni (WHERE)

MongoDB: operatore find()

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

<pre>SELECT user_id, status FROM people WHERE status = "A"</pre>	<pre>db.people.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })</pre> <p>Condizioni (WHERE)</p> <p>Selezione (SELECT)</p>
--	---

Di default, il campo `_id` viene sempre mostrato.

Per escluderlo dalla visualizzazione 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	<code>\$or</code>	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()

<pre>SELECT COUNT(*) FROM people</pre>	<pre>db.people.count() oppure db.people.find().count()</pre>
--	--

MongoDB: operatore count()

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

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

<pre>SELECT COUNT(*) FROM people WHERE age > 30</pre>	<pre>db.people.count({ age: { \$gt: 30 } })</pre>
--	---

MongoDB: ordinare i dati

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

MySQL	MongoDB
ORDER BY	<code>sort()</code>

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

MongoDB: ordinare i dati

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

MySQL	MongoDB
ORDER BY	<code>sort()</code>

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



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	<code>insertOne()</code>

MongoDB: inserire nuovi documenti

MySQL	MongoDB
INSERT INTO	insertOne()

<pre>INSERT INTO people(user_id, age, status) VALUES ("bcd001", 45, "A")</pre>	<pre>db.people.insertOne({ user_id: "bcd001", age: 45, status: "A" })</pre>
--	---

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
<pre>UPDATE <table> SET <statement> WHERE <condition></pre>	<pre>db.<table>.updateMany({ <condition> }, { \$set: {<statement>} })</pre>

MongoDB: aggiornare documenti esistenti

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

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

MongoDB: aggiornare documenti esistenti

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

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

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()

<pre>DELETE FROM people WHERE status = "D"</pre>	<pre>db.people.deleteMany({ status: "D" })</pre>
--	--

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({})



Politecnico
di Torino



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 precedenti 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 efficienti 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`



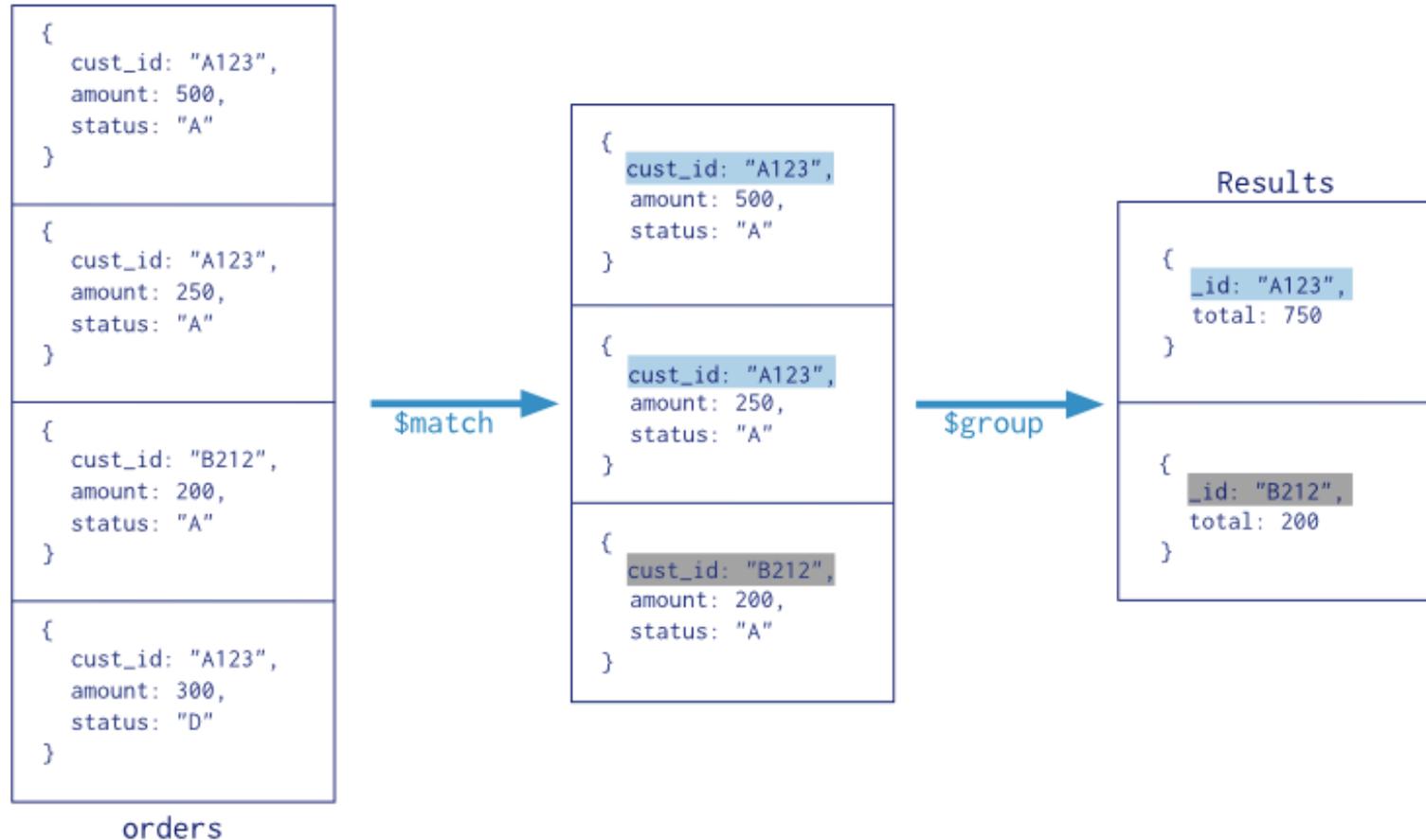
Operatori di aggregazione

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

Collection
↓
db.orders.aggregate(
 \$match phase → { \$match: { status: "A" } },
 \$group phase → { \$group: { _id: "\$cust_id", total: { \$sum: "\$amount" } } }
)



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



Interfaccia grafica per MongoDB

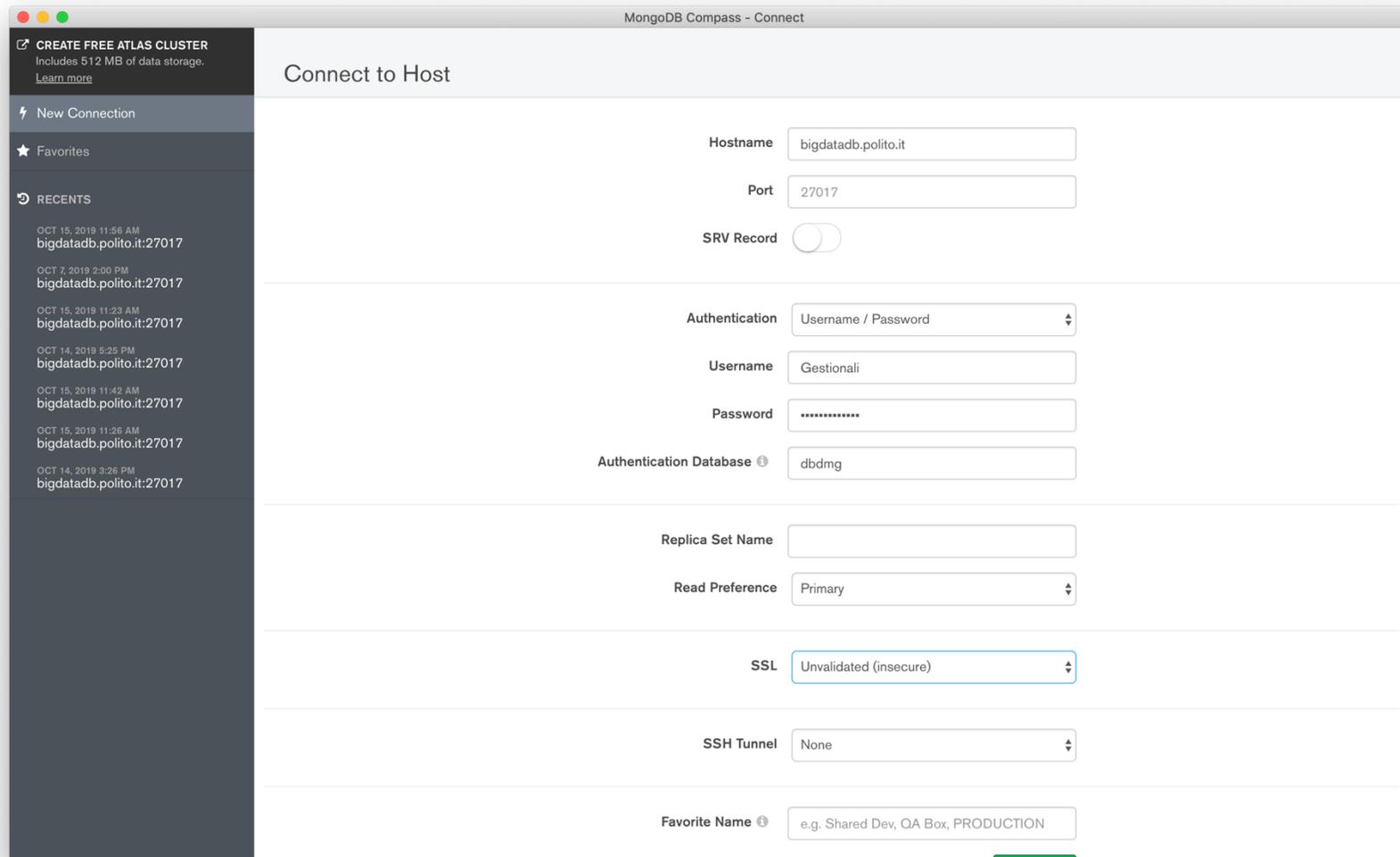
MongoDB Compass

- Consente di esplorare visivamente i dati.
- Disponibile per Linux, Mac, or Windows.
- Analizza i documenti e visualizza le strutture complesse all'interno delle collezioni
- Consente di visualizzare, comprendere e lavorare con i dati geo spaziali.



MongoDB Compass

- Si connette a un'istanza locale o remota di MongoDB.



The screenshot shows the 'Connect to Host' dialog in MongoDB Compass. The window title is 'MongoDB Compass - Connect'. On the left, there is a sidebar with a 'CREATE FREE ATLAS CLUSTER' button, a 'New Connection' button, and a 'RECENTS' list containing several entries for 'bigdatadb.polito.it:27017'. The main area is titled 'Connect to Host' and contains the following fields:

- Hostname:** bigdatadb.polito.it
- Port:** 27017
- SRV Record:**
- Authentication:** Username / Password
- Username:** Gestionali
- Password:** [Redacted]
- Authentication Database:** dbdmg
- Replica Set Name:** [Empty]
- Read Preference:** Primary
- SSL:** Unvalidated (insecure)
- SSH Tunnel:** None
- Favorite Name:** e.g. Shared Dev, QA Box, PRODUCTION

MongoDB Compass

- Consente di avere una panoramica dei dati sotto forma di lista di documenti o tabella strutturata.

The screenshot displays the MongoDB Compass interface for a cluster named 'My Cluster'. The left sidebar shows the database structure with '1 DBS' and '2 COLLECTIONS'. The 'dbdmg' database is expanded to show 'Bookings' and 'Parkings' collections. The main window is focused on the 'Parkings' collection, showing a list of 100 documents. The document list includes columns for '_id' and 'ObjectId'. A detailed view of a document is shown on the right, displaying fields such as '_id', 'plate', 'fuel', 'vendor', 'final_time', 'loc', 'init_time', 'vin', 'smartPhoneRequired', 'init_date', 'exterior', 'address', 'interior', 'final_date', 'engineType', and 'city'. The interface also shows statistics for the collection: 100 documents, 48.4KB total size, 496B average size, 5 indexes, 55.9KB total index size, and 11.2KB average index size.

MongoDB Compass

- Analizza i documenti e i loro attributi.
- Supporta nativamente le coordinate geo spaziali.

The screenshot displays the MongoDB Compass interface for a cluster named 'My Cluster'. The database 'bigdatadb.polito.it:27017' is selected, and the collection 'dbdmg.Parkings' is open. The 'Schema' tab is active, showing the 'loc' field with a 'coordinates' type. A map visualization shows the distribution of parking locations around Turin, Italy, with blue dots representing individual data points. The map includes labels for 'Collegno', 'LIMA', 'Turin', 'Beinasco', and 'San Mauro Torinese'. The 'plate' field is shown as an 'int32' type. A histogram at the bottom right shows the distribution of the 'plate' values, with a minimum of 1 and a maximum of 442. The interface also displays document statistics: 100 documents, 48.4KB total size, 496B average size, and 5 indexes with a total size of 55.9KB and an average size of 11.2KB. A query filter is present, and the 'ANALYZE' button is highlighted.

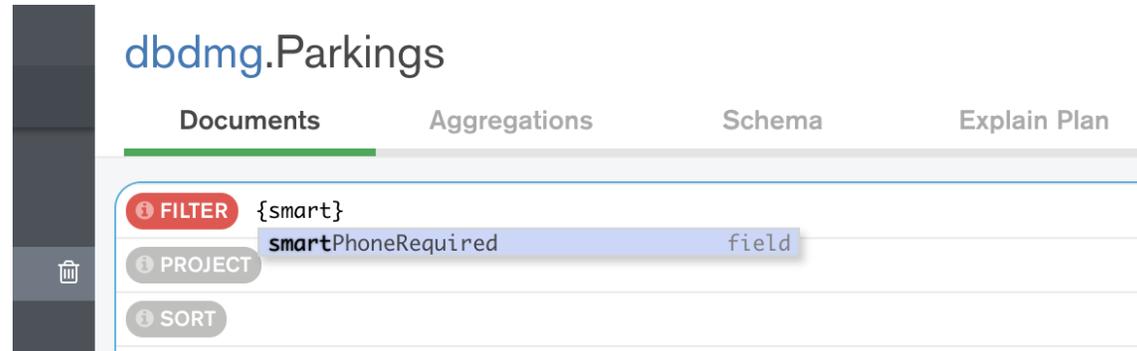
MongoDB Compass

- Consente di creare visivamente le interrogazioni ponendo delle condizioni sui dati.

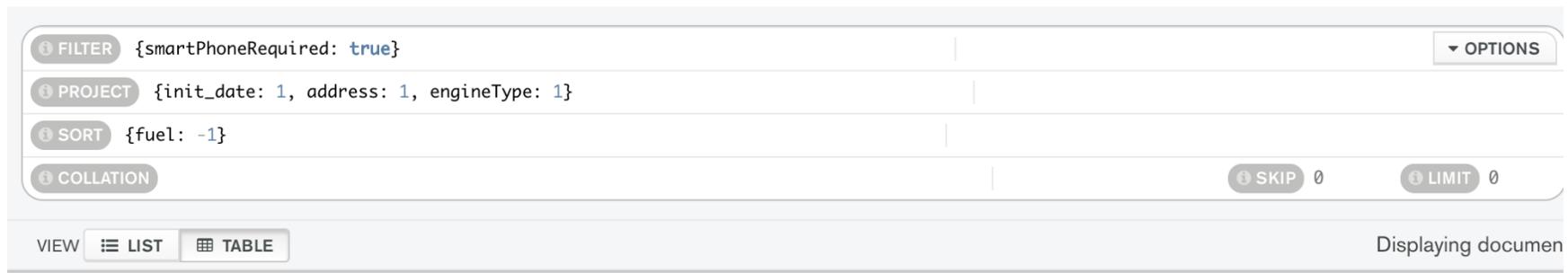
The screenshot displays the MongoDB Compass interface for a cluster named 'bigdatadb.polito.it:27017'. The database 'dbdmg.Parkings' is selected, showing 100 documents and 5 indexes. A query filter is applied: `{interior: 'GOOD', loc: {$geoWithin: { $centerSphere: [[7.664417178826483, 45.06173368230694], 0.0005190081853820`. The interface shows the schema for 'interior' (string) and 'loc' (coordinates). A map visualization is shown, highlighting a cluster of orange dots around Turin, Italy, within a red circle. The map includes labels for various locations like Collegno, LIMA, Torino, Beinasco, Irbassano, Nichelino, Moncalieri, Trofarello, Chieri, and San Mauro Torinese. The mapbox logo and a 'Drag to Build a Query' prompt are visible at the bottom of the map.

MongoDB Compass

- Auto-completamento abilitato di default.



- Permette di costruire le interrogazioni passo passo.



MongoDB Compass

- Analizza le performance di ogni interrogazione e fornisce suggerimenti per velocizzarla.

The screenshot displays the MongoDB Compass interface for a cluster named 'bigdatadb.polito.it:27017'. The current database is 'dbdmg' and the collection is 'Parkings'. The interface shows a query in the 'Explain Plan' tab with the following details:

- Query:** `{interior: 'GOOD', loc: { $geoWithin: { $centerSphere: [[7.664417178826483, 45.06173368230694], 0.0005190081853820] } } }`
- Documents:** 100 (Total Size: 48.4KB, Avg. Size: 496B)
- Indexes:** 5 (Total Size: 55.9KB, Avg. Size: 11.2KB)

The 'Query Performance Summary' section provides the following metrics:

- Documents Returned: 97
- Index Keys Examined: 0
- Documents Examined: 100
- Actual Query Execution Time (ms): 0
- Sorted in Memory: yes
- Warning: No index available for this query.

The execution plan shows a 'PROJECTION' stage with the following details:

- nReturned: 97
- Execution Time: 0 ms
- Transform by: `{ "init_date": 1, "address": 1, "engineType": 1 }`

The execution plan also shows a 'SORT' stage with the following details:

- nReturned: 97
- Execution Time: 0 ms

MongoDB Compass

- Consente di specificare vincoli.
- Trova i documenti incompatibili.

The screenshot shows the MongoDB Compass interface for a cluster named 'bigdatadb.polito.it:27017'. The current database is 'dbdmg' and the collection is 'Parkings'. The 'Validation' tab is active, showing a validation action set to 'ERROR' and a validation level set to 'STRICT'. The validation schema is displayed in a code editor, showing a \$jsonSchema with required fields 'exterior', 'interior', 'vendor', and 'fuel'. The 'vendor' field is a string, and the 'fuel' field is an integer. Below the schema, there are two sections: 'Sample Document That Passed Validation' and 'Sample Document That Failed Validation'. The 'Sample Document That Passed Validation' section shows a document with fields: _id, plate, fuel, vendor, final_time, loc, init_time, vin, and smartPhoneRequired. The 'Sample Document That Failed Validation' section is empty, indicating no documents failed validation.

```
1 - {
2   $jsonSchema: {
3     required: ["exterior", "interior", "vendor", "fuel"],
4     properties: {
5       vendor: {
6         bsonType: "string",
7         description: "must be a string"
8       },
9       fuel: {
10        bsonType: "int",
11        description: "must be an integer number"
12      },
13     }
14   }
15 }
```

Validation modified

Validation Action: **ERROR** Validation Level: **STRICT**

Validation modified

✓ Sample Document That Passed Validation

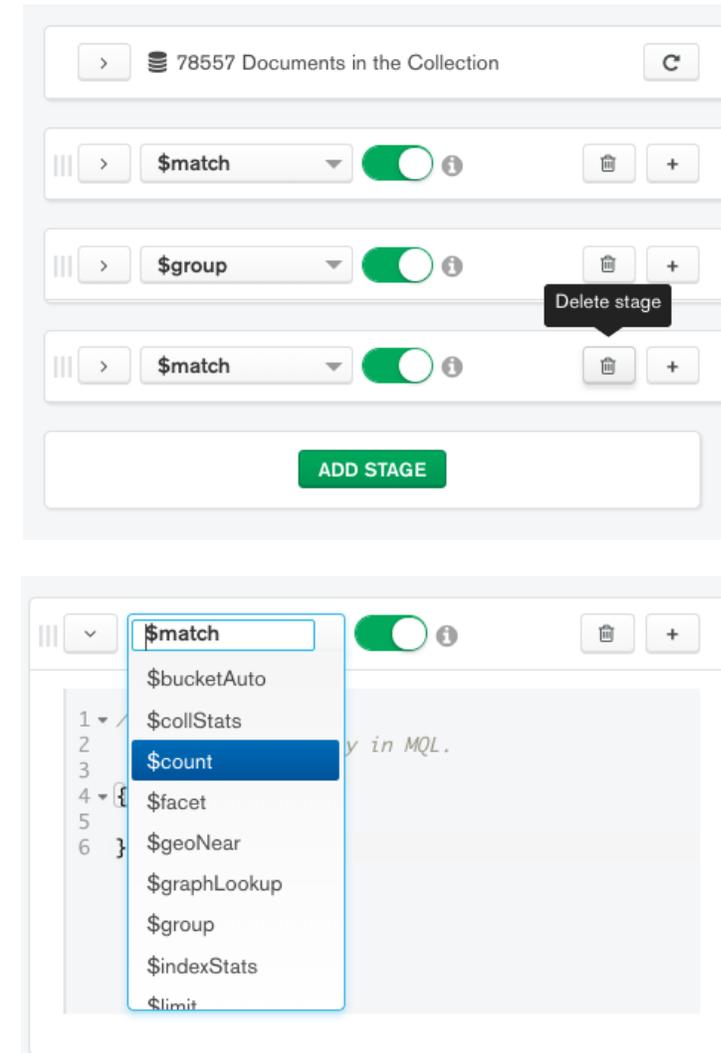
```
_id: ObjectId("59bef0cd2ad8532c2a60093d")
plate: 442
fuel: 37
vendor: "car2go"
final_time: 1505685847
loc: Object
init_time: 1505685697
vin: "VIN442"
smartPhoneRequired: true
```

✗ Sample Document That Failed Validation

No Preview Documents

MongoDB Compass: Aggregazione

- Consente di creare una a pipeline costituita da più fasi di aggregazione.
- Definisce dei filtri e degli attributi aggregati per ogni operatore.



MongoDB Compass: Fasi di aggregazione

```
1 ▾ /**
2   * _id - The id of the group.
3   * field1 - The first field name.
4   */
5 ▾ {
6   _id: "$vendor",
7   total: {
8     $sum: 1
9   }
10 }
```

Output after \$group stage (Sample of 2 documents)

```
_id: "car2go"
total: 48423
```

```
_id: "enjoy"
total: 30134
```

MongoDB Compass: Fasi di aggregazione

```
1 /**
2  * _id - The id of the
3  * field1 - The first
4  */
5 {
6   id: "$vendor",
7   total: {
8     $sum: 1
9   }
10 }
```

Il campo `_id` corrisponde al parametro della `GROUP BY` in SQL. Gli altri campi contengono gli attributi richiesti per ciascun gruppo.

Output after \$group stage (Sample of 2 documents)

<code>_id: "car2go"</code> <code>total: 48423</code>	<code>_id: "enjoy"</code> <code>total: 30134</code>
---	--

Un gruppo per ciascun "vendor".

MongoDB Compass: Pipeline

The screenshot shows the MongoDB Compass interface with a pipeline consisting of two stages: `$group` and `$match`.

Stage 1: \$group

```
1 /**
2  * _id - The id of the group.
3  * field1 - The first field name.
4  */
5 {
6   _id: "$vendor",
7   total: { $sum: 1 },
8   avg_fuel : { $avg : "$fuel" }
9 }
10
```

Output after \$group stage (Sample of 2 documents):

<code>_id: "car2go"</code> <code>total: 48423</code> <code>avg_fuel: 64.88284492906264</code>	<code>_id: "enjoy"</code> <code>total: 30134</code> <code>avg_fuel: 61.03381562354815</code>
---	--

Stage 2: \$match

```
1 /**
2  * query - The query in MQL.
3  */
4 {
5   avg_fuel: { $gt: 63 },
6   total : { $gt : 35000 }
7 }
```

Output after \$match stage (Sample of 1 document):

<code>_id: "car2go"</code> <code>total: 48423</code> <code>avg_fuel: 64.88284492906264</code>

Prima fase: raggruppamento per vendor

Seconda fase: condizione sui campi create nella fase precedente (avg_fuel, total).