




Basi Dati NoSQL

Introduzione a MongoDB

DBG

1



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


DBG

2

2

Terminologia – Concetti a confronto

Basi dati relazionali	Mongo DB
Tabella	Collezione
Record	Documento
Colonna	Campo


4

4

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.

```

{
  _id: <ObjectID1>,
  username: "123xyz",
  contact: {
    phone: 1234567890,
    email: "xyz@email.com",
  },
  access: {
    level: 5,
    group: "dev",
  }
}
    
```

} Embedded Sub-Document
} Embedded Sub-Document


5

5

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.

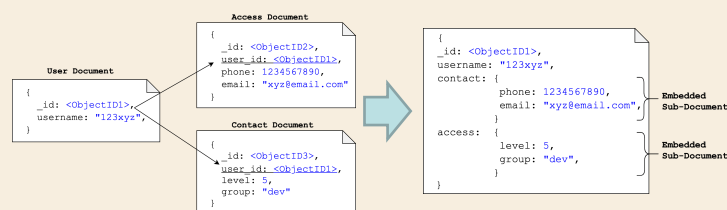


6

6

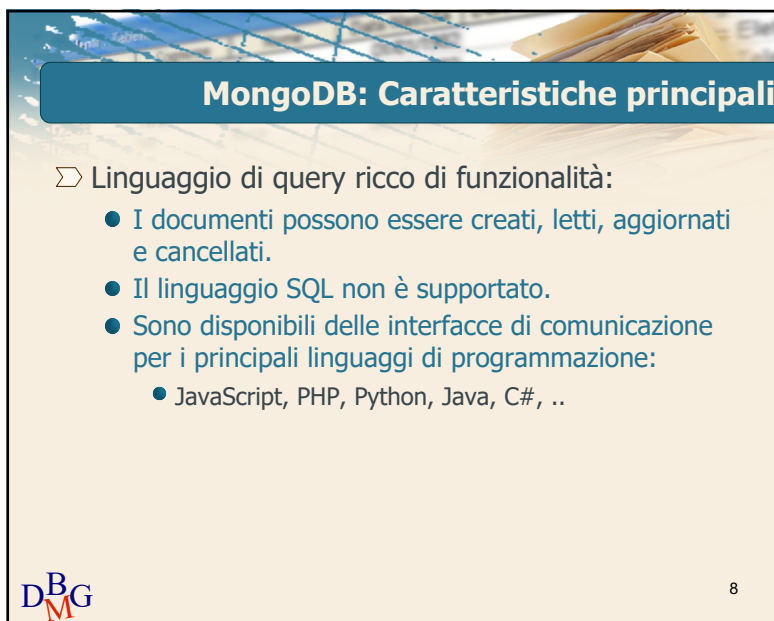
MongoDB: design dei documenti

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



7

7



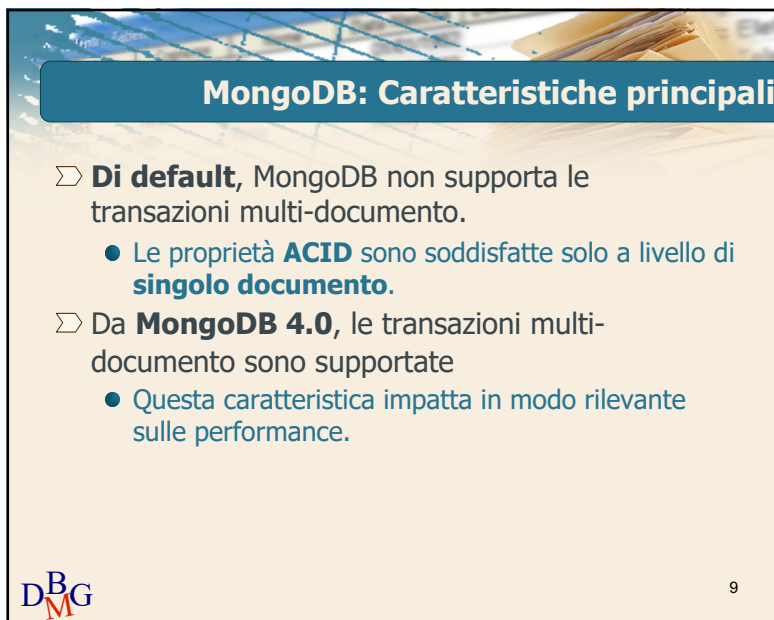
MongoDB: Caratteristiche principali

- ▷ 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#, ..

DBG

8

8



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.

DBG

9

9

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.

 10

10

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

 11

11

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.

➤ Replica asincrona

➤ Failover automatico

- Quando il nodo principale smette di funzionare, uno di quelli secondari inizia la procedura di sostituzione.

```

graph TD
    CA[Client Application Driver] -- Write --> P[Primary]
    P -- Replication --> S1[Secondary]
    P -- Replication --> S2[Secondary]
    CA -- "Read with Read Preference secondary" --> P
    
```

12

12

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

```


graph TD
    CA[Client Application Driver] -- Write --> P[Primary]
    P -- Replication --> S1[Secondary]
    P -- Replication --> S2[Secondary]
    CA -- "Read with Read Preference secondary" --> P
    
```

13

13

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> 14


14

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> 15

15




MongoDB

Operatori per selezionare i dati

DBG

16



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()


DBG

17

17

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>


18


18

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)


19


19

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)


20

20

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>
--	---

Condizioni (WHERE)

Selezione (SELECT)

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


21

21

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

22

22

MongoDB: operatori di confronto (>)

MySQL	MongoDB	Descrizione
>	\$gt	Maggiore di

<pre>SELECT * FROM people WHERE age > 25</pre>	<pre>db.people.find({ age: { \$gt: 25 } })</pre>
---	--

23

23

MongoDB: operatori di confronto (>=)

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

<pre>SELECT * FROM people WHERE age >= 25</pre>	<pre>db.people.find({ age: { \$gte: 25 } })</pre>
--	---



24

24

MongoDB: operatori di confronto (<)

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

<pre>SELECT * FROM people WHERE age < 25</pre>	<pre>db.people.find({ age: { \$lt: 25 } })</pre>
---	--


25

25

MongoDB: operatori di confronto (<=)

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

<pre>SELECT * FROM people WHERE age <= 25</pre>	<pre>db.people.find({ age: { \$lte: 25 } })</pre>
--	---



26

26

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

<pre>SELECT * FROM people WHERE age = 25</pre>	<pre>db.people.find({ age: { \$eq: 25 } })</pre>
--	--


27

27

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

<pre>SELECT * FROM people WHERE age != 25</pre>	<pre>db.people.find({ age: { \$neq: 25 } })</pre>
---	---



28

28

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



29

29

MongoDB: operatori condizionali (AND)

MySQL	MongoDB	Descrizione
AND	,	Entrambe soddisfatte

<pre>SELECT * FROM people WHERE status = "A" AND age = 50</pre>	<pre>db.people.find({ status: "A", age: 50 })</pre>
---	---



30

30

MongoDB: operatori condizionali (OR)

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

<pre>SELECT * FROM people WHERE status = "A" OR age = 50</pre>	<pre>db.people.find({ \$or: [{ status: "A" } , { age: 50 }] })</pre>
--	--


31

31

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>

32

32

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>
--	---

33


33

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>
---	---


34


34

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>


35

35




MongoDB

Inserire, aggiornare e cancellare documenti

DBG

36



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()

DBG


37

37

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>
--	---



38

38

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"}
] );
```


39

39

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>



40

40

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>
---	--

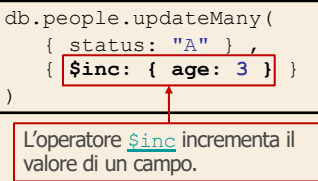


41

41

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.

42

42

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()

43

43

MongoDB: cancellare documenti

MySQL clause	MongoDB operator
DELETE FROM	deleteMany()
DELETE FROM people WHERE status = "D"	db.people.deleteMany({ status: "D" })

44

44

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({})

45

45




MongoDB

Indici

DBG

46



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).

DBG

47

47

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

 48

48


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`, ..

 49

49

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})`

50

50

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]`

51

51

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`



52

52

MongoDB: operatori geo spaziali

- ▷ Sintassi di `$near`:

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



53

53



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`

DBG

54

54



MongoDB

Operatori di aggregazione

DBG

55

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.



56

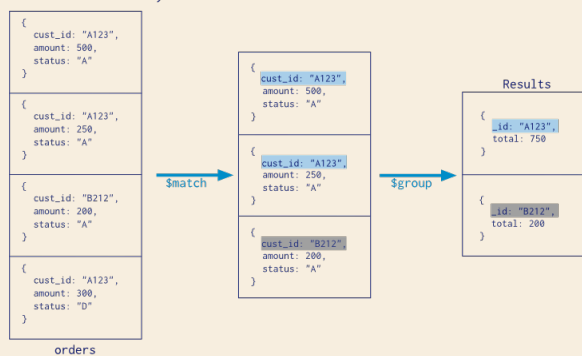
56

Aggregazione su MongoDB

```

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

```



57

57

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" }
    }
  }
] )

```

DBG

58

58

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" }
    }
  }
] )

```

DBG

59

59

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

60

60

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 } } }
] )
  
```

61

61

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
        
```

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

```

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

62

62

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
        
```

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

```

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

Condizioni: specificare le
condizioni come nel
campo HAVING

63

63



MongoDB Compass

Interfaccia grafica per Mongo DB



64



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.



65

MongoDB Compass

D
B
M

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

68

68

MongoDB Compass

D
B
M

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

69

69

MongoDB Compass

dbdmg.Parkings

Documents Aggregations Schema Explain Plan

1 FILTER {smart: true}

1 PROJECT {smartPhoneRequired: 1} field

1 SORT {fuel: -1}

1 COLLATION

SKIP 0 LIMIT 0

VIEW LIST TABLE Displaying document

➤ Auto-completamento abilitato di default.

➤ Permette di costruire le interrogazioni passo passo.

DBG 70

70

MongoDB Compass

My Cluster

My Collections

dbdmg

Bookings

Parkings

dbdmg.Parkings

Documents Aggregations Schema Explain Plan Indexes Validation

occurenze: 100

size: 48.4KB

index: 5

size: 50.9KB

size: 11.2KB

1 FILTER {smart: true}

1 PROJECT {fuel: 1, address: 1, engineType: 1}

1 SORT {fuel: -1}

1 COLLATION

SKIP 0 LIMIT 0

View Details An VISUAL TREE MM JSON

Query Performance Summary

Documents Returned: 97

Documents Examined: 100

Actual Query Execution Time (ms): 0

Index Range Examined: 0

Sorts Memory: yes

No index available for this query.

PROJECTION

Returned by: {fuel: 1, address: 1, engineType: 1}

Execution Time: 0 ms

DETAILS

SORT

Returned by: {fuel: -1}

Execution Time: 0 ms

DETAILS

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

DBG 71

71

MongoDB Compass

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

```

1 {
2   $jsonSchema: {
3     metadata: { uri: "mongodb://localhost:27017/dbdmg.Parkings" },
4     properties: {
5       plate: {
6         type: "string",
7         description: "must be a string"
8       },
9       year: {
10        type: "int",
11        description: "must be an integer number"
12      }
13     }
14   }
15 }
    
```

Validation modified [CANCEL] [UPDATE]

Sample Document That Passed Validation

```

{
  "_id": "5f811381759e196c2aef32c3a98950f1",
  "plate": "442",
  "year": 2019,
  "owner": "12345678",
  "date_created": "2019-05-07",
  "last_updated": "2019-05-07",
  "year": 2019,
  "owner": "12345678"
}
    
```

Sample Document That Failed Validation

No Preview Documents

DBG
MG

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

72

72

MongoDB Compass: Aggregazione

78557 Documents in the Collection

\$match [ON]

\$group [ON]

\$match [ON]

ADD STAGE

Drop down menu options:

- \$match
- \$bucketAuto
- \$collStats
- \$count
- \$facet
- \$geoNear
- \$graphLookup
- \$group
- \$indexStats
- \$limit

DBG
MG

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

73

73

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
-------------------------------	------------------------------


74

74


MongoDB Compass: Fasi di aggregazione

```

1 //**
2 * _id - The id of
3 * field1 - The fir
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
-------------------------------	------------------------------



Il campo `_id` corrisponde al parametro della `GROUP BY` in SQL

Gli altri campi contengono gli attributi richiesti per ciascun gruppo.

75

Un gruppo per ciascun "vendor".

75

MongoDB Compass: Pipeline

Prima fase: raggruppamento per vendor

```
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": "car29p", "total": 68423, "avg_fuel": 64.88284492986264 }</code>	<code>{ "_id": "enjoy", "total": 38134, "avg_fuel": 61.83381562354815 }</code>
---	--

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

```
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": "car29p", "total": 68423, "avg_fuel": 64.88284492986264 }</code>

DBG
M

76