DISPENSA
SGQS modulo rev 01 09/09/2019

Mongo DB
Database Non Relazionali

Simone Leonardi
Basi di dati
Tecnico Sviluppo Software
2020/2021

Milioni di database relazionali utilizzati da applicazioni che funzionano molto bene, ma ...

Nuovi trend

Organizzare dati non strutturati o semi-strutturati

Salvare dataset di grandi dimensioni offrendo scalabilità e prestazioni in modo economico

Reinventare i database relazionali

IBM

ORACLE

Nuove architetture

DBMG

Sono emersi database non relazionali

neo4j

mongoDB.

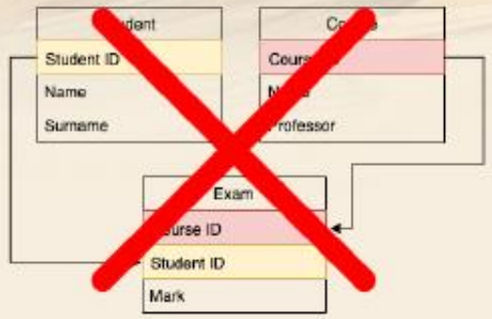
cassandra

CouchDB relax

redis

Not Only SQL

Nessuna operazione di join



Senza Schema

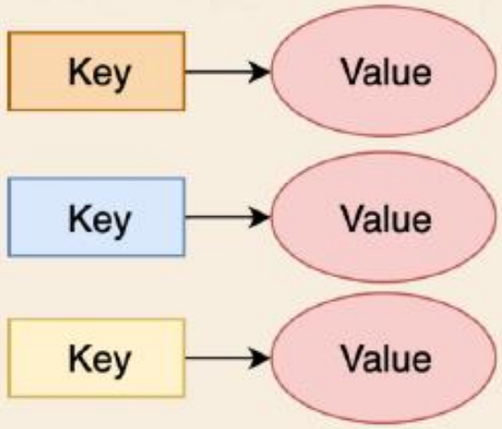
(nessuna tabella, schema implicito)

Student	Name	Surname
S123456		Rossi
S234567	Paolo	Bia

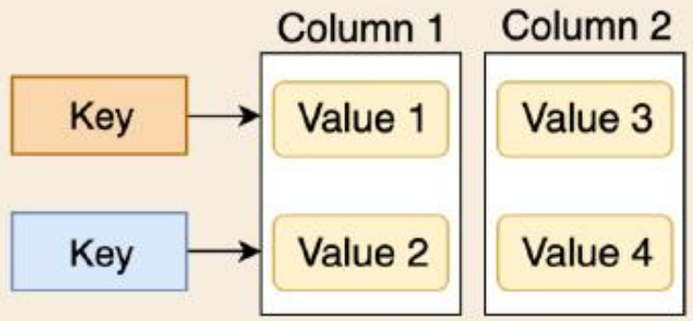
Database relazionale	Database non-relazionale
Basato su tabella, ogni record è una riga strutturata.	Soluzioni di archiviazione specializzate, ad esempio coppie di valori-chiave basate su documenti, database di grafi, archiviazione colonnare.
Schema predefinito per ogni tabella, modifiche consentite ma generalmente bloccanti (costose in ambienti distribuiti e live).	Senza schema: lo schema può cambiare dinamicamente per ogni documento, adatto per dati semi-strutturati o non strutturati.
Scalabile verticalmente, cioè tipicamente ridimensionato aumentando la potenza dell'hardware.	I database NoSQL sono scalabili orizzontalmente: vengono ridimensionati aumentando i server di database nel pool di risorse per ridurre il carico.
Utilizzo di SQL (Structured Query Language) per definire e manipolare i dati, un linguaggio molto versatile.	Linguaggi di query personalizzati, incentrati sul concetto di documenti, grafici e altre strutture di dati specializzate.

Database relazionale	Database non-relazionale
Adatto a query complesse, basato su join di dati.	Nessuna interfaccia standard per eseguire query complesse, nessun join.
Adatto per l'archiviazione di dati "piatta" e strutturata.	Adatto a dati complessi (ad esempio gerarchici), simili a JSON e XML.
Per esempio: MySql, Oracle, Sqlite, Postgres e Microsoft SQL Server.	**Per esempio:** MongoDB, BigTable, Redis, Cassandra, Hbase e CouchDB.

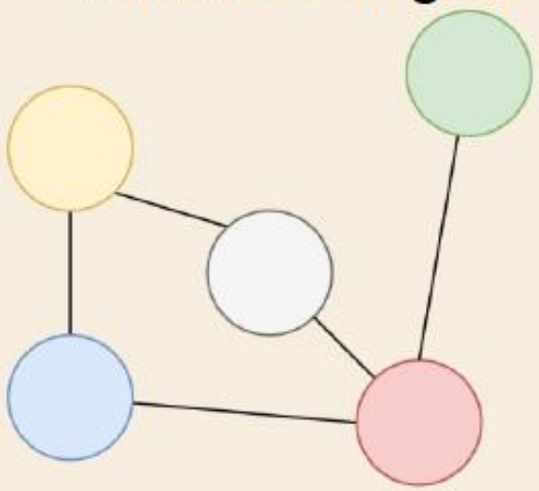
Chiave Valore



Orientati per Colonna



Database sui grafi



Basati sui documenti

```
{
first_name: "Mario",
last_name : "Rossi",
SSN: "AAAA00000000",
city: "Torino",
job: "Engineer",
cars: [
  {
    model: "Model S",
    year: "2018"
  },
  {
    model: "Model X",
    year: "2016"
  }
],
}
```

- Memorizza e recupera documenti
- I documenti sono coppie auto descrittive (**attributo = valore**)
  - Come `city: "Torino"`
- Le chiavi sono mappate nei documenti
- I documenti hanno una natura **eterogenea**
- Tra le soluzioni più utilizzate
- Esempi: **MongoDB**, CouchDB, RavenDB

```
{
first_name: "Mario",
last_name : "Rossi",
SSN: "AAAA00000000",
city: "Torino",
job: "Engineer",
cars: [
  {
    model: "Model S",
    year: "2018"
  },
  {
    model: "Model X",
    year: "2016"
  }
],
}
```

- JSON è un linguaggio indipendente per salvare e scambiare dati

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }  ],
```

Chiave

Valore

Chiave

Valore composto

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
```

Introduzione a MongoDB

Basi dati relazionali	Mongo DB
Tabella	Collezione
Record	Documento
Colonna	Campo

- 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",
         }
}
```

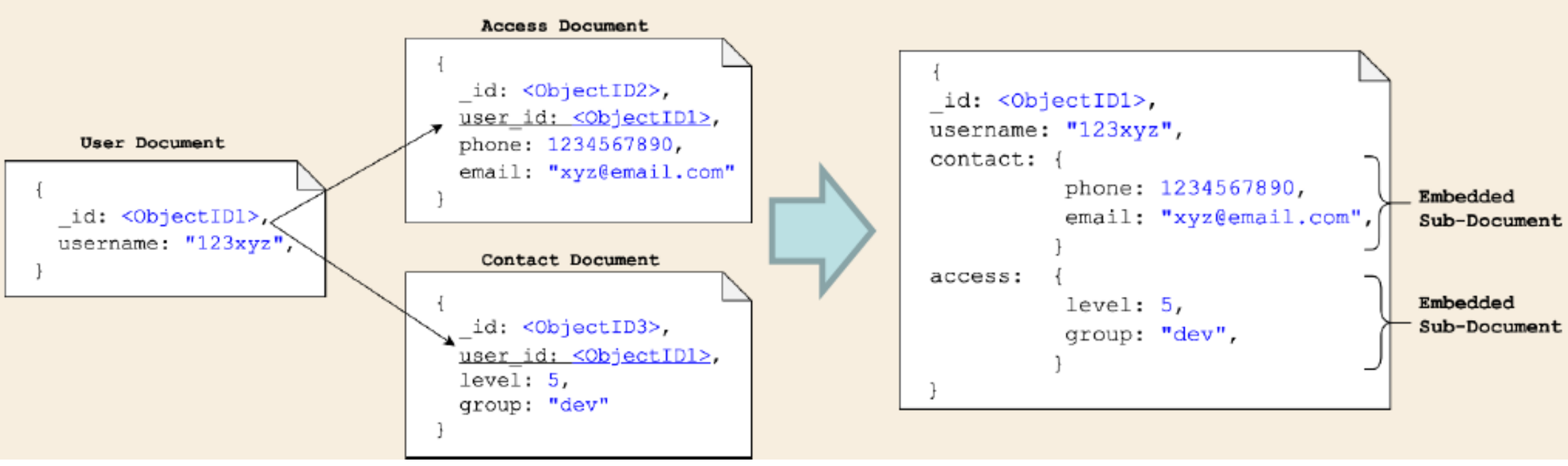
Embedded Sub-Document

Embedded Sub-Document

- 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.

Attenzione!

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



- 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#, ..

I casi d'uso più comuni di MongoDB includono:

- Internet of Things, Mobile, Analisi Real-Time, Personalizzazione, Dati geo spaziali.

Operatori per selezionare i dati

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()**

MySQL	MongoDB
SELECT	find()

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

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

MySQL	MongoDB
SELECT	find()

Condizioni (WHERE)

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

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

Selezione (SELECT)

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

```
SELECT *
FROM people
WHERE status = "A"
```

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

Condizioni (WHERE)

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

Condizioni (WHERE)

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

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

Selezione (SELECT)

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

- 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

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

- 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

MySQL	MongoDB	Descrizione
AND	,	Entrambe soddisfatte

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

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

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 }
   ]
}
)
```

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

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

- 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 })

Inserire, aggiornare e cancellare 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()`

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"
   }
)
```

- 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"}
   ] );
```

- 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>} })

`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.

- 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.

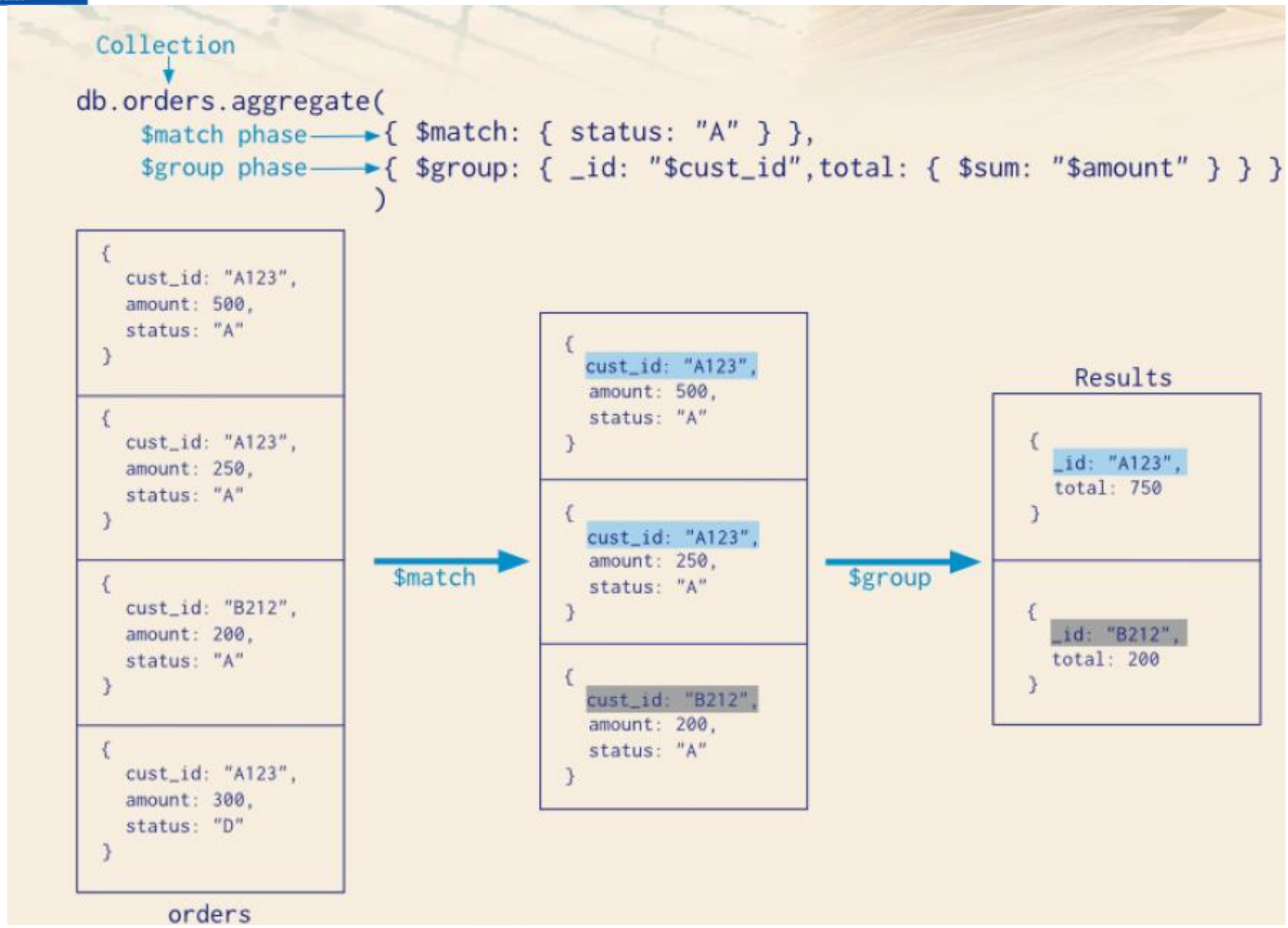
MySQL	MongoDB
`DELETE FROM`	`deleteMany()`

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

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

Operatori di aggregazione

- 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.


Collection
db.orders.aggregate(
$match phase
{ $match: { status: "A" } },
$group phase
{ $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
)
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
$match
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
$group
Results
{ _id: "A123", total: 750 }
{ _id: "B212", total: 200 }

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

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 } } }
] )
```

Altri esempi di operazioni di aggregazione (max, min, avg, …)

https://riptutorial.com/it/mongodb/example/24558/esempi-di-query-aggregate-utili-per-lavoro-e-apprendimento