Database noSQL e MongoDB

Outline

- Introduzione
- Tipologie di database noSQL
- MongoDB
  - CRUD
  - Relazioni fra documenti
  - Strutture ad albero
  - Aggregazioni
  - Esempi

Cenni storici

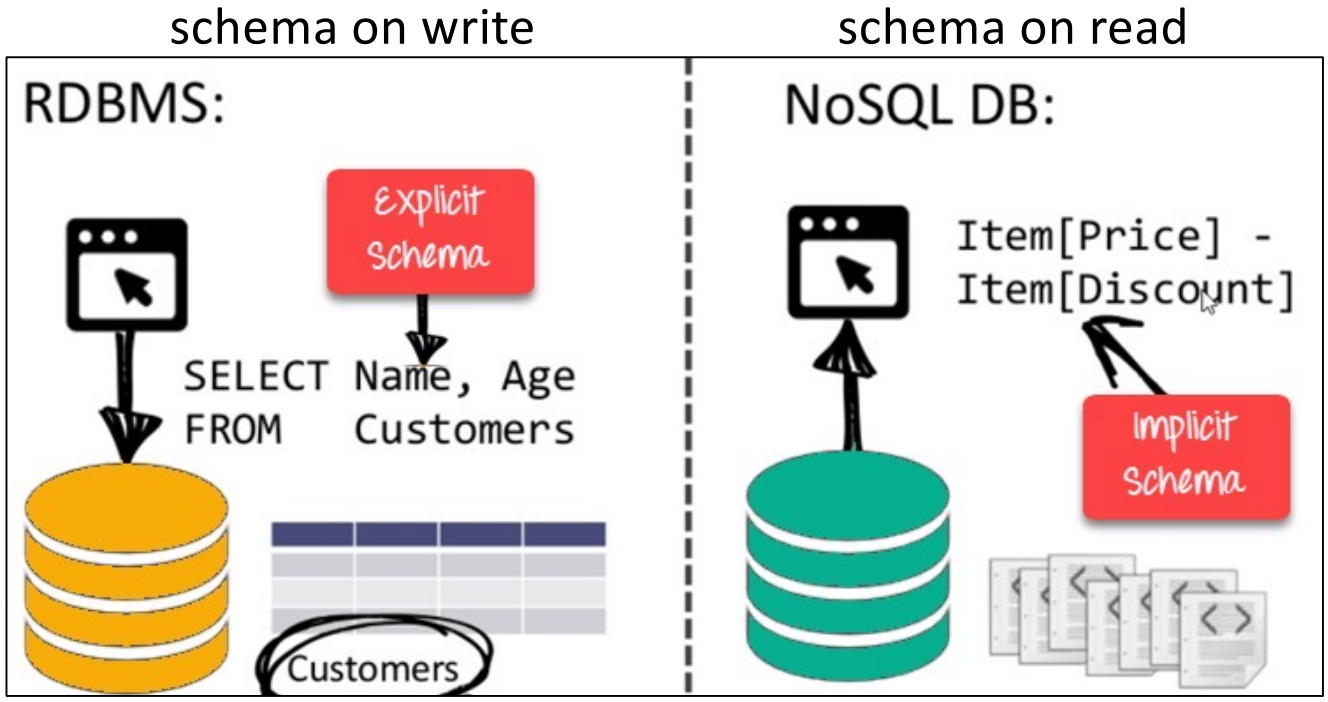
- Il termine “NoSQL” nasce nel 1998 - inizialmente si riferisce a DB relazionali che non utilizzano SQL
- ripreso nel 2009 diventa indicazione di DB non relazionali

Caratteristiche dei database noSQL

- noSQL: Not only SQL
  - Un database che non usa lo schema relazionale
  - Non usa tabelle con record di formato fissato
  - Utilizza strutture dati auto-consistenti o addirittura blob
  - Non necessita di ORM o di normalizzazione dei dati
  - Non utilizza linguaggi di query
  - Non utilizza vincoli di integrità referenziale e non ci sono transazioni

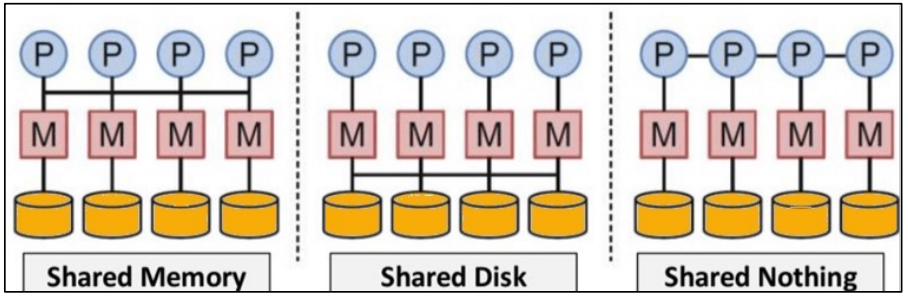
Caratteristiche dei database noSQL

- Non ha uno schema ovvero ha schemi rilassati
- Non richiede la definizione a priori dello schema
- Offre una serie di strutture dati per gestire i dati



Caratteristiche dei database noSQL

- Offre semplici API per la manipolazione anche a basso livello
- In genere offrono interfaccia HTTP REST e protocolli di testo per la comunicazione, ad es. JSON
- Interfaccia Web
- Ambiente distribuito che utilizza eventual consistency
- Architettura «Shared Nothing»



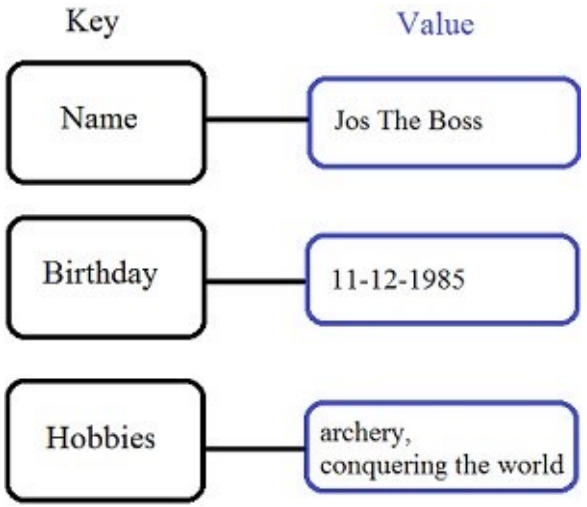
Tipologie dei database noSQL

- persistenza (non relazionale)
- senza schema fisso (schemaless)
- structured storage
- alta scalabilità
- orientati alle colonne, chiave/valore, orientati al documento, a grafo

db-engines.com/en/ranking

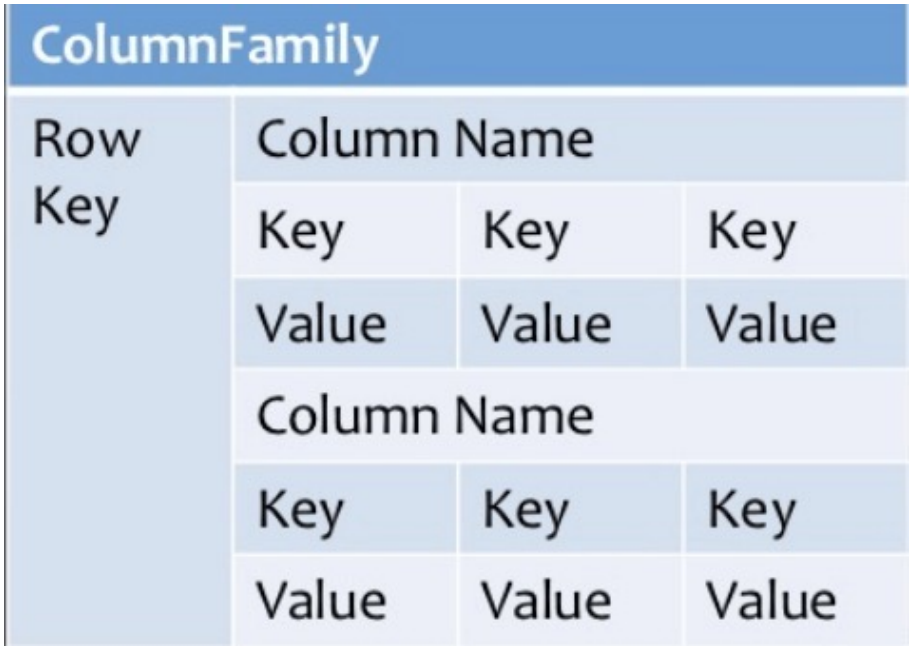
Tipologie dei database noSQL

- Coppie chiave-valore
  - I dati sono conservati come coppie chiave-valore
    - Le chiavi sono univoche
    - I valori possono essere eterogenei: stringhe, JSON, BLOB (Binary Large OBjects)
  - Le chiavi sono implicitamente lo schema: non è richiesto schema a priori
    - Es. gestione dei carrelli elettronici
  - Amazon Dynamo



Tipologie dei database noSQL

- Database orientati alle colonne
  - Molto efficienti per query cumulative (conteggio, somma, media ...) lungo le singole colonne
  - Basati su Google BigTable
  - Cassandra, Hbase
  - I dati sono indicizzati direttamente sulle colonne
  - Possono essere raggruppati per «righe» o più in generale in «gruppi di colonne» che non debbono coincidere necessariamente con un intero record



ROWID	Name	Birthday	Hobbies
1	Jos The Boss	11-12-1985	archery, conquering the world
2	Fritz von Braun	27-1-1978	building things, surfing
3	Freddy Stark		swordplay, lollygagging, archery
4	Delphine Thewiseone	16-9-1986	

Row-oriented lookup: from top to bottom and for every entry all columns are taken in memory.

Name	ROWID
Jos The Boss	1
Fritz Schneider	2
Freddy Stark	3
Delphine Thewiseone	4

Birthday	ROWID
11-12-1985	1
27-1-1978	2
16-9-1986	4

Hobbies	ROWID
archery	1, 3
conquering the world	1
building things	2
surfing	2
swordplay	3
lollygagging	3

A column-oriented database stores each column separately

Tipologie dei database NoSQL

- Database documentali
  - Ottimo per CMS, piattaforme di blogging, analytics, e-commerce ...
  - MongoDB
  - Ogni record è un «documento» rappresentato in un formato testo strutturato come JSON e/o XML

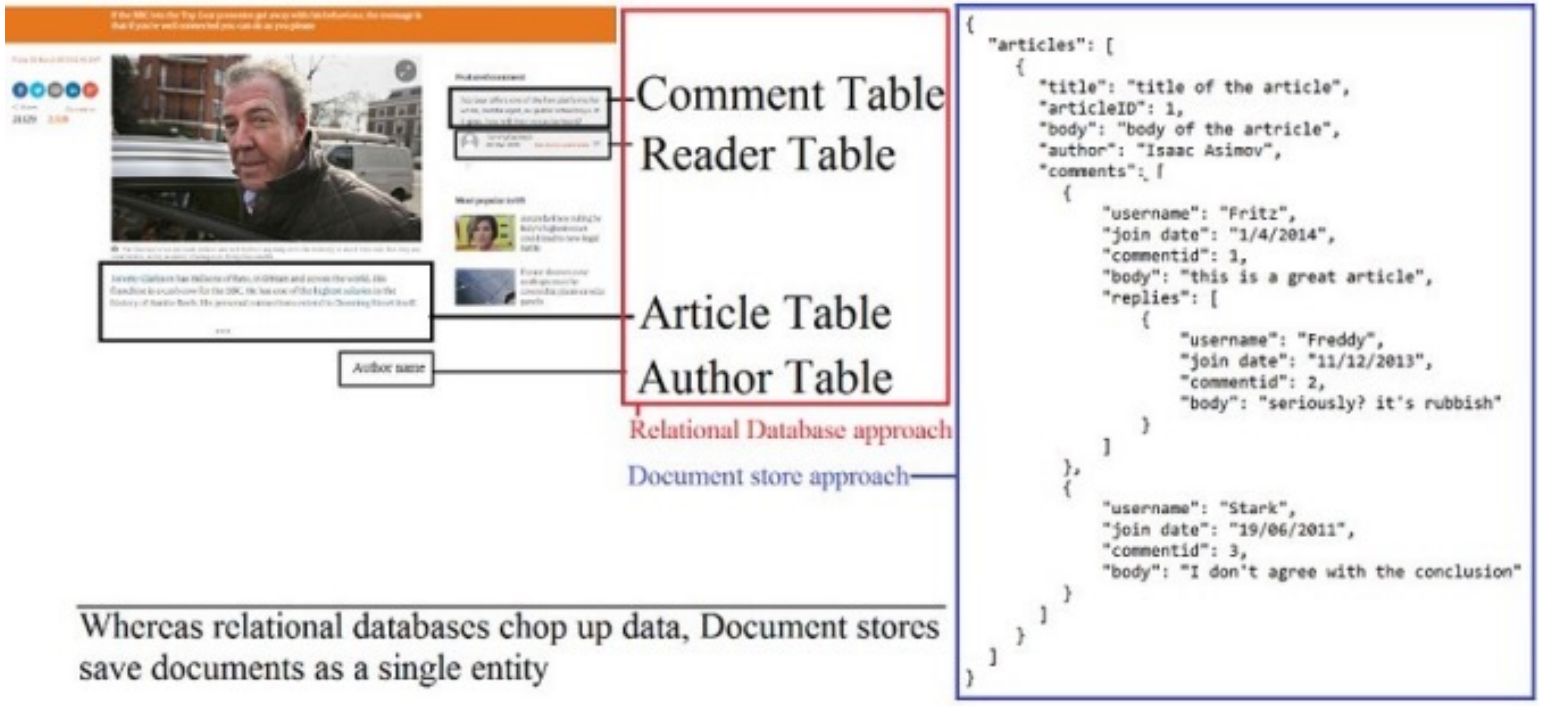
Relational data model

Highly-structured table organization with rigidly-defined data formats and record structure.

Document data model

Collection of complex documents with arbitrary, nested data formats and varying "record" format.

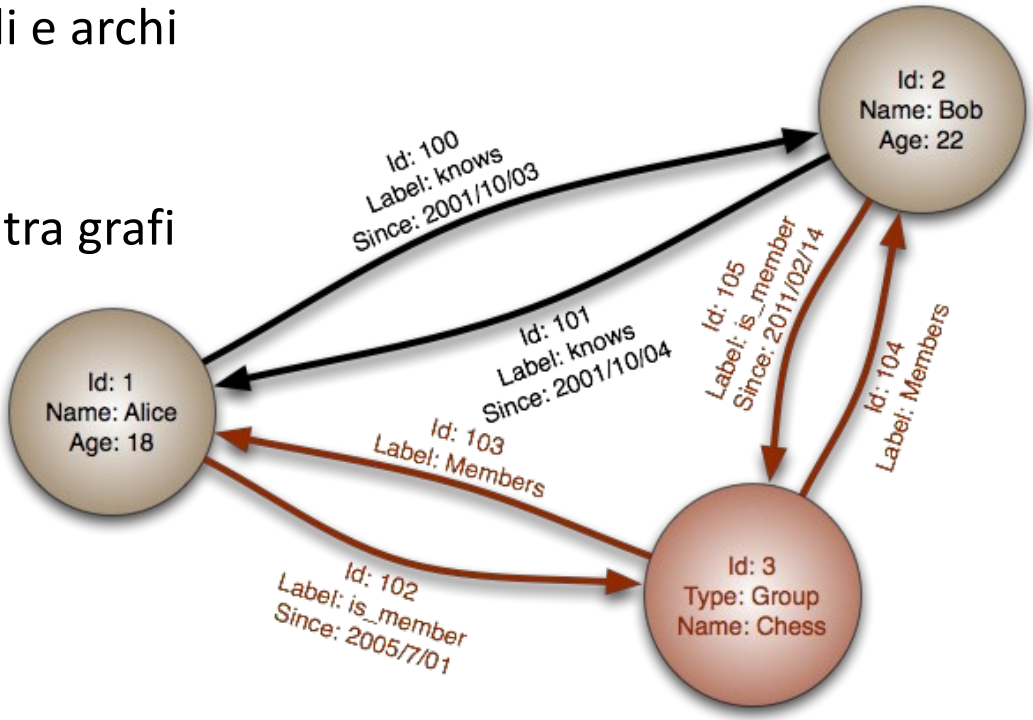
Tipologie dei database NoSQL



Whereas relational databases chop up data, Document stores save documents as a single entity

Tipologie dei database NoSQL

- Database a grafo
  - I dati sono rappresentati come nodi e archi etichettati di un grafo
  - Struttura «multirelazionale»
  - Le query si mappano in operazioni tra grafi (ad es. shortest path)
  - Neo4j



Implementazioni di database noSQL

- HBase (hbase.apache.org)
- Cassandra (cassandra.apache.org)
- **MongoDB** (mongodb.org)
- Neo4j (neo4j.com)
- OrientDB (orientdb.org)
- ...

Vantaggi dei database noSQL

- Possono essere usati sia per scopi di analytics sia come data lake
- Orientati ai Big Data
- Non c'è più un SPOF
- Performance e scalabilità orizzontale
- Gestione di dati strutturati, semi-strutturati o non strutturati
- Programmazione semplice in OOP
- Non necessitano di server dedicati ad alte performance
- Pensati per i database distribuiti
- Schemi sui dati non predefiniti che possono essere alterati senza interruzione del servizio

Svantaggi dei database noSQL

- Non c’è standardizzazione
- Limitate capacità di query
- Le tecnologie RDBMS sono molto più mature
- I sistemi open source non sono molto apprezzati/popolari nelle organizzazioni pubbliche o private

MongoDB

- Open-source
- Dati in formato JSON-Like (BSON)
  - (JSON: JavaScript Object Notation – json.org)
  - (BSON: Binary JavaScript Object Notation – bsonspec.org)
- Struttura flessibile
- Alta disponibiltà e scalabilità

https://www.mongodb.com/docs/manual/

MongoDB

- Disponibilità di MongoDB
  - **MongoDB Atlas**
    - The fully managed service for MongoDB deployments in the cloud
  - MongoDB Enterprise
    - The subscription-based, self-managed version of MongoDB
  - MongoDB Community
    - The source-available, free-to-use, and self-managed version of MongoDB
- Interazione con MongoDB
  - MongoDB shell (mongosh)
  - Driver per linguaggi di programmazione
    - **Python -> pymongo**

Terminologia e concetti

- Database → Database
- Tabella (Relazione) → Collezione
- Riga (Istanza) → Documento o BSON
- Colonna (Attributo) → Campo
- Join → Collegamento
- Raggruppamento → Aggregazione

CRUD: Create, Read, Update, and Delete

SQL vs. MongoDB

TABELLA

```
CREATE TABLE users (
id MEDIUMINT NOT NULL
AUTO_INCREMENT,
user_id Varchar(30),
age Number,
status char(1),
PRIMARY KEY (id)
)
```

COLLEZIONE

- Creata implicitamente alla prima operazione insert()
  - La chiave primaria _id è aggiunta automaticamente e quindi non specificata.

```
db.users.insert( {
   user_id: "abc123",
   age: 55,
   status: "A"
})
```

- Oppure si può creare esplicitamente

```
db.createCollection("users")
```

https://www.mongodb.com/docs/manual/tutorial/insert-documents/

SQL vs. MongoDB

```
ALTER TABLE users
ADD join_date DATETIME
```

- Non necessario!
- Si può aggiungere un campo ad un documento attraverso l'operazione di update() e l'operatore $set

```
db.users.update(
{ },
{ $set: { join_date: new Date() } },
{ multi: true }
)
```

https://www.mongodb.com/docs/manual/tutorial/update-documents/

SQL vs. MongoDB

```
ALTER TABLE users
DROP COLUMN join_date
```

```
db.users.update( { },
{ $unset: { join_date: "" } },
{ multi: true }
)
```

```
DROP TABLE users
```

```
db.users.drop()
```

MongoDB – db.collection.insert()

db.collection.insert(
<document or array of documents>, {
writeConcern: <document>,
ordered: <boolean> }
)
Es:

```
db.products.insert( { item: "card", qty: 15 } )

{ "_id" : ObjectId("5063114bd386d8fadbd6b004"), "item" : "card", "qty" : 15 }
```

SQL vs. MongoDB

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

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

```
SELECT * FROM users
```

```
db.users.find()
```

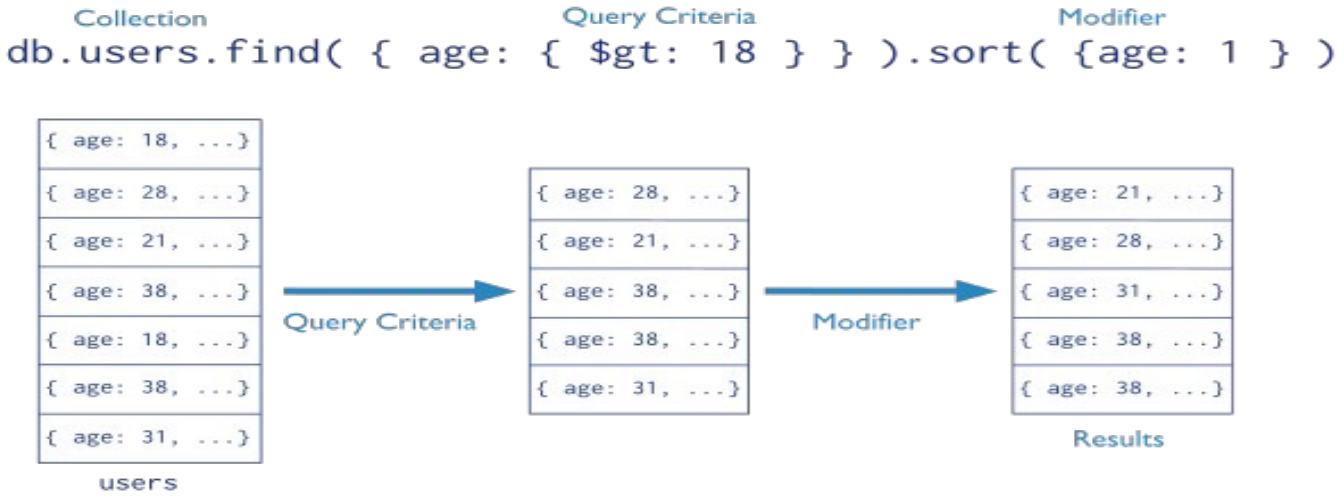
MongoDB - db.collection.find()

- db.collection.find(<criteria>, <projection>)

  https://www.mongodb.com/docs/manual/tutorial/query-documents/

Es:

```
db.products.find( { qty: { $gt: 25 } } )
```



SQL vs. MongoDB

```
SELECT id, user_id, status
FROM users
```

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

```
SELECT user_id, status
FROM users
```

```
db.users.find( { },
{ user_id: 1, status: 1,
_id: 0}
)
```

SQL vs. MongoDB

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

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

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

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

SQL vs. MongoDB

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

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

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

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

SQL vs. MongoDB

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

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

```
SELECT *
FROM users
WHERE status = "A"
ORDER BY user_id DESC
```

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

SQL vs. MongoDB

```
SELECT COUNT(*)
FROM users
```

```
db.users.find().count()
```

```
SELECT COUNT(user_id)
FROM users
```

```
db.users.find(
{ user_id: { $exists: true } } )
.count()
```

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

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

Operatori di confronto

Operatore	Descrizione
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$in	Matches any of the values specified in an array.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$nin	Matches none of the values specified in an array.

Operatori logici e di elemento

Operatore	Descrizione
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$not	Inverts the effect of a query expression and returns documents that do not match the query expression.
$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.
$exists	Matches documents that have the specified field.
$type	Selects documents if a field is of the specified type.

https://www.mongodb.com/docs/manual/reference/bson-types/

MongoDB - db.collection.update()

db.collection.update(
<query>,
<update>,
{
upsert: <boolean>,
multi: <boolean>,
writeConcern: <document>
})

https://www.mongodb.com/docs/manual/tutorial/update-documents/

SQL vs. MongoDB

```
UPDATE users
SET status = "C"
WHERE age > 25
```

```
db.users.update(
{ age: { $gt: 25 } },
{ $set: { status: "C" } },
{ multi: true }
)
```

```
UPDATE users
SET age = age + 3
WHERE status = "A"
```

```
db.users.update(
{ status: "A" } ,
{ $inc: { age: 3 } },
{ multi: true }
)
```

MongoDB - db.collection.delete()

db.collection.delete(<query>)

```
DELETE FROM users
WHERE status = "D"
```

```
db.users.delete(
{ status: "D" }
)
```

https://www.mongodb.com/docs/manual/tutorial/remove-documents/

Relazioni fra elementi (1-1)

```
Persona {
_id: "joe",
name: "Joe Bookreader"
}

Indirizzo {
person_id: "joe",
street: "123 Fake Street", city: "Faketon",
state: "MA",
zip: "12345"
}
```

```
Persona {
_id: "joe",
name: "Joe Bookreader"
address: {
  street: "123 Fake Street", city: "Faketon",
  state: "MA",
  zip: "12345"
} }
```

Relazioni fra elementi (1-N)

```
{ _id: "joe",
 name: "Joe Bookreader"
}
{ person_id: "joe",
 street: "123 Fake Street", city: "Faketon",
 state: "MA",
 zip: "12345"
}
{ person_id: "joe",
 street: "1 Some Other Street", city: "Boston",
 state: "MA",
 zip: "12345»
}
```

```
{ _id: "joe",
 name: "Joe Bookreader",
 addresses: [{
  street: "123 Fake Street",
  city: "Faketon",
  state: "MA",
  zip: "12345"
 }, {
  street: "1 Some Other Street",
  city: "Boston",
  state: "MA",
  zip: "12345"
} ]
}
```

Relazioni fra elementi (1-N)

```
{ title: "MongoDB: The Definitive Guide",
 author: [ "Kristina Chodorow", "Mike Dirolf" ],
 published_date: ISODate("2010-09-24"),
 pages: 216,
 language: "English",
 publisher: {
  name: "O'Reilly Media",
  founded: 1980,
  location: "CA"
}}

{ title: "50 Tips and Tricks for MongoDB Developer",
 author: "Kristina Chodorow",
 published_date: ISODate("2011-05-06"),
 pages: 68,
 language: "English",
 publisher: {
  name: "O'Reilly Media",
  founded: 1980,
  location: "CA"
}}
```

```
{name: "O'Reilly Media",
 founded: 1980,
 location: "CA",
 books: [12346789, 234567890, ...]
}

{_id: 123456789,
 title: "MongoDB: The Definitive Guide",
 author: [ "Kristina Chodorow", "Mike Dirolf" ],
 published_date: ISODate("2010-09-24"),
 pages: 216,
 language: "English"
}

{_id: 234567890,
 title: "50 Tips and Tricks for MongoDB Developer",
 author: "Kristina Chodorow",
 published_date: ISODate("2011-05-06"),
 pages: 68,
 language: "English"
}
```

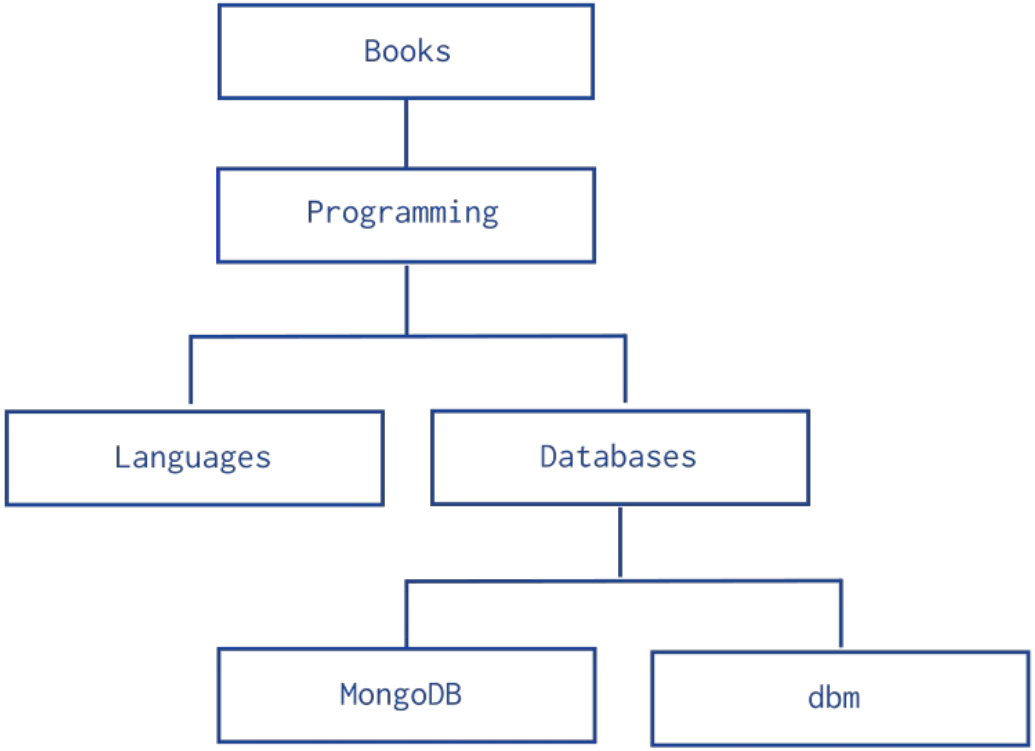
Relazioni fra elementi (1-N)

```
{ _id: "oreilly",
 name: "O'Reilly Media",
 founded: 1980,
 location: "CA"
}

{ _id: 123456789,
 title: "MongoDB: The Definitive Guide",
 author: [ "Kristina Chodorow", "Mike Dirolf" ],
 published_date: ISODate("2010-09-24"),
 pages: 216,
 language: "English",
 publisher_id: "oreilly"
}
```

```
{ _id: 234567890,
 title: "50 Tips and Tricks for MongoDB Developer",
 author: "Kristina Chodorow",
 published_date: ISODate("2011-05-06"),
 pages: 68,
 language: "English",
 publisher_id: "oreilly"
}
```

Strutture ad albero



Strutture ad albero (Parent References)

```
db.categories.insert( { _id: "MongoDB", parent: "Databases" } )
db.categories.insert( { _id: "dbm", parent: "Databases" } )
db.categories.insert( { _id: "Databases", parent: "Programming" } )
db.categories.insert( { _id: "Languages", parent: "Programming" } )
db.categories.insert( { _id: "Programming", parent: "Books" } )
db.categories.insert( { _id: "Books", parent: null } )
```

```
db.categories.findOne( { _id: "MongoDB" } ).parent
db.categories.createIndex( { parent: 1 } )
db.categories.find( { parent: "Databases" } )
```

https://www.mongodb.com/docs/manual/indexes/

Strutture ad albero (Child References)

```
db.categories.insert( { _id: "MongoDB", children: [] } )
db.categories.insert( { _id: "dbm", children: [] } )
db.categories.insert( { _id: "Databases", children: [ "MongoDB", "dbm" ] } )
db.categories.insert( { _id: "Languages", children: [] } )
db.categories.insert( { _id: "Programming", children: [ "Databases",
"Languages" ] } )
db.categories.insert( { _id: "Books", children: [ "Programming" ] } )

db.categories.findOne( { _id: "Databases" } ).children
db.categories.createIndex( { children: 1 } )
db.categories.find( { children: "MongoDB" } )
```

Strutture ad albero (Array of Ancestors)

db.categories.insert({ _id: "MongoDB", ancestors: ["Books", "Programming", "Databases"], parent: "Databases" })

db.categories.insert({ _id: "dbm", ancestors: ["Books", "Programming", "Databases"], parent: "Databases" })

db.categories.insert({ _id: "Databases", ancestors: ["Books", "Programming"], parent: "Programming" })

db.categories.insert({ _id: "Languages", ancestors: ["Books", "Programming"], parent: "Programming" })

db.categories.insert({ _id: "Programming", ancestors: ["Books"], parent: "Books" })

db.categories.insert({ _id: "Books", ancestors: [ ], parent: null }) db.categories.findOne({ _id: "MongoDB" }).ancestors

db.categories.createIndex({ ancestors: 1 })

db.categories.find({ ancestors: "Programming" })

Aggregazioni

- Pipelines

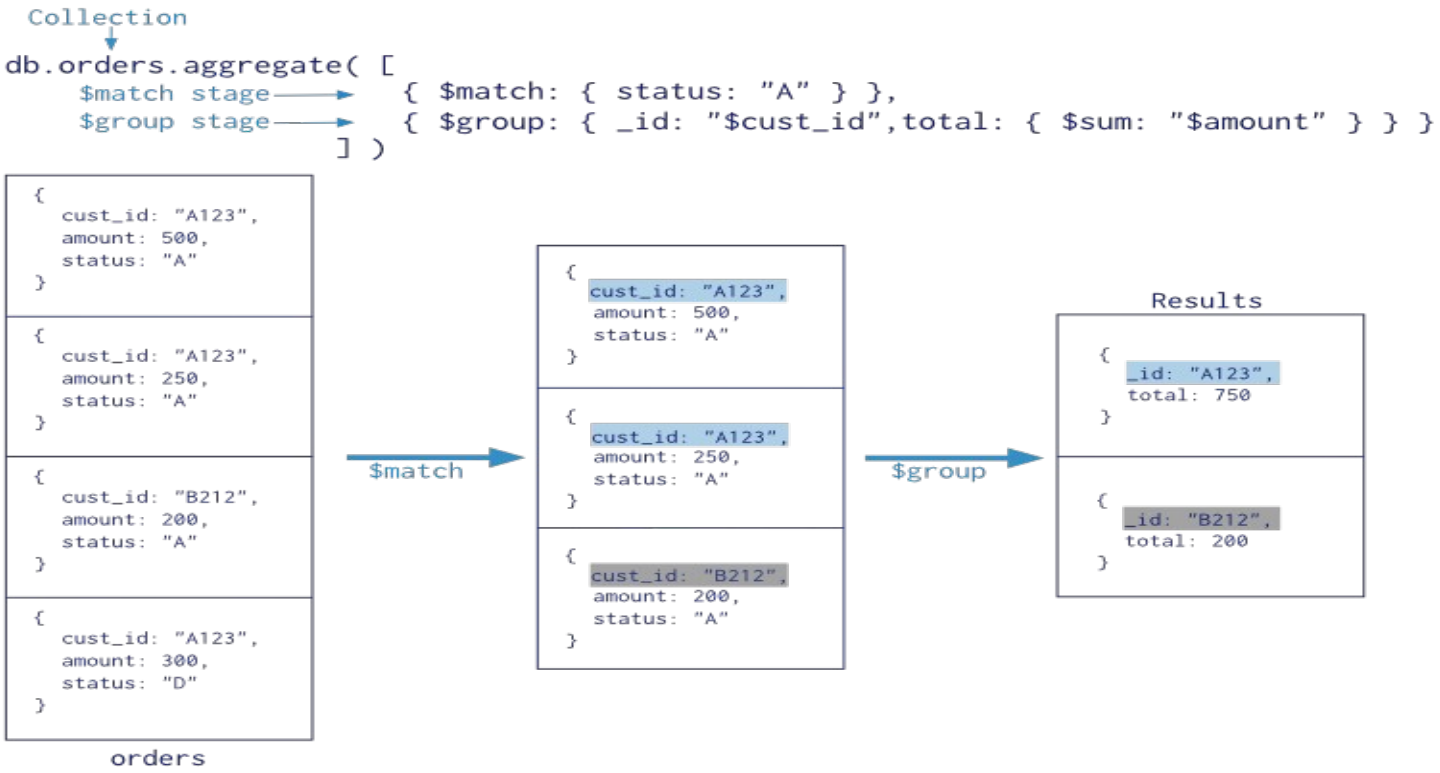
  https://www.mongodb.com/docs/manual/aggregation/

- Map-Reduce

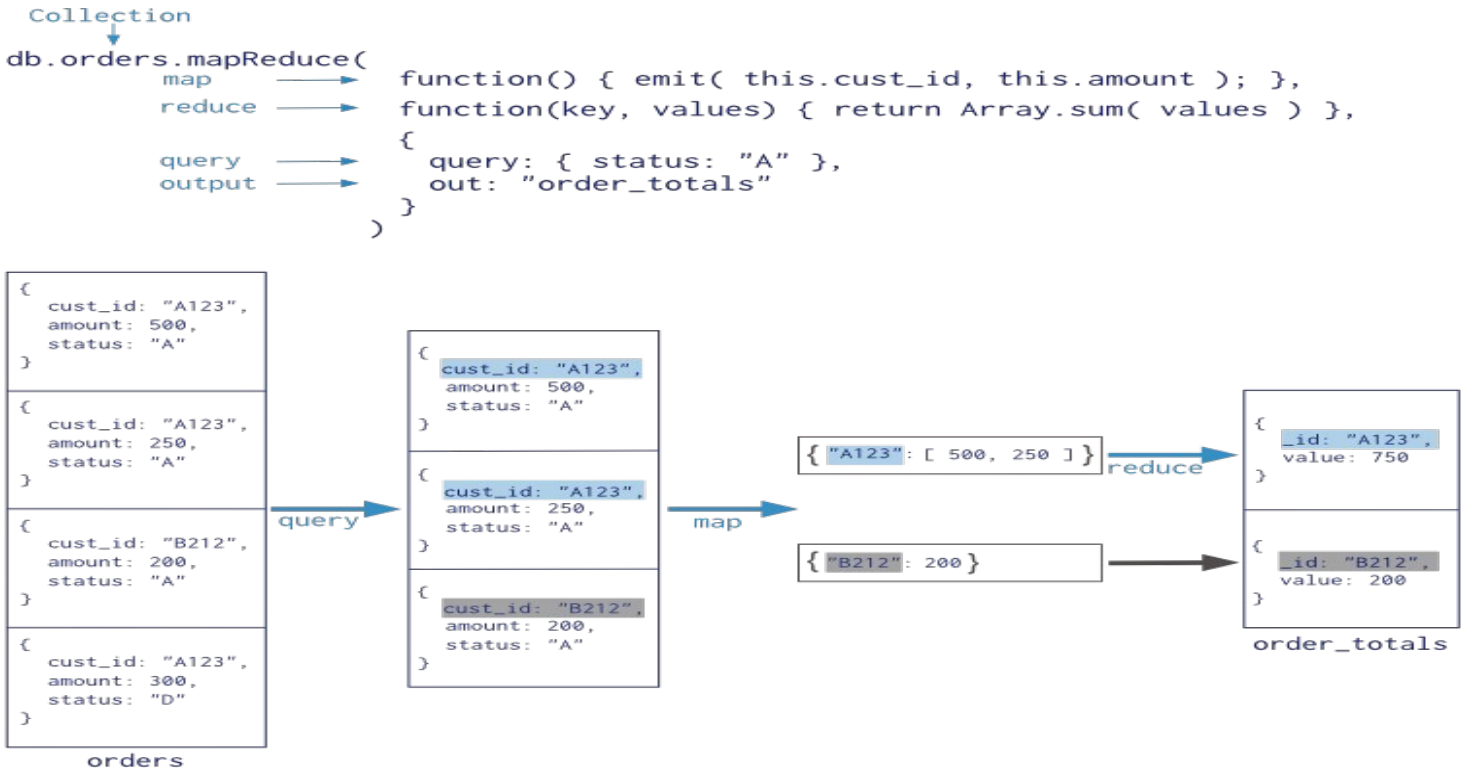
  Deprecato da MongoDB 5.0

  https://www.mongodb.com/docs/manual/core/map-reduce/

Aggregazioni (pipelines)



Aggregazioni (Map-Reduce)



SQL to Aggregation Mapping Chart

SQL	Aggregation
WHERE	**$match**
GROUP BY	**$group**
HAVING	**$match**
SELECT	**$project**
ORDER BY	**$sort**
LIMIT	**$limit**
SUM()	**$sum**
COUNT()	**$sum $sortByCount**
join	**$lookup**

Aggregazioni (Pipelines)

- stage
  - { $group: { _id: <expression>, <field1>: { <accumulator1> : <expression1> }, ... } }
- Accumulator Operators
  - { $sum: [<argument1>, <argument2> ...] }
  - { $avg: <expression> }
  - { $max: <expression> }
  - { $min: <expression> }
  - { $first: <expression> }
  - { $last: <expression> }
  - ...

{ $sum: <expression> }

sales collection:

{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-01-01T08:00:00Z") }

{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-02-03T09:00:00Z") }

{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 5, "date" : ISODate("2014-02-03T09:05:00Z") }

{ "_id" : 4, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-02-15T08:00:00Z") }

{ "_id" : 5, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-02-15T09:05:00Z") }

```
db.sales.aggregate( [
{ $group: {_id: { day: { $dayOfYear: "$date"}, year: { $year: "$date" } },
     totalAmount: { $sum: { $multiply: [ "$price", "$quantity" ] } },
     count: { $sum: 1 }} }
] )
```

{ "_id" : { "day" : 46, "year" : 2014 }, "totalAmount" : 150, "count" : 2 }

{ "_id" : { "day" : 34, "year" : 2014 }, "totalAmount" : 45, "count" : 2 }

{ "_id" : { "day" : 1, "year" : 2014 }, "totalAmount" : 20, "count" : 1 }

{ $add: [<expression1>, <expression2>, ...] }

sales collection:
{ "_id" : 1, "item" : "abc", "price" : 10, "fee" : 2, date: ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "fee" : 1, date: ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "fee" : 0, date: ISODate("2014-03-15T09:00:00Z") }

```
db.sales.aggregate([{ $project: { item: 1, total: { $add: ["$price", "$fee"] }}}])
```

{ "_id" : 1, "item" : "abc", "total" : 12 }
{ "_id" : 2, "item" : "jkl", "total" : 21 }
{ "_id" : 3, "item" : "xyz", "total" : 5 }

Esempio

```
{ cust_id: "abc123",
 ord_date: ISODate("2012-11-02T17:04:11.102Z"),
 status: 'A',
 price: 50,
 items: [ { sku: "xxx", qty: 25, price: 1 },
          { sku: "yyy", qty: 25, price: 1 } ]
}
```

```
db.orders.aggregate( [ {
 $group: { _id: null,
 count: { $sum: 1 } }
}])
```

SELECT COUNT(*) AS count
FROM orders

Esempio

```
{ cust_id: "abc123",
 ord_date: ISODate("2012-11-02T17:04:11.102Z"),
 status: 'A',
 price: 50,
 items: [ { sku: "xxx", qty: 25, price: 1 },
          { sku: "yyy", qty: 25, price: 1 } ]
}

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

SELECT SUM(price) AS total
FROM orders

Esempio

```
{ cust_id: "abc123",
 ord_date: ISODate("2012-11-02T17:04:11.102Z"),
 status: 'A',
 price: 50,
 items: [ { sku: "xxx", qty: 25, price: 1 },
          { sku: "yyy", qty: 25, price: 1 } ]
}

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

```
SELECT cust_id, SUM(price) AS total
FROM orders
GROUP BY cust_id
```

Esempio

```
{ cust_id: "abc123",
 ord_date: ISODate("2012-11-02T17:04:11.102Z"),
 status: 'A',
 price: 50,
 items: [ { sku: "xxx", qty: 25, price: 1 },
          { sku: "yyy", qty: 25, price: 1 } ]
}

db.orders.aggregate( [
{
 $group: {
 _id: "$cust_id",
 total: { $sum: "$price" }
 }
}, { $sort: { total: 1 } } ])
```

```
SELECT cust_id, SUM(price) AS total
FROM orders
GROUP BY cust_id
ORDER BY total
```

Esempio

```
{ cust_id: "abc123",
 ord_date: ISODate("2012-11-02T17:04:11.102Z"),
 status: 'A’,
 price: 50,
 items: [ { sku: "xxx", qty: 25, price: 1 },
          { sku: "yyy", qty: 25, price: 1 } ]
}

db.orders.aggregate( [ {
 $group: {
 _id: "$cust_id",
  count: { $sum: 1 }
 }
},
{ $match: { count: { $gt: 1 } }}
])
```

```
SELECT cust_id, count(*)
FROM orders
GROUP BY cust_id
HAVING count(*) > 1
```

Esempio

```
db.orders.aggregate( [ {
 $group: { _id: {
 cust_id: "$cust_id",
 ord_date: { $dateToString: {
  format: "%Y-%m-%d",
  date: "$ord_date" }}
 },
 total: { $sum: "$price" } }
 }
])
```

```
SELECT cust_id, ord_date,
SUM(price) AS total
FROM orders
GROUP BY cust_id, ord_date
```

Esempio

```
db.orders.aggregate( [ {
 $group: { _id: {
 cust_id: "$cust_id",
 ord_date: { $dateToString: {
  format: "%Y-%m-%d",
  date: "$ord_date" }}
 },
 total: { $sum: "$price" } }
},
{ $match: { total: { $gt: 250 } } }
])
```

```
SELECT cust_id, ord_date,
SUM(price) AS total
FROM orders
GROUP BY cust_id, ord_date
HAVING total > 250
```

Riepilogo

- Introduzione
- Tipologie di database noSQL
- MongoDB
  - CRUD
  - Relazioni fra documenti
  - Strutture ad albero
  - Aggregazioni
  - Esempi
- Approfondimento
  - https://www.mongodb.com/docs/