

Database Non Relazionali: Approfondimento su MongoDB

1. Introduzione ai Database Non Relazionali (NoSQL)

I database NoSQL (Not Only SQL) sono progettati per gestire dati non strutturati o semi-strutturati, a differenza dei database relazionali (SQL), che utilizzano tabelle, righe e colonne con schemi rigidi.

Caratteristiche principali dei DB NoSQL

1. Schema-less: Non richiedono uno schema predefinito, i documenti possono avere campi diversi.
2. Scalabilità orizzontale: Distribuiscono i dati su più server (sharding).
3. Modelli di dati flessibili:
 - Documenti (MongoDB, CouchDB)
 - Chiave-Valore (Redis, DynamoDB)
 - Colonne (Cassandra)
 - Grafi (Neo4j)
4. Performance elevate per operazioni di lettura/scrittura massicce.

Perché MongoDB è un database a documenti?

MongoDB memorizza i dati in documenti BSON (Binary JSON), che sono strutturati come oggetti JSON ma con tipi di dati aggiuntivi (es. `ObjectId`, `Date`, `Binary`).

 Approfondimento: [MongoDB Data Modeling](#)

2. Struttura dei Dati in MongoDB

Come sono organizzati i dati?


Concetto SQL	Concetto MongoDB
Database	Database
Tabella	Collezione
Riga	Documento BSON
Colonna	Campo
Chiave primaria	<code>_id</code> (ObjectId)

Esempio di Documento MongoDB

```
{
  "_id": ObjectId("507f191e810c19729de860ea"), // Chiave primaria univoca
  "titolo": "Inception",
  "anno": 2010,
  "regista": "Christopher Nolan",
  "cast": ["Leonardo DiCaprio", "Tom Hardy"],
  "rating": 8.8,
  "location": { // Sottodocumento annidato
    "city": "Los Angeles",
    "country": "USA"
  }
}
```

Tipi di dati supportati in BSON

Tipo	Esempio
String	<code>"nome": "Mario"</code>
Number	<code>"età": 30</code>
Boolean	<code>"attivo": true</code>
Array	<code>"tags": ["action", "sci-fi"]</code>
Object	<code>"indirizzo": { "città": "Roma" }</code>
Date	<code>"data": ISODate("2023-10-05")</code>
ObjectId	<code>"_id": ObjectId("...")</code>
Null	<code>"note": null</code>

 Documentazione: [BSON Data Types](#)

3. Relazioni tra Collezioni in MongoDB

A differenza dei database SQL, MongoDB non usa JOIN tradizionali, ma offre tre approcci per gestire le relazioni:

1. Embedding (Documenti Annidati)

- I dati correlati sono memorizzati all'interno dello stesso documento.
- Vantaggi: Query più veloci (tutti i dati in un unico accesso).
- Svantaggi: Può portare a duplicazione dei dati.

Esempio: Un blog post con i commenti incorporati.

```
{  
  
  "_id": ObjectId("..."),  
  
  "titolo": "Introduzione a MongoDB",  
  
  "autore": "Alice",  
  
  "commenti": [  
  
    { "testo": "Ottimo articolo!", "autore": "Bob" },  
  
    { "testo": "Molto utile", "autore": "Charlie" }  
  
  ]  
  
}
```

2. Referencing (Collegamenti tra Collezioni)

- Si usa un ID per riferirsi a un documento in un'altra collezione (simile alle foreign key in SQL).
- Vantaggi: Riduce la duplicazione.
- Svantaggi: Richiede query aggiuntive per recuperare i dati collegati.

Esempio: Un post collegato ai commenti tramite `ObjectId`.

```
// Collezione `posts`  
  
{  
  
  "_id": ObjectId("post123"),  
  
  "titolo": "Guida MongoDB",  
  
  "autore": "Alice"  
  
}
```

```
// Collezione `comments`  
  
{  
  
  "_id": ObjectId("comment1"),  
  
  "post_id": ObjectId("post123"), // Riferimento al post  
  
  "testo": "Interessante!"  
  
}
```

3. Uso di \$lookup (Join in Aggregation Pipeline)

- Simula un JOIN SQL unendo dati da due collezioni.
- Esempio: Trova tutti i commenti di un post.

```
db.posts.aggregate([  
  
  {  
  
    $lookup: {  
  
      from: "comments",           // Collezione da unire  
  
      localField: "_id",          // Campo nella collezione corrente  
  
      foreignField: "post_id",    // Campo nella collezione esterna  
  
      as: "commenti"             // Nome del campo risultante  
  
    }  
  
  }  
  
]);
```

 Approfondimento: [MongoDB Relationships](#)

4. La Chiave Primaria `_id` in MongoDB

- Cos'è `_id`?
 - È un campo obbligatorio in ogni documento.
 - Se non specificato, MongoDB genera un `ObjectId` univoco (es: `507f191e810c19729de860ea`).
 - Può essere sovrascritto con un valore personalizzato (es: una stringa, un numero).
- Struttura di un `ObjectId`

Un `ObjectId` è composto da:

- Timestamp (4 byte): Data di creazione.
- Macchina (3 byte): Identificativo del server.
- PID (2 byte): ID del processo MongoDB.
- Contatore (3 byte): Valore incrementale.

Esempio di query con `_id`:

```
db.users.find({ _id: ObjectId("507f191e810c19729de860ea") });
```

 Documentazione: [ObjectId Specification](#)

5. Comandi Fondamentali in MongoDB

Operazioni CRUD

Operazione	Comando MongoDB	Esempio
Insert	<code>insertOne()</code> / <code>insertMany()</code>	<code>db.movies.insertOne({ titolo: "The Matrix" })</code>
Find	<code>find()</code> / <code>findOne()</code>	<code>db.movies.find({ anno: 1999 })</code>
Update	<code>updateOne()</code> / <code>updateMany()</code>	<code>db.movies.updateOne({ titolo: "The Matrix" }, { \$set: { rating: 9.0 } })</code>
Delete	<code>deleteOne()</code> / <code>deleteMany()</code>	<code>db.movies.deleteOne({ titolo: "Inception" })</code>

Esempi Pratici

1. Inserire un documento:

```
db.users.insertOne({  
  
  nome: "Mario",  
  
  email: "mario@example.com",  
  
  isAdmin: false  
  
});
```

2. Trovare documenti con filtri:

```
db.movies.find({ anno: { $gte: 2000 }, rating: { $gt: 8 } });
```

3. Aggiornare un campo:

```
db.users.updateOne(  
  { email: "mario@example.com" },  
  { $set: { isAdmin: true } }  
);
```

4. Cancellare un documento:

```
db.movies.deleteOne({ titolo: "Inception" });
```

 Guida completa: [MongoDB CRUD Operations](#)