

# MongoDB

## Esempio di DB NoSQL: MongoDB

- Valore → **Numero**, intero o reale  
{"nome": "mario", "eta": 15, "punti": 13.45}
- Valore → **Stringa**, tra apici  
{"nome": "mario", "cognome": "rossi"}
- Valore → **Booleano**, true o false  
{"nome": "mario", "impiegato": true}
- Valore → **Array**, tra parentesi quadre  
{"nome": "mario", "cap": ["134", "042"]}
- Valore → **Oggetto**, tra parentesi graffe → il valore di un campo può essere un altro documento  
{"nome": "mario", "indirizzo": {"città": "bologna", "via": "po", "numero": 3} }

Utilizzo / Creazione DB → use provaDB

Creazione di una Collezione (vota) → db.createCollection ("cincoli")

Documento in MongoDB → oggetto JSON

## Esempio di DB NoSQL: MongoDB

- Nella stessa collezione, è possibile inserire documenti **eterogenei**, ossia con **strutture campo/valore** differenti.

COLLEZIONE ANAGRAFICA

DOC1	Marco	22		
DOC2	Massimo	Rossi		
DOC3	Maria	Bianchi	24	1/5/1990

sistemi schemaless  
↓  
architetture software che non richiedono uno schema dati predefinito e rigido

Inserimento di un documento in una collezione

db.NOMECOLLEZIONE.insertOne ({ nome: "Marco", cognome: "Verdi", eta: 22, domicilio: ["Roma", "Bari"] })  
↳ inserisce un solo documento all'interno della collezione

db.NOMECOLLEZIONE.insertMany ([{ nome: "Marco", cognome: "Verdi"}, { nome: "Giacomo", cognome: "Bla" }])  
↳ inserisce + documenti all'interno di una collezione

\_id → il campo \_id che rappresenta il campo id univoco che varia per documento nella collezione può essere esplicito (avendo lo inserisco io), oppure implicito (automatico del DBMS).

db.NOMECOLLEZIONE ( \_id: 1, nome: "Marco", cognome: "Rossi") esplicito

Rimozione di un documento da una collezione

db.NOMECOLLEZIONE.remove({}) → Svuota la collezione eliminando tutti gli elementi

db.NOMECOLLEZIONE.remove (SELETTORE) → Elimina dalla collezione tutti i documenti che hanno matching con il selettore

SELETTORE → si trova tra parentesi

↳ { nome: "Marco" }

↳ { \$or[ { nome: "Marco" }, { cognome: "Rossi" } ] }

↳ \$gt → greater than

↳ { prezzo: { \$gt: 50 } } → trova i prodotti con prezzo > 50

↳ \$lt → less than

↳ { matricola: { \$lt: 2000 } } → trova gli studenti con matricola < 2000

↳ \$gte → maggiore uguale

↳ \$lte → minore uguale

↳ \$ne → not equal

↳ Viene utilizzato per selezionare i documenti in cui il valore di un campo non è uguale al valore specificato

↳ db.user.remove ( { citta: { \$ne: "Roma" } } ) → rimuove tutti gli utenti che non vivono a Roma

↳ \$in → seleziona i documenti in cui il valore di un campo corrisponde a uno qualsiasi dei valori presenti in un array specificato.

↳ db.products.remove ( { categoria: { \$in: [ "Elettronica", "Libri" ] } } ) → rimuove i prodotti che appartengono alle categorie "Elettronica" o "Libri"

## UPDATE

update(SELETTORE, CAMP)

Di base MongoDB modifica il primo documento che trova (neuve ha il comportamento opposto)

↳ justOne: true

↳ Sostituisce il documento del selettore con la coppia selettore + campo

↳ db.utente.update ( { nome: "Marco" }, { eta: 24 } ) → Sostituisce il documento relativo a Marco, con la coppia nome, eta (scrive tutto)

update ( SELETTORE, { \$SET: CAMP } )

↳ modifica un campo specifico

↳ db.profilo.update ( { nome: "Marco" }, { \$set: { eta: 45 } } ) → Nel documento relativo all'impiegato Marco, aggiorna campo eta' mettendolo pari a 45

db.profilo.update ( { nome: "Marco" }, { \$set: { eta: 45 }, { multi: true } } ) → indica a MongoDB di aggiornare tutti i documenti che soddisfano il selettore

db.NOME COLLEZIONE.update({SELETTORE}, {\$push:CAMPI})

aggiunge → inserisce un nuovo campo

↳ db.profilo.update({name: "Marco"}, {\$push: {eta: 20}})

Se il campo esiste ed è un valore singolo (es. un numero) gallirà.

Se il campo esiste già ed è un array →

il costruttore aggiunge il nuovo valore in coda eta[40, 45]

## Ricerca

↳ db.NOME COLLEZIONE.find()

# Esempio di DB NoSQL: MongoDB

## Esempi di utilizzo del costrutto di find

### ○ db.anagrafica.find()

SELECT \* FROM anagrafica

### ○ db.anagrafica.find({"nome": "Mario", "eta": 30})

SELECT \* FROM anagrafica  
WHERE ((Nome='Mario') AND (ETA=30))

→ solo quelli che specificano condizioni specifiche

### ○ db.anagrafica.find({"nome": "Mario"}, {"eta": 1})

SELECT \_ID, ETA FROM anagrafica  
WHERE ((Nome='Mario'))

↓ nella visualizzazione  
ci dice se quel campo lo voglio  
includere questo 1  
mentre non lo voglio  
includere questo 0

BASI DI DATI  
PROF. MARCO DI FELICE – CORSO DI LAUREA IN INFORMATICA PER IL MANAGEMENT

# Esempio di DB NoSQL: MongoDB

## Esempi di utilizzo del costrutto di find

### ○ db.anagrafica.find({\$or:[{"nome": "Mario"}, {"eta": 56}]}, {"eta": 1})

SELECT \_ID, ETA  
FROM anagrafica  
WHERE ((Nome=Mario) OR (ETA=56))

### ○ db.anagrafica.find({"eta": {\$gte: 60}})

SELECT \*  
FROM ANAGRAFICA  
WHERE (ETA >= 60)

ORDINAMENTO db.NomeCollezione.find(...).sort(CAMPO/CAMPPI)

# Esempio di DB NoSQL: MongoDB

## Operatore di Ordinamento di una collezione

- o db.nomeCollezione.find(...).**sort(CAMPO/CAMPPI)**

1 = Ordinamento crescente, -1 = Ordinamento decrescente

```
db.anagrafica.find({"name": "Mario"}).sort({"eta":1})  
SELECT *  
FROM anagrafica  
WHERE (Name="Mario")  
ORDER BY ETA;
```

Conteggio di documenti → db.NomeCollezione.find(...).count()

# Esempio di DB NoSQL: MongoDB

## Operatore di Conteggio di documenti

- o db.nomeCollezione.find(...).**count()**

```
db.anagrafica.find({"nome": "Mario"}).count()  
SELECT COUNT(*)  
FROM anagrafica  
WHERE (Name="Mario")
```

FIND con esclusione di duplicati → DISTINCT

# Esempio di DB NoSQL: MongoDB

## Operatori di Filtro di documenti duplicati

- o db.nomeCollezione.distinct([CAMPO], SELETTORE)

```
db.anagrafica.distinct({"eta":1},{name:"Mario"})
```

```
SELECT DISTINCT(eta)
FROM anagrafica
WHERE (Name="Mario")
```

# Esempio di DB NoSQL: MongoDB

## Estensione Procedurale di MongoDB

→ quando voglio scorrere il risultato  
di una query sui dati

- ✧ Il file di script può contenere **costrutti iterativi e/o di selezione**:

```
while(condizione) { LISTACOMANDI}
if(condizione) { LISTACOMANDI }
else { LISTACOMANDI }
```

- ✧ I **cursori** vengono usati per scorrere il risultato di una query.

```
cursor = db.collection.find(...); → restituisce
while ( cursor.hasNext() ) { un insieme di righe
    printjson( cursor.next() );
}
```

# Esempio di DB NoSQL: MongoDB

```
conn = new Mongo();
db = conn.getDB("tennis2");
db.createCollection("soci");
cursor = db.collection.find({"name"="mario"});
while (cursor.hasNext()) {
    printjson(cursor.next());
}
cursor = db.collection.find({"name"="mario"});
if (cursor.hasNext()) {
    print("Trovato!");
}
```

myscript.js

MongoDB come tutti i sistemi (NoSQL) non mette a disposizione i costrutti di vincoli di integrità referenziale tra collezioni / tabelle

↳ Vuol dire che posso inserire dati esistenti in una collezione e in una no

\$lookup

→ operatore Join

→ Strumento che MongoDB usa per eseguire una join. Serve a unire documenti provenienti da due collezioni diverse all'interno di una pipeline di aggregazione

```
db.collezioneA.aggregate([
  {
    $lookup: {
      from: "collezioneB",           // 1. La collezione da "unire"
      localField: "id_comune",       // 2. Campo della collezioneA
      foreignField: "_id",          // 3. Campo della collezioneB
      as: "dati_uniti"             // 4. Nome del nuovo campo array creato
    }
  }
])
```

### Caratteristiche

↳ Sempre un Array → Il risultato del \$lookup viene salvato in un nuovo campo definito in as. Questo campo sarà sempre un array, anche se trova una sola corrispondenza (o nessuna)

↳ Left Join → Se non trova corrispondenza nella collezione "B", il documento della collezione "A" viene comunque restituito, ma con l'array dei risultati vuoto [].

## Mongo DB

Collezione → collection

↪ equivalente di una tabella nei database relazionali

db.collection.insertOne({ ... }) → inserisce un singolo documento

{name: "Federico", age: 22, gpa: 3.0})

db.collection.insertMany([ { ... }, { ... } ]) → inserisce più documenti contemporaneamente

db.collection.find() → restituisce tutti i documenti della collezione

db.students.insertOne({  
 ↓  
 name: "Andrea", age: 21, gpa: 2.8, gallTime: true, ... })

la stringa  
 Si può fare anche  
 con soltanto ''  
 Può contenere spazi

↓  
 integer

↓  
 col punto  
 (reale)

↓  
 booleano  
 true/false

, registerDate: new Date("20-04-2002"), graduationDate: null, courses: ["Biology", "ITA", "Eng"],  
 ↓  
 no value  
 ↗ array

, address: { street: "22 Savona", city: "Roma", zip: 12345 }

↑  
oggetto

↪ collezione  
dentro una  
collezione

## Sort Documents

db.students.find().sort({name: 1})

↓  
1 → ordine alfabetico (crescente)  
-1 → ordine inverso (decrescente)

db.students.find().limit(n)

↓  
numero di  
elementi

find({query}, {projection})

db.students.find().sort({gpa: -1}).limit(1) → Seleziona quello con il più alto gpa

db.collection.find({name: "Andrea", gpa: 4.0}) → ritorna il documento con quel nome e quel gpa

db.student.find({{}, {name: true}}) → mi fa vedere solo il nome di tutti gli studenti della collezione, in realtà mi fa vedere anche l'id

↪ Per non far vedere l'id

↪ {\_id: false, name: true})