Tecnologie Web: MongoDb

Prof. Raffaele Montella, PhD
raffaele.montella@uniparthenope.it

Sommario

- Introduzione
- Database NoSQL
- MongoDB
- PyMongo
- CRUD
- Conclusioni

Introduzione

- I servizi web sono per natura privi di connessione e di stato.

- Un servizio web esaurisce il proprio compito fra la request da parte del client e la relativa response.

- Se il servizio web deve interagire con dati persistenti, allora è necessario l’uso di un sistema di memorizzazione (storage).

Introduzione: Persistenza

- **File System:**
  I dati sono mantenuti sul file system adoperando tipicamente una interfaccia di tipo POSIX (Linux, MacOS, Windows, …)

- **Binary Large Objects (BLOBs)**:
  I dati sono identificati da oggetti binari di grandi dimensioni identificati univocamente.

- **Database:**
  I dati sono esposti mediante un modello logico che astrae

Introduzione: Definizione

“*In informatica, con base di dati, banca dati, database o anche solo DB, si indica una collezione di dati correlati (o archivio strutturato che conserva i dati omogenei per formato e contenuto) che, mediante un computer, vengono utilizzati per rappresentare una certa porzione del mondo reale.*”

Introduzione: Database

- Lo scopo di un database è quello di raccogliere un certo insieme di dati.
- Offrire i dati, o un sottoinsieme di essi, a utenti o altri componenti software.
- Le metodologie con cui i database rendono persistenti i dati sono molteplici.

Introduzione: DBMS

- Il modello logico di un database descrive come i dati sono rappresentati:
  - Relazionale
  - Gerarchico
  - Reticolare
  - Ad oggetti

- **Database Management System (DBMS):**
  Il componente software che si occupa della gestione e dell’astrazione del database si definisce come.

Introduzione: DBMS

- CRUD:
  - Create
  - Read
  - Update
  - Delete

- Funzionalità del DBMS:
  - CRUD
  - Query

Introduzione: Database Relazionali

- Structured Query Language (SQL).

- Database Relazionali:
  - Organizzati in tabelle, in cui ogni riga è detta record (o tupla) e ogni colonna “attributo”.

  - Il numero di colonne è costante in numero e tipologia.

  - Valori di attributi non disponibili sono gestiti come valori nulli.

Introduzione: Database non (solo) relazionali

- Organizzati in documenti, grafi, coppie chiave/valore, non sono escluse le tabelle.
- Sono utilizzati per dati non strutturati o che è non conveniente strutturare.
- Devono essere progettati tenendo conto delle query che si andranno ad effettuare.

Introduzione: Database non (solo) relazionali

- Gli attributi che caratterizzano i documenti possono essere variabili in numero e tipologia.
- Gestione dei valori nulli.
- I dati sono rappresentati da un documento JSON.

Introduzione: Database non (solo) relazionali

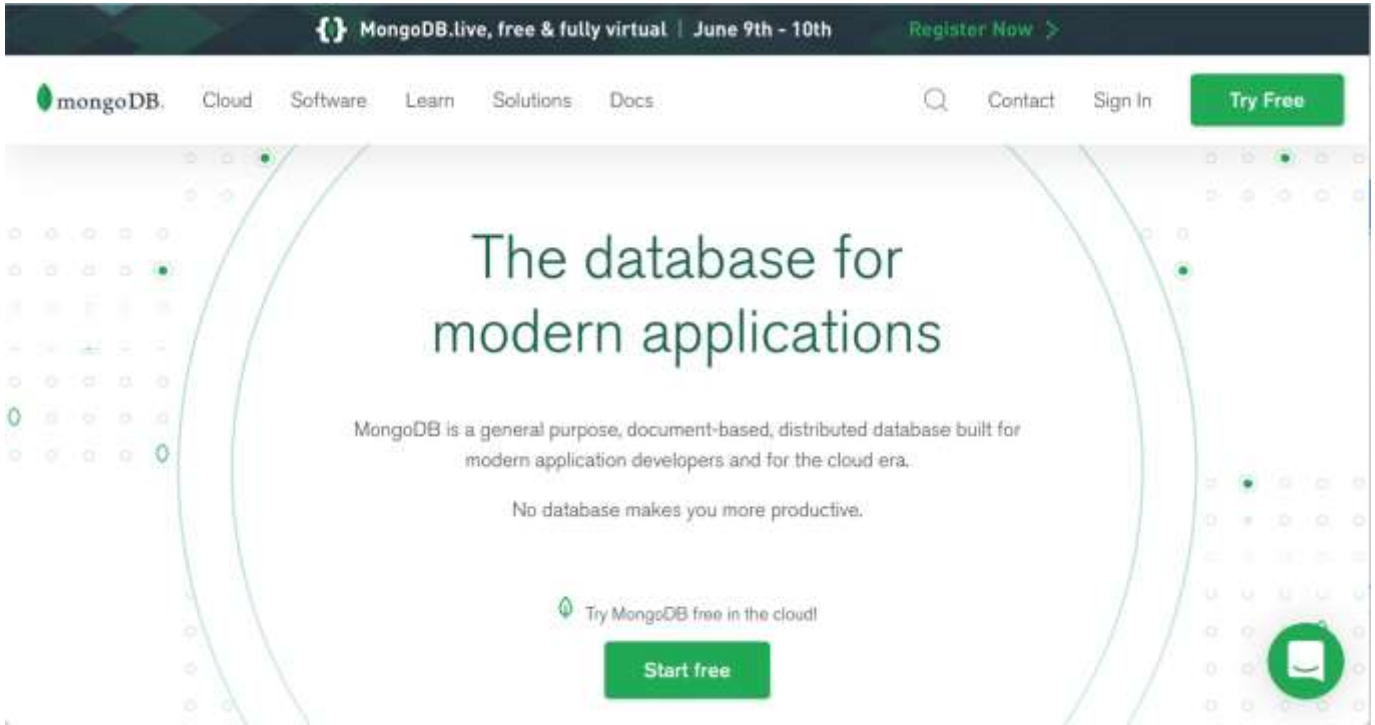
- Risorse computazionali:
  - I database noSQL sono più esosi rispetto ai SQL.
  - I database noSQL sono più scalabili rispetto ai SQL.
  - È spesso conveniente utilizzare soluzioni cloud.

Introduzione: MongoDB

- MongoDB (da "humongous", enorme) è un DBMS non relazionale, orientato ai documenti.
- Classificato come un database di tipo NoSQL.
- Documenti in stile JSON con schema dinamico (BSON).
- Rende l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce.

Introduzione: MongoDB

- Rilasciato sotto una combinazione della GNU Affero General Public License e dell'Apache License.



Introduzione: MongoDB

- Sviluppato inizialmente dalla società di software 10gen (ora diventata MongoDB Inc.) nell'ottobre 2007.
- Prodotto di Platform as a Service (PaaS)
- L'azienda si è spostata verso un modello di sviluppo open source nel 2009.
- MongoDB Inc. offre supporto commerciale cloud-

Introduzione

- MongoDB è gratuito.
  Le versioni rilasciate prima del 16 ottobre 2018 sono pubblicate sotto AGPL.

- Tutte le versioni rilasciate dopo il 16 ottobre 2018, incluse le correzioni per le versioni precedenti, sono pubblicate sotto Licenza pubblica lato server (SSPL) v1.

Introduzione: MongoDB

- Fanno uso di MongoDB:

Introduzione: MongoDB

- Il Python il linguaggio di riferimento.

- Funziona come un ORM, senza avere necessità di realizzare il mapping tra oggetti e modello relazionale.

- Gli strumenti per accedere ai dati sono:
  - Query
  - Indici
  - Collezioni.

Usare Docker per avere un’istanza di MongoDb

- Installare Docker sulla propria macchina (https://www.docker.com)
- Scaricare l’immagine più recente di MongoDb

  ```
  docker pull mongo
  ```
- Creare una directory persistente per mantenere i dati di MongoDB

  ```
  cd
  mkdir mongodata
  ```
- Definire la variabile di ambiente MONGODATA

  ```
  export MONGODATA="$PWD/mongodata"
  ```

Usare Docker per avere un’istanza di MongoDb

- Lanciare il container con il comando run montando la directory /data/db del container su $MONGODATA della macchina host:

  ```
  docker run -it -v $MONGODATA:/data/db -p 27017:27017 --name mongodb -d mongo
  ```

- Verificare che il container sia attivo

  ```
  docker ps
  ```

- Controllare i log di MongoDB

  ```
  docker logs mongodb
  ```

PyMongo

- È l'interfaccia Python verso MongoDb.
- Si suppone che mongodb**d** sia già disponibile nel sistema in uso.
- Preparare l'ambiente:

```
python3 -m venv venv
. venv/bin/activate
pip install pymongo
```

PyMongo: HelloWorld

```
# Importo il namespace
from pymongo import MongoClient

# Crea un'istanza del client
client = MongoClient("mongodb://localhost:27017/")

# Accede ad un database (se non presente, allora è creato
db = client['microblog')

# Accedo ad una collezione (se non presente, allora è creata
posts = db['posts']
```

PyMongo: HelloWorld

- L’accesso ad un server MongoDB è possibile specificando ip e porta o utilizzando l’URL.

- Se necessario, vanno specificate le credenziali per l’accesso.

- Sebbene database e collezioni possano essere create al volo da programma, l’effettiva a livello di DBMS sarà effettuata solo quando saranno aggiunti dati.

PyMongo: insert_one

```
# Creo un post come dizionario
post = { “author”: “John”,
        “text”:”My first microblog post!”,
        “tags”:[“mongodb”,”python”,”pymongo”],
        “date”: datetime.datetime.utcnow() }

# Inserisco il post nella collezione
result = posts.insert_one(post)

# Visualizza l’id univoco del documento
print(result.inserted_id)
```

PyMongo: insert_one

- Il metodo `insert_one()` restituisce un oggetto InsertOneResult, che ha una proprietà, **inserted_id**, che contiene l'id del documento inserito.

- Se non si specifica un campo `_id`, MongoDB ne aggiungerà uno di default e assegnerà un ID univoco per ciascun documento.

PyMongo: insert_many

```
# Creazione di un elenco di utenti come lista di dizionari
users = [
  { "_id": 1, "name": "John", "address": "Highway 37"},
  { "_id": 2, "name": "Peter", "address": "Lowstreet 27"},
  { "_id": 3, "name": "Amy", "address": "Apple st 652"},
  { "_id": 4, "name": "Hannah", "address": "Mountain 21"},
  { "_id": 5, "name": "Michael", "address": "Valley 345"},
  { "_id": 6, "name": "Sandy", "address": "Ocean blvd 2"} ]

# Aggiungo la lista
result = client.insert_many(users)
```

PyMongo: insert_many

- Per inserire più documenti in una raccolta in MongoDB, usiamo il metodo `insert_many()`.

- Il primo parametro del metodo `insert_many()` è un elenco contenente dizionari con i dati da inserire.

- Come nel caso del singolo documento, se non è specificato _id, ne è assegnato uno univoco di default.

PyMongo: find_one

- I metodi `find` e `find_one` sono usati per cercare dati in una collezione (query di selezione, SELECT).

- `find_one()` restituisce un singolo documento che corrisponde a una query (o “None” se non ci sono corrispondenze).

- È utile quando si sa che esiste una sola corrispondenza o che si è interessati solo alla prima.

PyMongo: find_one & find

```
# Recupero il primo utente
first_user = client.find_one()

# Visualizza il documento trovato
print(first_user)

# Recupero tutti gli utenti
items = client.find()

# Visualizza tutti i documenti trovati
for item in items:
        print(item)
```

PyMongo: find

- Per selezionare i dati da una collezione si usa il metodo `find()`.
- Il metodo `find()` restituisce un'istanza cursore, che consente di scorrere tutti i documenti nel risultato della ricerca.
- Il primo parametro del metodo `find()` è un oggetto query.
- Se nessun parametro è specificato, tutti gli attributi del

PyMongo: find_one & find

```
# Recupero un utente dato il suo id
a_user = client.find({“_id”:3})

# Visualizza il documento trovato
print(a_user)

# Recupero _id e name di tutti gli utenti
items = client.find({},{“_id”:1,”name”:1,”address”:0})

# Visualizza tutti i documenti trovati
for item in items:
        print(item)
```

PyMongo: find

- Il secondo parametro del metodo `find()` è un oggetto che descrive quali campi includere nel risultato.
- Questo parametro è facoltativo.
- Se omesso tutti i campi verranno inclusi nel risultato.
- Si ottiene un errore se si specificano entrambi i valori 0 e 1 nello stesso oggetto (tranne se uno dei campi è il campo _id).
- Se si specifica un campo con il valore 0, tutti gli altri campi

PyMongo: query avanzate

- Per costruire le query avanzate è possibile utilizzare i modificatori come valori nell'oggetto di tipo query.

- **Per esempio:**
  Per trovare i documenti in cui il campo "indirizzo" inizia con la lettera "S" o superiore (in ordine alfabetico), si utilizza il modificatore “maggiore di” : {"$ gt": "S"}

```
myquery = { "address": { "$gt": "S" } }
```

- https://docs.mongodb.com/manual/reference/operator/query/

PyMongo: query avanzate

- È possibile utilizzare le espressioni regolari come modificatori.

- Le espressioni regolari possono essere utilizzate solo per le query di tipo stringa.

PyMongo: query avanzate

- **Per esempio:**
  Per trovare solo i documenti in cui il campo "indirizzo" inizia con la lettera "S", si usa l'espressione regolare {"$ regex": "^ S"}:

```
myquery = { "address": { "$regex": "^S" } }
```

- https://www.w3schools.com/python/python_regex.asp

PyMongo: conteggio

- Se si vuole sapere solo quanti documenti corrispondono a una query, è possibile eseguire l'operazione `count_documents()` invece di una query completa.

- È possibile ottenere il conteggio di tutti i documenti in una collezione: `posts.count_documents ({})`

- È possibile contare i documenti che corrispondono a una query specifica: `posts.count_documents ({"author": "Mike"})`

PyMongo: ordinamento

- Si usa il metodo sort() per ordinare il risultato in ordine crescente o decrescente.

- Il metodo sort() accetta un parametro per "nomecampo" e un parametro per "direzione" (il crescente è la direzione predefinita).

- **Esempio:**
  Ordinare il risultato in ordine alfabetico crescente per nome:

```
items = users.find().sort(“name”)
```

PyMongo: ordinamento

- Si usa il valore -1 come secondo parametro per ordinare in modo decrescente.
  - `sort("nome", 1)  #crescente`
  - `sort("nome", -1)  #decrescente`

- **Esempio:**
  Ordinare il risultato in ordine alfabetico decrescente per nome:
  `items = users.find().sort(“name”,-1)`

PyMongo: delete_one & delete_many

```
# Definisco la query per selezionare un utente
query = { “address”:”Mountain 21” }

# Elimino l’utente
result = users.delete_one(query)

# Elimino più utenti alla volte
result = users.delete_many({ "address": {"$regex": "^S"} })

# Elimino tutti i documenti
Result = users.delete_many()
# Elimino la collezione
users.drop()
```

PyMongo: delete_one & delete_many

- Per cancellare un documento, usiamo il metodo `delete_one()`.
- Il primo parametro del metodo `delete_one()` è un oggetto query che definisce quale documento eliminare.
- Se la query trova più di un documento, viene cancellata solo la prima occorrenza.

PyMongo: delete_one & delete_many

- Per cancellare più di un documento , si usa il metodo `delete_many()`.
- Il primo parametro del metodo `delete_many()` è un oggetto query che definisce quali documenti eliminare.
- Per eliminare tutti i documenti in una raccolta, si passa un oggetto di tipo query vuoto al metodo `delete_many()`.
- Con il metodo `drop()` si cancella l'intera collezione.

PyMongo: update_one

- È possibile aggiornare un record o documento , come viene chiamato in MongoDB, usando il metodo `update_one()`.

- Il primo parametro del metodo `update_one()` è un oggetto di tipo query che definisce quale documento aggiornare.

- Se la query trova più di un record, viene aggiornata solo la prima occorrenza.

PyMongo: update_one

```
# Definisco la query per selezionare un utente
query = { “address”:”Valley 345” }

# Definisco i nuovi valori
result = users.delete_one(query)

# Elimino più utenti alla volte
values = { "$set": { "address": "Canyon 123" } }

# Effettuo la query di aggiornamento
users.update_one(query, values)
```

PyMongo: update_many

- Per aggiornare tutti i documenti che soddisfano i criteri della query, si utilizza il metodo `update_many()`.

- Esempio:
  Aggiornare tutti i documenti in cui l'indirizzo inizia con la lettera "S":

```
query = { "address": { "$regex": "^S" } }
values = { "$set": { "name": "Minnie" } }
result = users.update_many(query, values)
print(result.modified_count)
```

PyMongo: limit

- Per limitare il risultato si usa il metodo `limit()`.
- Il metodo `limit()` accetta un parametro , un numero che definisce quanti documenti restituire.
- Se si considera di avere una collezione di 14 "users".
- Esempio:
  Limitare il risultato per restituire solo 5 documenti.
  ```
  result = users.find().limit(5)
  ```

PyMongo: indicizzazione

- L'aggiunta di indici può aiutare ad accelerare la creazione di determinate query.

- Può aggiungere funzionalità per archiviare i documenti e potervi accedere più facilmente.

- **Esempio:**
  Creare un indice univoco su una chiave che non permette di aggiungere altri documenti perché il valore, per quella chiave,esiste già nell'indice.

PyMongo: indicizzazione

```
# Creazione dell'indice
result = db.users.create_index(
    [('name', pymongo.ASCENDING)],
    unique=True)

# Restituisce gli indici attivi
sorted(list(db.users.index_information()))
[u'_id', u'name']
```

PyMongo: indicizzazione

```
# Aggiungere utenti
users = [
  { "name": "Linda", "address": "Lake View Dr. 21"},
  { "name": "Peter", "address": "Floridiana Green 3A"},
  { "name": "Angelica", "address": "Breakout Rd 90"}]

# Aggiungo la lista
result = client.insert_many(users)
```

- **Traceback (most recent call last):**
  DuplicateKeyError: E11000 duplicate key error index: mydatabase.users.$name dup key: { : “Peter” }

Conclusioni

- I Database NoSQL sono semplici da usare.
- Si integrano facilmente nelle tecnologie web server side.
- A livello di prototipo sono semplici da installare e gestire.
- Sono adatti a dati non strutturati, non strutturabili o non convenienti da strutturare.

Conclusioni

- NON sono sempre la migliore delle alternative.
- In produzione sono costosi da gestire e manutenere.
- È consigliabile un approccio cloud.