NoSQL Database for Web Applications

Advanced Computer Programming

Prof. Luigi De Simone

Summary

- **Argomenti**
  - Database e DBMS
  - Database Relazionali vs Database Non-Relazionali
  - MongoDB
  - PyMongo

- **Riferimenti**
  - MongoDB & Python, Nial O'Higgins, O'Reilly
  - https://www.mongodb.com/docs/manual/
  - https://www.mongodb.com/docs/languages/python/pymongo-driver/current/
  - https://pymongo.readthedocs.io/en/stable/

Database e Web Server

- I **Database** sono comunemente utilizzati sulla parte **back-end delle web applications**, dai web server
- I database forniscono un modo per salvare i dati in **modo persistente** in memoria
- Memorizzare i dati in un database **riduce il carico sulla memoria centrale** del server e permette di poter **recuperare** i dati anche in caso di crash del server
- Molte richieste inviate ad un server potrebbero richiedere una query su un database
  - Un client potrebbe richiedere informazioni che sono memorizzate nel database, oppure sottomettere dei dati al server richiedendone la memorizzazione sul database.

Database e DBMS

- Un **Database** è una **collection** di dati organizzati in maniera tale da poter essere facilmente acceduti, gestiti ed aggiornati
- Un **Database Management System (DBMS)** è un software che permette la creazione, definizione e manipolazione di un database
- Permette all'utente di memorizzare, processare ed analizzare dati
- Il DBMS fornisce un interfaccia o tool che permette di effettuare operazioni come la creazione di un database, memorizzazione dei dati in esso, aggiornamento dei dati, creazione di tabelle nel database, etc.

Databases and DBMS

- Un DBMS fornisce anche **protezione** e **sicurezza** ai database, e permette di mantenere anche la **consistenza dei data** nel caso di utenti multipli.
- **Esempi di DBMS**
  - MySql
  - Oracle
  - SQL Server
  - IBM DB2
  - PostgreSQL
  - Amazon SimpleDB (cloud based)
  - Etc.

Features of DBMS

- I DBMS sono caratterizzati dalle seguenti features:
  - **Data stored into Tables**
  - **Reduced Redundancy via Normalization**
  - **Data Consistency**
  - **Support Multiple user and Concurrent Access**
  - **Query Language**
  - **Security**
  - **Support for transactions**

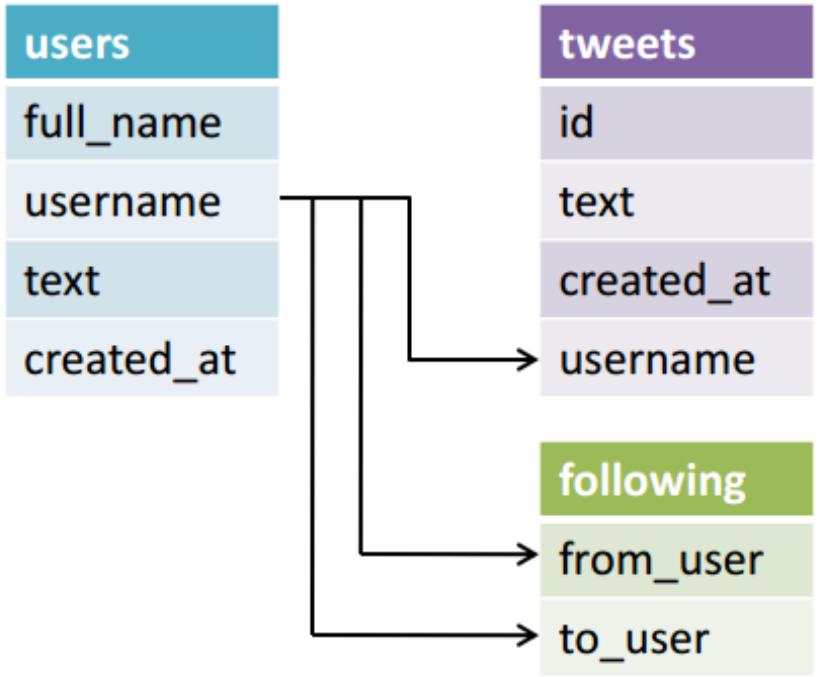
DB Relazionali e Non-Relazionali

- I Database sono divisi in due principali categorie:
  - **Relazionali** o **Sequence Databases using SQL languages**
  - **Non-relazionali** o **Non-sequence databases using NoSQL languages**
- Possono essere utilizzati da un'organizzazione singolarmente o in maniera combinata, in base alla natura del dato e della funzionalità necessaria

Database Relazionali

- Questo tipo di database usa uno **schema**, il quale è un template utilizzato per definire la struttura dei dati che saranno memorizzati nel database

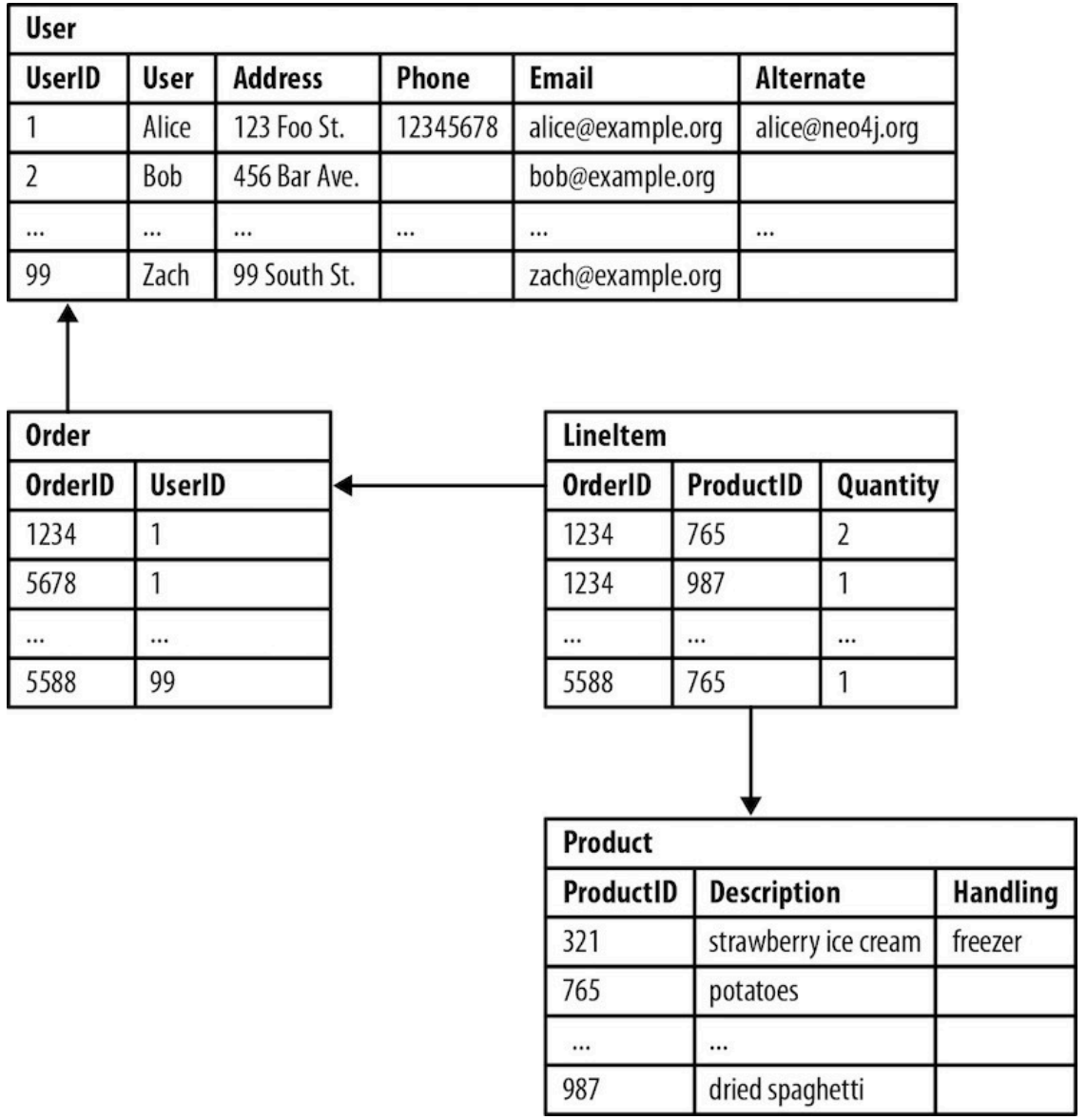
- L'esempio più significativo è **MySQL** database
- Possono esserci varie tabelle all'interno di un database di questo tipo
- Per ogni tabella ci sono delle **keys** ad esse associate



Database Relazionali

- Le chiavi permettono un **acceso rapido** ad una particolare riga o colonna
- La tabelle, che sono anche chiamate **Entità**, sono **tutte in relazione l'una con l'altra**
- La tabella con le informazioni di un cliente dovrebbe prevedere uno specifico ID per ogni cliente che può semplicemente indicare tutto ciò che c'è da sapere su quel cliente, come l'indirizzo, il nome e le informazioni di contatto.
- La tabella con la descrizione dei prodotti può assegnare un ID ad ogni prodotto. La tabella dove tutti gli ordini sono memorizzati potrebbe registrare solo questi IDs e la loro quantità.
- Ogni modifica in una di queste tabelle avrà effetti su tutte le altre in modo predicibile e sistematico.

Database Relazionali



User

UserID	User	Address	Phone	Email	Alternate
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
...	...	...	...	...	...
99	Zach	99 South St.		zach@example.org	

Order

OrderID	UserID
1234	1
5678	1
...	...
5588	99

LineItem

OrderID	ProductID	Quantity
1234	765	2
1234	987	1
...	...	...
5588	765	1

Product

ProductID	Description	Handling
321	strawberry ice cream	freezer
765	potatoes	
...	...	
987	dried spaghetti	

Vantaggi dei Database Relazionali

- I database relazionali **seguono uno schema ben definito**, pertanto ogni nuova entry deve essere conforme con questo template. **In questo modo i dati sono prevedibili e facilmente valutabili.**
- **ACID-compliance** è un must for tutti gli RDBMS (Relational DBMS) databases. Questo significa che devono garantire le proprietà di **Atomicity, Consistency, Isolation,** and **Durability**.
- Sono ben strutturati e lasciano **poco spazio all'occorrenza di errori**.

Svantaggi dei Database Relazionali

- La presenza di uno schema ben definito e di forti vincoli nei database relazionali rende quasi **impossibile il loro uso negli scenari big data**
- Gli utenti potrebbero voler scalare l'RDBMS su server multipli e più potenti che sono costosi e difficili da gestire. Per scalare un database relazionale, esso deve essere distribuito su server multipli, ma **gestire le tabelle su differenti server è difficoltoso**
- I vincoli dello Schema **ostacolano la migrazione** dei dati da e verso RDBMS diversi. Devono essere identici, altrimenti non funzionerà.

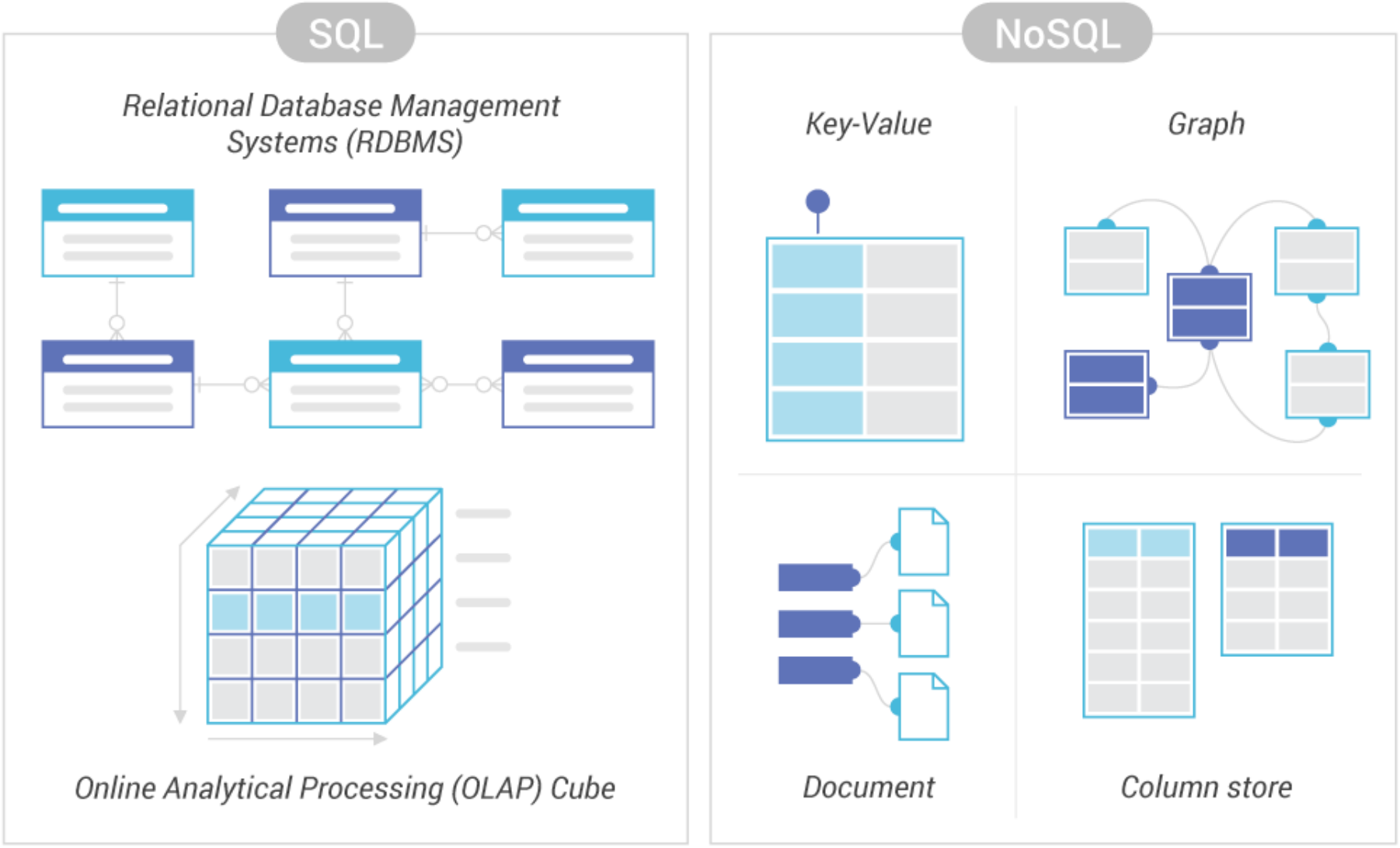
Database Non-Relazionali

- I database Non-Relazionali sono più **indulgenti nella loro struttura e forma** rispetto ai database relazionali
- Invece di tabelle con righe e colonne, essi utilizzano **collections di differenti categorie** come utenti e ordini
- Le collections sono caratterizzate da **documents**
- Possono esserci documents multipli in una collection, i quali possono non seguire un **particolare pattern o schema**
  - Un document può avere un nome, indirizzo, e prodotto in una collection
- Inoltre, collection differenti non devono necessariamente avere alcuna relazione

Tipologie di Database Non-Relazionali

- **Key-Value**: memorizza e fornisce dati sottoforma di **coppie key-value**
  - Alcuni esempi sono *Amazon DynamoDB* and *Redis*
- **Document Stores**: questi DB possono avere un ampio dizionario di tipi e valori, i quali possono essere innestati. I **dati sono memorizzati in JSON documents**
  - Alcuni dei NoSQL database più famosi sono *Couchbase* e *MongoDB*
- **Search Engines**: Si distinguono dai document stores in quanto permettono di accedere ai dati attraverso **semplici text-based searches**
  - Alcuni esempi sono *Solr*, *Splunk* ed *Elasticsearch*

SQL vs NOSQL


SQL
Relational Database Management Systems (RDBMS)
Online Analytical Processing (OLAP) Cube
NoSQL
Key-Value
Graph
Document
Column store

Pros e Cons dei database Non-Relazionali

- **Pros**:
  - La loro natura schema-free rende **facile la loro gestione e la memorizzazione di grandi volumi di dati**. Possono anche essere **facilmente scalati orizzontalmente**.
  - I dati non sono molto complessi e **possono essere distribuiti tra differenti nodi** per migliorarne l'accessibilità
- **Cons**:
  - Dato che non hanno una struttura o schema specifico per la memorizzazione dei dati, **non è possibile fare affidamento sulla presenza di un certo campo** perché potrebbe non sempre essere disponibile
  - Non avere relazioni rende **moto difficile l'aggiornamento dei dati** dato che bisognerà aggiornare ogni singolo dettaglio in maniera separata

Document stores DB: *MongoDB*

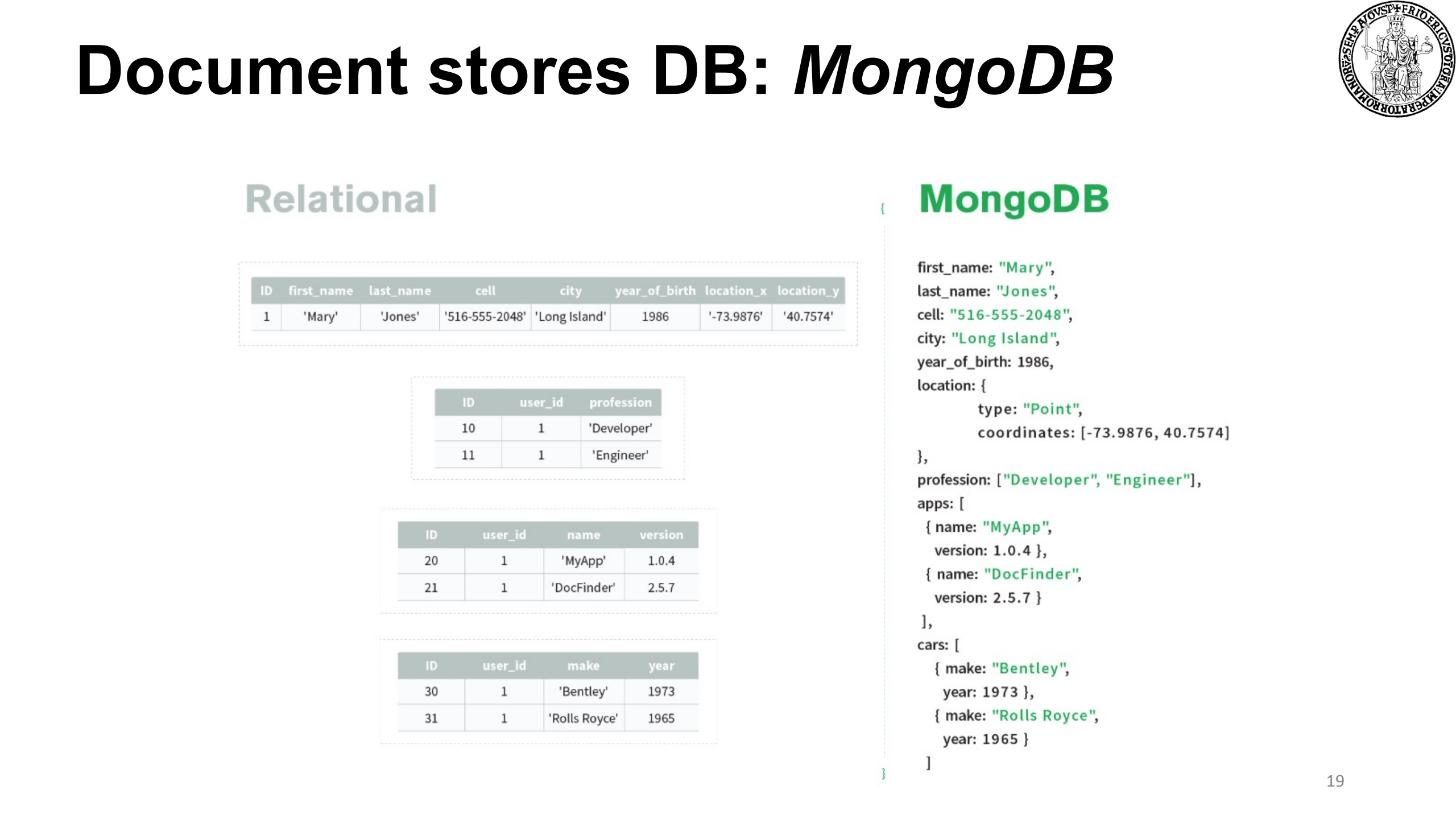
- Un **document-oriented database**, o document store, è un programma progettato per memorizzare, recuperare e gestire document-oriented information, anche noti come semi-structured data
- L'esempio principale è **MongoDB**
- E' costruito basandosi sui JSON-like documents
- Ogni database contiene delle **collections** le quali contengono **documents**.
- Ogni *document* può essere differente, con un **numero variabile di campi**. La dimensione ed il contenuto di ogni document può essere diverso l'uno dall'altro.
- I documents **non necessitano uno schema definito in anticipo**. I campi possono essere creati on-the-fly.

Document stores DB: *MongoDB*

Componenti chiave di MongoDB

- **_id:** Campo richiesto in ogni MongoDB document. Il campo *_id* rappresenta un valore unico nel MongoDB document. Tale campo è come una chiave primaria per il document.
- **Collection:** Raggruppamento di MongoDB documents. Una collection è l'equivalente di una tabella nei DBMS relazionali.
- **Cursor:** Puntatore al risultato di una query. I Client possono iterare sfruttando il cursore per recuperare i risultati.
- **Database:** Contenitore per le collection di un (nel caso relazionale è un contenitore di tabelle).
- **Document:** Un record in una collection MongoDB è chiamata document.
- **Field:** Una coppia name-value in un document. Un document ha zero o più campi.

Document stores DB: *MongoDB*



Install *MongoDB*

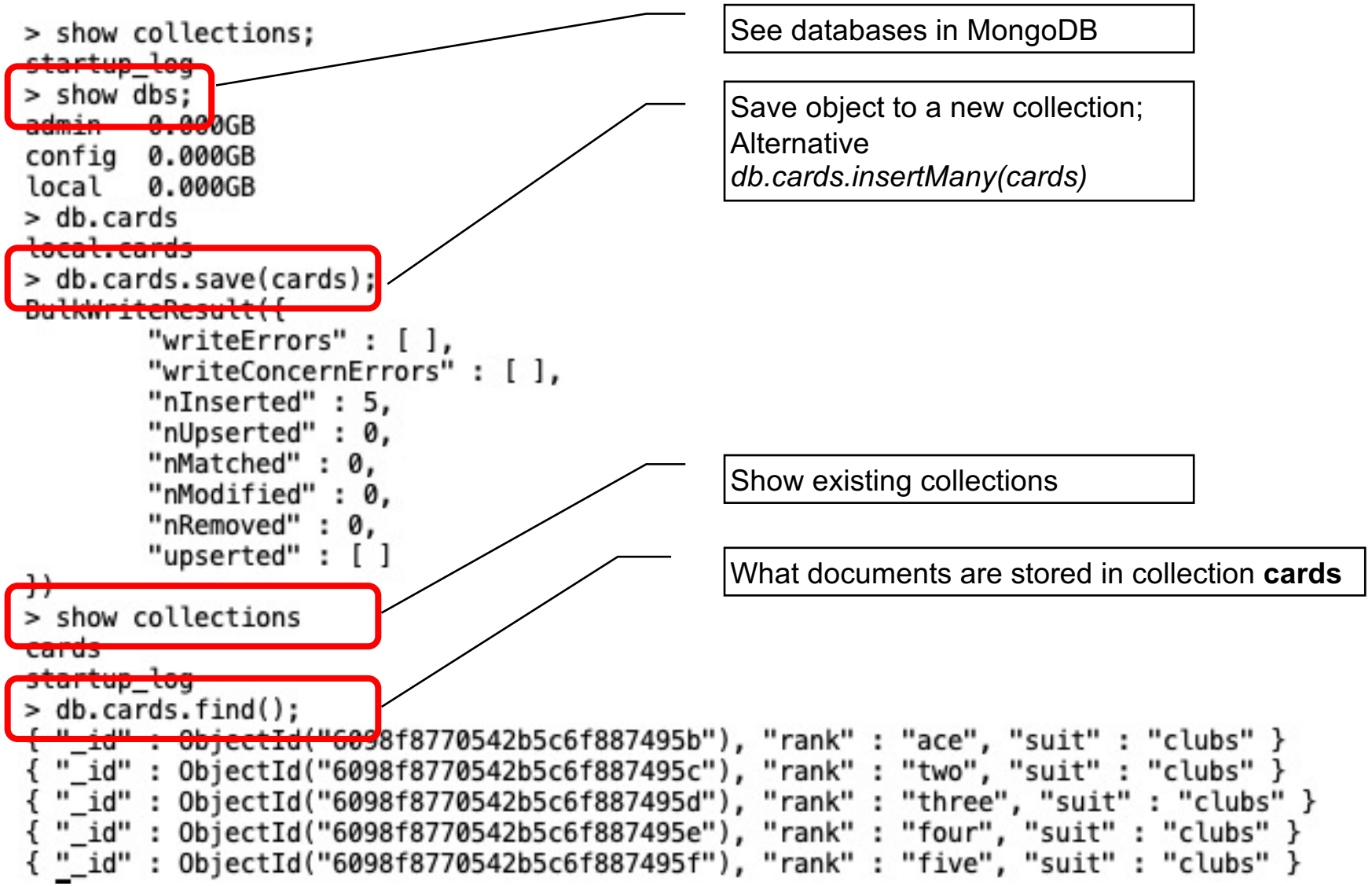
- Linux
  - https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/
- Mac OSX
  - ```
    brew tap mongodb/brew
    brew update
    brew install mongodb-community@6.0
    ```
- Start MongoDB server
  - `sudo systemctl start mongod` **`# Linux`**
  - `brew services start mongodb-community@6.0` **`# MAC OSX`**
- Start mongo shell
  - `mongosh (Mac OSX)`
  - `mongo (Linux)`

```
root@test:~# mongo
MongoDB shell version v4.4.5
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("0b4c7cfe-0b8c-4b7d-b31d-69d7b4ea1a88") }
MongoDB server version: 4.4.5
Welcome to the MongoDB shell.
```
```

MongoDB Collections

```
> cards = [{ "rank" : "ace", "suit" : "clubs" }];
[ { "rank" : "ace", "suit" : "clubs" } ]
> ["two", "three", "four", "five"].forEach( function (rank) { cards.push( { "rank":rank, "suit":"clubs" } ) });
> cards
[
        {
                "rank" : "ace",
                "suit" : "clubs"
        },
        {
                "rank" : "two",
                "suit" : "clubs"
        },
        {
                "rank" : "three",
                "suit" : "clubs"
        },
        {
                "rank" : "four",
                "suit" : "clubs"
        },
        {
                "rank" : "five",
                "suit" : "clubs"
        }
]
```

MongoDB Collections



DELETE METHODS in *MongoDB*

- **Remove some elements within collections**

```
# db.collectionName.remove({QUERY});
```

- **Remove all elements within collections**

```
# db.collectionName.remove({});
```

- **Remove a database**

```
# use databaseName;
# db.dropDatabase();
```

MongoDB in Python: PyMongo

- **PyMongo** è il driver ufficiale per l'utilizzo di MongoDB in applicativi Python
- Fornisce tutto il necessario per interagire con un'istanza di MongoDB

- **Installazione**:
  - `pip install pymongo`

- **Utilizzo:**
  - `import pymongo`

Connessione con MongoClient

- Il primo step è la **connessione** all'istanza di MongoDB in esecuzione
  - La classe `MongoClient` permette di effettuare tale connessione

```
from pymongo import MongoClient
client = MongoClient("localhost", 27017)
```

  - E' possibile anche utilizzare una **string connection**

```
client = MongoClient("mongodb://localhost:27017/")
```

- Ottenuto il client, è possibile **accedere ai database** gestiti dall'istanza di MongoDB

```
db = client['test-database']
```

Collection e formato dati

- Attraverso il riferimento al database è possibile accedere alle singole collection che compongono il DB

```
collection = dbname['test_collection']
```

- Sulle collection è possibile effettuare le varie **operazioni CRUD**
- I dati in MongoDB sono rappresentati e memorizzati sottoforma di **JSON-style documents**.
- In particolare, in PyMongo sono utilizzati i **dizionari** per rappresentare i **documents**.

```
post = {
  "author": "Mike",
  "text": "My first blog post!",
  "tags": ["mongodb", "python", "pymongo"]
}
```

Inserimento di documents

- **Inserimento di un solo documento** in una collection:

```
# post è un dictionary che rappresenta un document
collection.insert_one(post)
```

Da notare che se il dizionario/document non presenta un campo **_id**, tale campo sarà aggiunto e popolato automaticamente

- **Inserimento di documenti multipli** in una collection:

```
# posts è una lista di dictionary, cioè una lista di document
collection.insert_many(posts)
```

```
posts = [
    {
        "author": "Mike",
        "text": "Another post!",
        "tags": ["bulk", "insert"],
    },
    {
        "author": "Eliot",
        "title": "MongoDB is fun",
        "text": "and easy too!",
    },
]
```

Ricerca di documents

- **Ricerca di un solo documento** in una collection

```
result = collection.find_one({"author": "Eliot"})
```

  - Ritorna **il primo documento** che fa match con il filtro specificato (cioè il campo *author* è pari a *Eliot* nell'esempio), oppure **None** in caso contrario

- **Ricerca di documenti multipli** in una collection:

```
results = collection.find({"author": "Eliot"})
for post in results:
   print("author:", post['author'])
```

  - Ritorna un *cursor* per iterare sui risultati della query, cioè **tutti i documenti** che fanno match con il filtro specificato

Filtri

- **I filtri possono prevedere anche l'utilizzo di operatori per creare delle query più complesse**

```
res = collection.find({"items": {"$in" : ["item-A", "item-B"]}})
```

Operator	Meaning	Example	SQL Equivalent
$gt	Greater Than	"score":{"$gt":0}	>
$lt	Less Than	"score":{"$lt":0}	<
$gte	Greater Than or Equal	"score":{"$gte":0}	>=
$lte	Less Than or Equal	"score":{"$lte":0}	<=
$all	Array Must Contain All	"skills":{"$all":["mongodb","python"]}	N/A
$exists	Property Must Exist	"email":{"$exists":True}	N/A
$mod	Modulo X Equals Y	"seconds":{"$mod":[60,0]}	MOD()
$ne	Not Equals	"seconds":{"$ne":60}	!=
$in	In	"skills":{"$in":["c","c++"]}	IN
$nin	Not In	"skills":{"$nin":["php","ruby","perl"]}	NOT IN
$nor	Nor	"$nor":[{"language":"english"},{"country":"usa"}]	N/A
$or	Or	"$or":[{"language":"english"},{"country":"usa"}]	OR
$size	Array Must Be Of Size	"skills":{"$size":3}	N/A

Rimozione di documents

- **Rimozione di un solo documento** in una collection

```
collection.delete_one({"author": "Eliot"})
```

  - Cancella **il primo documento** che fa match con il filtro specificato

- **Rimozione di documenti multipli** in una collection:

```
collection.delete_many({"author": "Eliot"})
```

  - Cancella **uno o più documenti** che fanno match con il filtro specificato

Aggiornamento di documents

- **Aggiornamento di un solo documento** in una collection

```
collection.update_one({"author": "Eliot"}, {"$set": {"author": "eliot"}})
```

  - Aggiorna **il primo documento** che fa match con il filtro (primo parametro, cioè il campo *author* è pari a *Eliot* nell'esempio).
  - L'aggiornamento da realizzare è indicato dal modificatore (secondo parametro, cioè settate il campo *author* ad *eliot*, nell'esempio)

- **Aggiornamento di documenti multipli** in una collection:

```
collection.update_many({"author": "Eliot"}, {"$set": {"author": "eliot"}})
```

  - Aggiorna **uno o più documenti** che fanno match con il filtro specificato (primo parametro).
  - L'aggiornamento da realizzare è indicato dal modificatore (secondo parametro)

Modificatori

- **Esistono differenti modificatori che possono essere utilizzati con le operazioni di aggiornamento**

Modifier	Meaning	Example
$inc	Atomic Increment	“$inc”:{"score”:1}
$set	Set Property Value	“$set”:{"username”:"niall"}
$unset	Unset (delete) Property	“$unset”:{"username”:1}
$push	Atomic Array Append (atom)	“$push”:{"emails”:"foo@example.com"}
$pushAll	Atomic Array Append (list)	“$pushall”:{"emails”:["foo@example.com”,"foo2@example.com"]}
$addToSet	Atomic Append-If-Not-Present	“$addToSet”:{"emails”:"foo@example.com"}
$pop	Atomic Array Tail Remove	“$pop”:{"emails”:1}
$pull	Atomic Conditional Array Item Removal	“$pull”:{"emails”:"foo@example.com"}
$pullAll	Atomic Array Multi Item Removal	“$pullAll”:{"emails”:["foo@example.com”, “foo2@example.com"]}
$rename	Atomic Property Rename	“$rename”:{"emails”:"old_emails"}

Esempio

```
from pymongo import MongoClient

def get_database():

   # Provide the mongodb atlas url to connect python to mongodb using pymongo
   CONNECTION_STRING = "mongodb://127.0.0.1:27017"
   # Create a connection using MongoClient. You can import MongoClient or use pymongo.MongoClient
   client = MongoClient(CONNECTION_STRING)
   # Create the database
   return client['test’]

def add_items(collection):
   item_1 = {
       "_id" : "U1IT00001",
       "item_name" : "Blender",
       "max_discount" : "10%",
       "batch_number" : "RR450020FRG",
       "price" : 340,
       "category" : "kitchen appliance"
    }
    item_2 = {
        "_id" : "U1IT00002",
        "item_name" : "Egg",
        "category" : "food",
        "quantity" : 12,
        "price" : 36,
        "item_description" : "brown country eggs"
    }

    collection.insert_many([item_1, item_2])
```

Esempio

```
if __name__ == "__main__":
    # Get the database
    dbname = get_database()
    print(dbname)
    collection_name = dbname["user_1_items"]
    add_items(collection_name)
```