# INTERFACCIA COMPASS DI MONGODB:

È un Sistema NoSQL Document Based.

Ne Esistono anche Altri di TIPI

- Come Social Network, visto che Abbiamo diverse Connessioni Tra Utenti
- a Grafo (Neo4J)
- Chiave-Valore (AmzDynamoDB)
- .....

Struttura a DOCUMENTI Molto Simile alla programmazione ad Oggetti (come JAVA)

MongoDB è:

- Schemaless:
  - Struttura Evolve Nel Tempo
  - Dati NON Strutturati o Semi Strutturati
- Un Sistema DISTRIBUITO

Se ho dei CAMPI Null che NON Voglio inserire ⇒ Mi Occupano Anche Meno SPAZIO

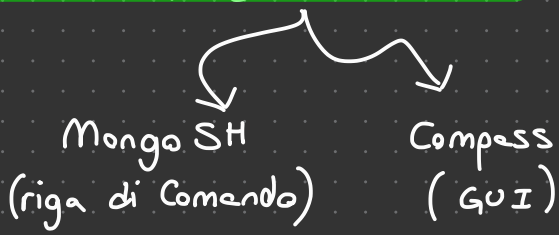
Devo Mantenere Però COERENZA Tra i dati Se Vengono Rappresentati Più Volte (DOPO UN AGGIORNAMENTO AD ESEMPIO)

Pensato per lavorare su CLUSTER, Ridondanza e Alta disponibilità (REPLICA)

Linguaggio D'Interrogazione è Molta Semplificata, ho operazioni Implementate Come:

- .findOne()
- .findMany()

Ha un Architettura CLIENT-SERVER

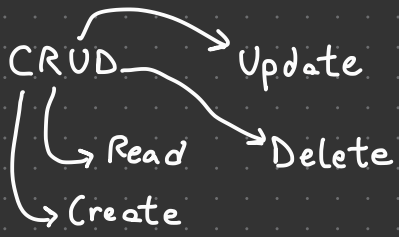



Ho MoHi TOOL per le PERFORMANCE:

- MongoStat
- Mongo Stop

Mongo DB

# OPERAZIONI SUPPORTATE:




db.Collection.insertOnes() ⇒ Ritorna un OGGETTO Json con:

- Acknowledge ⇒ Buon fine o Meno
- ObjectId ⇒ Assegnato Automaticam.

Collezioni Accettano di TUTTO, Vanno Sempre a Buon fine.

Con InsertMany posso Aggiungerci più DOCUMENTI, Separati da '{ }'

- find() ⇒ Restituisce TUTTO
- .find(□) ⇒ Dove □ è un Oggetto JSON.

Da Ricordare che le CONDIZIONI Sono in AND

Posso fare PROIEZIONE con .project(), di DEFAULT viene Restituito tutto

- Update One ( □, □ ) ⇒ Solo il primo documento Verrà Aggiornato


Ho dei METADATI

- # Documenti che Soddisfacevano il FILTRO
- # Documenti Modificati (0 o 1 in Questo Caso)

Posso Anche fare ad una Replace (equivalente dell'UPDATE di TUTTO) visto che viene Mantenuto l'OBJECT-ID

---

Mongo DB principalmente INCAPSULA LE INFORMAZIONI per Garantire le performance che però genera RIDONDANZA, e devo Preoccuparmi per Tenerle COERENTI

Posso avere un puntatore (RIFERIMENTO) ad un OBJECT_ID e Quindi diventa MOLTO MENO EFFICIENTE.

DENORMALIZZAZIONE ⇒ Soluzione MISTA, Sempre il Problema di Mantenere COERENZA

MongoDB in Realtà Use BSon, che ha un Supporto Sui

→ Binary

dati più IMPORTANTE:

- Date
- Binario
- Timestamp

DB deve Rimanere Sempre il Più leggero, Tramite le Aggregazioni posso Alleggerire Quello che Carico in Memoria (E NON CARICARE TUTTO e POI RAFFINARLO)

Operatore di JOIN ⇒ .LooKup che mi Permette SEMPRE e Solo un LEFT-OUTER JOIN (e Sarà Sempre un EQUI-JOIN, QUINDI SU UGUAGLIANZA DEI VALORI)

Annidamento migliora le prestazioni Solo se è un ANNIDAMENTO CORRETTO (che dipende... Bisogna Partire dalle QUERY) [e ci possono Ballare ORDINI DI GRANDEZZA]

Ma NON Sempre Si Conoscono le QUERY e ci Si affida al RELAZIONALE che va Sempre BENE.