CORSO I.F.T.S

“TECNICHE PER LA PROGETTAZIONE E LA GESTIONE DI DATABASE ”

Ing. Mariano Di Claudio

Lezione del 17/09/2014

Indice

1. **Aspetti fondamentali delle tecnologie Big Data**
   - Aspetti di *Data Management*
   - Aspetti *Architetturali*
   - Aspetti di *Accesso e Rappresentazione dei Dati*
   - Aspetti di *Ingestion, Mining e Data Analysis*

2. **Aspetti di Data Management**
   - NoSQL Database: Definizione
   - Paradigma ACID e Approccio BASE
   - NoSQL Database: Principali Tipologie
   - MongoDB
   - CouchBase

Panoramica sulle tecnologie Big Data

Considerando le tecnologie e le soluzioni adatte a lavorare con i **Big Data** bisogna prestare attenzione a **4 classi di aspetti fondamentali**:

- **Aspetti di *Data Management***
- **Aspetti *Architetturali***
- **Aspetti di *Accesso e Rappresentazione dei Dati***
- **Aspetti di *Ingestion, Mining e Data Analysis***

Aspetti fondamentali

1. **Aspetti di Data Management**
   Le principali soluzioni attualmente disponibili si stanno muovendo nella direzione di essere in grado di gestire adeguatamente **quantità di dati crescenti** (più o meno rapidamente) nel tempo.

2. **Aspetti Architetturali**
   Nell'elaborazione di un insieme molto grande di dati è importante **ottimizzare il carico di lavoro**, ad esempio adottando **un'architettura parallela (distribuita),** o fornendo una **allocazione dinamica delle risorse di calcolo**.

Aspetti fondamentali

3. **Aspetti di Accesso ai Dati e di Rappresentazione**
   E' opportuno avere dei tool che consentano una **visualizzazione scalabile dei risultati** ottenuti dall'analisi dei Big Data.

4. **Aspetti di Ingestion, Mining e Data Analysis**
   L'immagazzinamento, l'Analisi statistica, le Facet Query, la risposta alle query in modo rapido, con **dati aggiornati e pertinenti** alle ricerche effettuate, **coerenza e consistenza** sono importanti caratteristiche che un'architettura ideale in questo contesto deve sostenere o garantire.

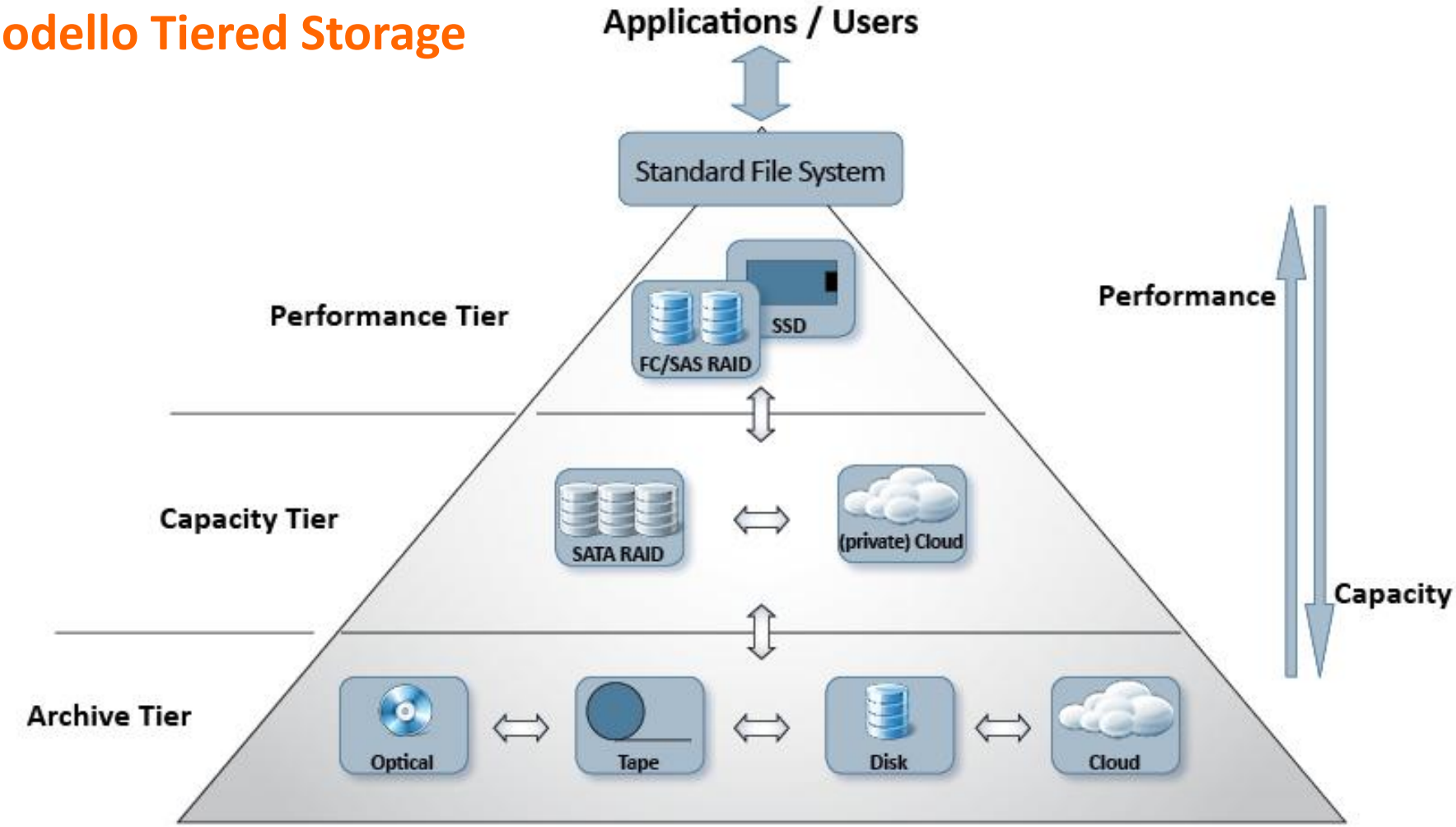
Aspetti di Data Management

- **Scalabilità**: Ogni sistema di archiviazione per i Big Data deve essere **scalabile**, si stanno diffondendo soluzioni di ***commodity hardware*** (HW facile da reperire e dal costo contenuto), con la possibilità di aumentare i dischi di storage.

- **Tiered Storage:** Modello di archiviazione gerarchico in cui diverse categorie di dati (livello di protezione richiesto, frequenza d'uso, requisiti prestazionali etc.) vengono assegnate a diversi sistemi di storage, in modo da ridurre i costi e ottimizzare i tempi di risposta per ottenere i dati richiesti.

Aspetti di Data Management

Modello Tiered Storage



Aspetti di Data Management

- **Elevata disponibilità:** requisito chiave in un'architettura Big Data. Il progetto deve essere distribuito (*con meccanismi di ridondanza dei dati*) ed eventualmente appoggiarsi ad una *soluzione cloud*.

- **Sostegno alle applicazioni di analisi e gestione dei contenuti**: l'analisi dei dati può richiedere giorni e può coinvolgere diverse macchine che lavorano in parallelo. Questi dati possono essere richiesti in parte ad altre applicazioni.

- **Integrazione con sistemi cloud esistenti pubblici e privati**: la questione cruciale è il trasferimento di tutti i dati. Il **supporto ad ambienti cloud esistenti**, faciliterebbe la migrazione dei dati evitando perdite e inconsistenze.

Aspetti di Data Management

- **Automazione del workflow:** pieno sostegno alla creazione, all'organizzazione e al trasferimento di flussi di lavoro.

- **Self Healing:** *L'architettura deve essere in grado di gestire i guasti dei componenti e di ripararli senza intervento di assistenza*. Tecniche che **reindirizzino automaticamente ad altre risorse**, il lavoro svolto dalla macchina guasta, che sarà automaticamente messa offline.

- **Sicurezza:** Molti dei sistemi che operano con i Big Data sono costruiti su modelli di web service, con pochi strumenti per contrastare le minacce web, mentre **è essenziale che i dati siano protetti da furti e da accessi non autorizzati**.

NoSQL Definizione

“Not Only SQL” – Identifica un sottoinsieme di strutture sw di memorizzazione, **progettate per ottimizzare e migliorare le performance delle principali operazioni su dataset di grandi dimensioni.**

Perché NoSQL?

- ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability), sono le proprietà logiche che devono soddisfare le transazioni nei DB tradizionali. *Questo paradigma però non è dotato di una buona scalabilità*.
- Le Applicazioni Web hanno esigenze diverse: **alta disponibilità**, **scalabilità** ed **elasticità**, cioè bassa latenza, schemi flessibili, distribuzione geografica (a costi contenuti).
- I DB di nuova generazione (NoSQL) sono maggiormente adatti a soddisfare questi bisogni essendo **non-relazionali, distribuiti, open source e scalabili orizzontalmente**.

Paradigma ACID

Atomicità

- Una transazione **è indivisibile nella sua esecuzione** e la sua **esecuzione deve essere o totale o nulla**, non sono ammesse esecuzioni parziali.

Consistenza

- Quando inizia una transazione il **database si trova in uno stato coerente** e quando la transazione termina il **database deve essere in un altro stato coerente**. Cioè non devono essere violati eventuali vincoli di integrità, quindi **non devono verificarsi contraddizioni (*inconsistenza*) tra i dati archiviati nel DB.**

Paradigma ACID

Isolamento

- Ogni transazione deve essere **eseguita in modo isolato e indipendente dalle altre transazioni**, l'eventuale fallimento di una transazione non deve interferire con le altre transazioni in esecuzione.

Durabilità (*o Persistenza*)

- Quando una transazione richiede un *commit work*, **le modifiche apportate non dovranno più essere perse**. **Per evitare** che nel lasso di tempo fra il momento in cui il DB si impegna a scrivere le modifiche e quello in cui le scrive effettivamente si verifichino **perdite di dati dovuti a malfunzionamenti, vengono tenuti dei registri di log dove sono annotate tutte le operazioni sul DB**.

NoSQL – Pro & Cons

PRO

- *Schema free*
- *Alta Disponibilità*
- *Scalabilità*
- *Supporto ad una facile replicazione*
- *API semplici*
- *Eventually Consistent / **BASE** (no ACID)*

CONTRO

- *Limitate Funzionalità di query*
- *Difficoltà di spostamento dei dati da un NoSQL DB ad un altro sistema (ma il 50% sono JSON-oriented);*
- *Assenza di modalità standard per accedere ad un archivio dati NoSQL*

Eventually Consistent

- **Modello di consistenza** utilizzato nei sistemi di calcolo distribuito.

- Il sistema di storage **garantisce che se un oggetto non subisce nuovi aggiornamenti**, alla fine (quando la finestra di inconsistenza si chiude) **tutti gli accessi restituiranno l'ultimo valore aggiornato.**

- Nasce con l'**obiettivo** di **favorire** le ***performance***, ***la scalabilità*** e la reattività nel servire un ***elevato numero di richieste*** (rischio minimo di letture di dati non aggiornati).

- Approccio **BASE** (**B**asically **A**vailable, **S**oft state, **E**ventual consistency).

- Conosciuta con il nome di **"Replicazione Ottimistica".**

Approccio BASE

- Le proprietà **BASE** sono state introdotte *Eric Brewer* (Teorema CAP).

- Queste proprietà rinunciano alla **consistenza** per garantire una maggiore **scalabilità** e **disponibilità** delle informazioni.

- **B**asically **A**vailable: Il sistema deve garantire la disponibilità delle informazioni.

- **S**oft State: Il sistema può cambiare lo stato nel tempo anche se non sono effettuate scritture o letture.

- **E**ventual consistency: Il sistema può diventare consistente nel tempo (anche senza scritture) grazie a dei sistemi di recupero ella consistenza.

Approccio BASE

Secondo questo approccio ***le inconsistenze sono transitorie****, cioè ogni nodo del DB del cluster, alla fine, otterrà le ultime modifiche consistenti ai dati.*

Il modello BASE, prevede **tre modalità di funzionamento**:

1. ***Casual Consistency*** – ad ogni modifica l'applicazione avvisa le altre sessioni che da quel momento vedranno il dato aggiornato.

2. ***Read your own writes –*** La sessione che esegue la modifica dei dati, vedrà questi cambiamenti immediatamente, mentre le altre sessioni li vedranno con un leggero ritardo.

Approccio BASE

3. Monotonic Consistency – Una sessione non potrà mai vedere dati con versioni precedenti a quella letta. I dati saranno sempre quelli letti oppure quelli modificati in seguito.

Uno dei **meccanismi per assicurare la *consistenza*** (in particolare la *Monotonic Consistency*) è basato sui ***vector clock***. L'obiettivo è quello di applicare le modifiche ai dati nella sequenza corretta.

- Ogni nodo del cluster mantiene un numero progressivo che identifica i cambiamenti (**change number**).
- Il **vector clock è una lista di change number**, provenienti da tutti i nodi.
- Ad ogni cambiamento il **vettore è inviato a tutti i nodi con l'aggiornamento**.
- Ogni nodo **analizza il vettore ricevuto e lo confronta con il precedente** per determinare se l'aggiornamento è in sequenza.
- Se non è nella giusta sequenza, l'aggiornamento non è applicato immediatamente, ma è conservato in attesa delle modifiche precedenti.

NoSQL DB - Dettagli

Caratteristiche chiave dei Database NoSQL sono sicuramente:

- **Ambiente Multi-Nodo**

Solitamente si crea un modello composto da un insieme di **più nodi distribuiti** (***cluster***), che possono essere aggiunti o rimossi.

- **Sharding dei dati**

Mediante opportuni algoritmi i **dati vengono suddivisi e distribuiti su più nodi**, recuperandoli quando necessario (es. *algoritmo di gossip*).

- **Replica delle informazioni**

I **dati** distribuiti sui vari nodi, spesso sono anche **copiati un certo numero di volte** per *garantire la disponibilità delle informazioni*.

NoSQL DB - Dettagli

I *NoSQL DB* sono oggetto di **continui studi** al fine di **migliorare**:

- **Performance**

Vengono studiati e implementati dei ***nuovi algoritmi*** al fine di **aumentare le prestazioni complessivi dei sistemi NoSQL**.

- **Scalabilità orizzontale**

E' fondamentale riuscire ad **aumentare/diminuire la dimensione di un cluster aggiungendo/rimuovendo nodi in modo "invisibile"**; cioè l'intero *sistema non deve fermarsi* o accorgersi di cosa sta accadendo.

- **Performance sulla singola macchina**

E' importante riuscire ad **aumentare anche le prestazioni di ciascuna macchina**, poiché spesso sono responsabili di ***ricercare informazioni nel cluster*** a nome di applicativi esterni.

Tipologie di Database NoSQL

- **Key-Value DB**
- Col-Family/Big Table DB
- Document DB
- Graph Database
- XML DB
- Object DB
- Multivalue DB
- ACID NoSQL

Key-Value Database

- Database **estremamente semplici**: c'è una *chiave* è c'è il *resto del dato* (il valore), il DB è una sorta di grande **hash table.**

- Se si vuole ricercare il valore di un determinato dato, la **ricerca** viene fatta **tramite chiave**.

- Soluzione che permette di ottenere una **buona velocità** ed **eccellente scalabilità orizzontale** (parallelizzazione dei nodi).

- *Diverse combinazioni di coppie chiave-valore danno origine a diversi database.*

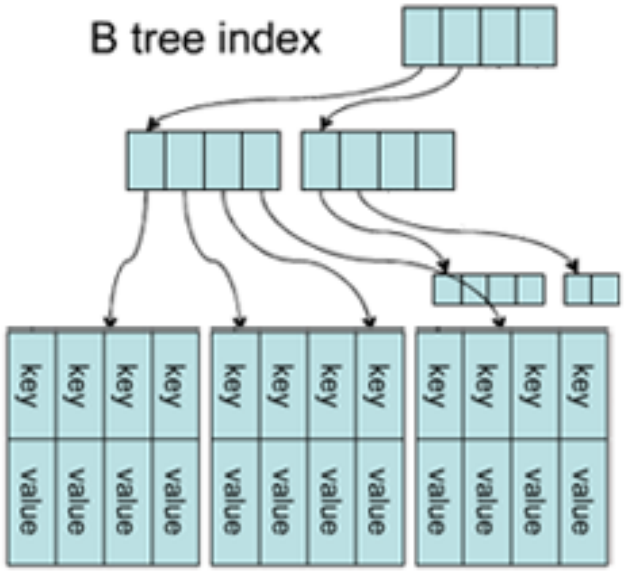
Key-Value Database

- E' di fondamentale importanza la **progettazione della chiave**, poiché lo storage si basa su **indirizzamento diretto**.

- Alcuni Key-Value DB permettono la definizione di **indici secondari** (es. B+tree) **per le chiavi**, ma si hanno perdite di performance e problemi nella gestione del clustering.

- Meglio utilizzare chiavi naturali o meccanismi distribuiti per la loro generazione come ***UUID*** **(Universally Unique Identifier) -** 32 caratteri esadecimali (8 - 4 - 4 - 4 - 12)

550e8400-e29b-41d4-a716-446655440000



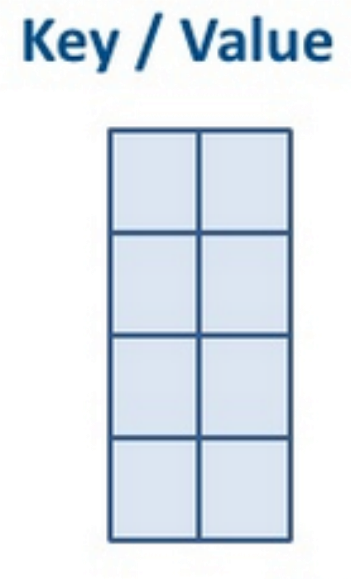
Key-Value Database

- **Modello di Dati:** raccolta di coppie Chiave-Valore.

- **Utilizzo:** Questo tipo di database NoSQL è indicato per la gestione liste con un numero elevato di elementi, di informazioni delle sessioni delle web app, degli "shopping cart" per gli acquisti online (con volumi di dati variabili nei diversi periodi dell'anno).

- **Basati** su *Amazon Dynamo Paper.*

- **Esempi:** Dynomite, Voldemort, Tokyo, **Riak**



Tipologie di Database NoSQL

- Key-Value DB
- **Col-Family/Big Table DB**
- Document DB
- Graph Database
- XML DB
- Object DB
- Multivalue DB
- ACID NoSQL

Column Family/Big Table Database

- In un **DBMS relazionale** una ipotetica tabella anagrafica avrebbe la seguente struttura:

ID	COGNOME	NOME	DATA DI NASCITA
1	Rossi	Mario	01/01/1960
2	Verdi	Giuseppe	01/01/1961
3	Bianchi	Antonio	01/01/1962

- Le informazioni sarebbero memorizzate riga per riga nel seguente modo:

  - 1, Rossi, Mario, 01/01/1960;
  - 2, Verdi, Giuseppe, 01/01/1961;
  - 3, Bianchi, Antonio, 01/01/1962;

Column Family/Big Table Database

- In un **DB Column Oriented** la stessa tabella anagrafica avrebbe la seguente struttura:

ID	COGNOME	NOME	DATA DI NASCITA
1	Rossi	Mario	01/01/1960
2	Verdi	Giuseppe	01/01/1961
3	Bianchi	Antonio	01/01/1962

- le stesse **informazioni** vengono **divise in colonne**:
  - 1, 2, 3;
  - Rossi, Verdi, Bianchi;
  - Mario, Giuseppe, Antonio;
  - 01/01/1960, 01/01/1961, 01/01/1962;

Column Family/Big Table Database

- In un **DB Column Oriented** la stessa tabella anagrafica avrebbe la seguente struttura:

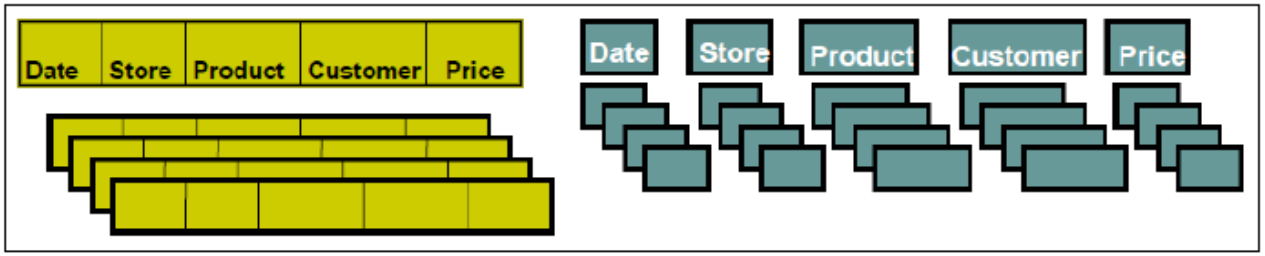
ID	COGNOME	NOME	DATA DI NASCITA
1	Rossi	Mario	01/01/1960
2	Verdi	Giuseppe	01/01/1961
3	Bianchi	Antonio	01/01/1962

- La tabella **memorizza una colonna alla volta** e alla fine procede con la memorizzazione della colonna successiva.

- Oltre alla colonna è memorizzata anche una **chiave surrogata** necessaria per **ricostruire il record**.

Column Family/Big Table Database

- I database chiave-valore possono evolvere in **column-family database** (composti da gruppi di colonne), in cui i dati sono memorizzati per colonne invece che per righe.



- I dati sono organizzati ancora come una hash table, ma si usano **due o più livelli di indicizzazione**.

- La **column-family** è un'informazione che deve essere presente nello schema. Raggruppa un insieme di colonne appartenenti alla stessa tipologia di informazioni, Es. ***"anagrafica"*** (Nome, Cognome, Età…)

Column Family/Big Table Database

- Le **colonne** che costituiscono una *column-family* **non devono essere definite a priori**.

- **Ogni record** potrà avere **informazioni differenti** all'interno di una *column-family.*

- Le **colonne vuote**, per le quali non si ha alcun valore, **non saranno presenti** all'interno di una *column-family*.

Esempio

Consideriamo un caso d'uso con due ***Unità Informative (UI),*** nello specifico due persone, per le quali abbiamo informazioni memorizzate all'interno di due ***Column Family (CF)***:

anagrafica	**indirizzo**

Column Family/Big Table Database

- **Archiviazione differente** rispetto ai RDBMS. Le colonne prive di valore non vengono riportate.

- **Notevole guadagno in termini di memoria**, significativo soprattutto su grandi numeri.

Column Family/Big Table Database

CF : *anagrafica*

UI : 1 *nome → Massimo | cognome → Carro | genere → Maschile | eta → 39*

UI : 2 *nome → Sara | cognome → Piccolo | genere → Femminile | **peso ->55***

- Aggiunta della chiave hobby e eliminazione del genere nell'UI 1

CF : *anagrafica*

UI : 1 *nome → Massimo | cognome → Carro | **hobby → lettura** | eta → 39*

UI : 2 *nome → Sara | cognome → Piccolo | genere → Femminile | peso ->55*

- L'aggiunta o l'eliminazione di una colonna non implica una ridefinizione dello schema (***alter table***).

- **Risparmio in termini di tempo e di memoria allocata** (soprattutto considerando che le informazioni modificate spesso coinvolgono un numero ristretto di righe).

Column Family/Big Table Database

- Il **punto di forza** dei database colonnari riguarda **l'immagazzinamento dei dati**. Offrono un elevato grado di compressione dei dati e non sprecano spazio per campi vuoti.

- Utili con **serie di dati storici** o dati provenienti da **più fonti** (sensori, dispositivi mobile e siti web), quindi spesso **differenti** tra loro, e con **elevata velocità**.

- Indicati per per sistemi di datawarehousing e read/only reporting, principalmente **OLAP** (On Line Analytical Processing). Sono infatti adatti all'uso di tecniche software per l'analisi interattiva e veloce di grandi quantità di dati.

Column Family/Big Table Database

- **Modello di Dati:** Big Table, Column Family.

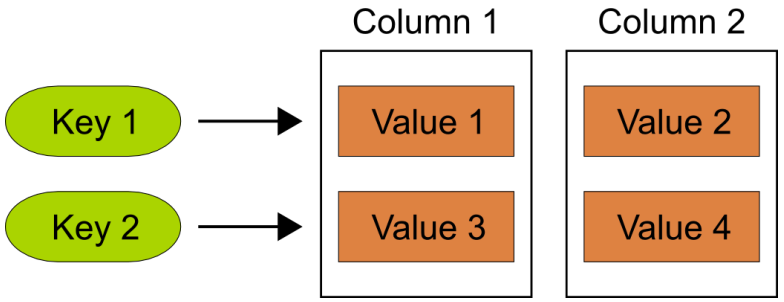
- **Pro:** Ogni riga può avere un **numero diverso di colonne**, evitando la presenza di dati ***NULL***.

- **Pro:** Una query recupera solo valori da determinate colonne e **non dall'intera riga.**

- **Pro:** Colonne composte da dati uniformi. Facilità di compressione e maggiore velocità di esecuzione e memorizzazione (possibilità di gestire PB).

- **Basati** su *Google BigTable Paper*

- **Esempi: Hbase**, Hypertable, Cassandra



Tipologie di Database NoSQL

- Key-Value DB
- Col-Family/Big Table DB
- **Document DB**
- Graph Database
- XML DB
- Object DB
- Multivalue DB
- ACID NoSQL

Document DB

- I database di tipo *document-oriented* sono **progettati per memorizzare, ricercare e gestire dati semistrutturati** (da non confondere con i sistemi di gestione documentale).

- **I "documenti",** sono **insiemi di coppie chiave/valore** spesso organizzati in formato JSON o XML (formati autodescrittivi). Sono quindi una sorta di *sofisticazione* dei **key/value DB**

- I **dati non sono memorizzati in tabelle** con campi uniformi e predefiniti, ma ogni documento è caratterizzato da specifiche caratteristiche (coppie chiave/valore strutturate in modo semplice).

```
{
  Nome:"Gianni",
  Indirizzo:"Via Roma 11",
  Hobby:"Fotografia"
}
```

Document DB

- **Minore rigidità**, non c'è uno **schema standard**. E' possibile **aggiungere campi o ampliarli** con dati (o parti di essi) multipli, inoltre non compaiono campi vuoti.

- La **struttura non pone dei vincoli** allo schema e garantisce una **buona flessibilità.**

```
{
  Nome:“Pino“,
  Indirizzo:"Via Galilei 34",
  Figli: [
     {Nome:”Luca”, Eta:5},
     {Nome:”Anna”, Eta:3},
  ]
}
```

- Questi DB sono perfettamente adatti alla programmazione Object Oriented (OO Programming). Gli oggetti infatti possono essere serializzati in un documento eliminando il problema dell'*impedance mismatching*.

Document DB

- Particolarmente **adatti** quando i **dati** sono **difficilmente rappresentabili** con un modello relazionale per via dell'elevata complessità (*dati complessi ed eterogenei*).

- Convenienti anche per rispondere alla ***Varietà dei Big Data***. Dati hanno una **struttura dinamica**, o **strutture molto differenti** tra loro, o presentano un grande numero di dati opzionali.

- Potrebbero essere utilizzati per implementare sistemi che formano uno strato su un DB relazionale o a oggetti.

- Utilizzati particolarmente nel caso di **flussi di dati medici** o dati provenienti dai **social network**.

Document DB

- Ogni documento è individuato mediante una **chiave univoca** (stringa, URI o path). Di solito c'è un **indice delle chiavi** per velocizzare le ricerche.

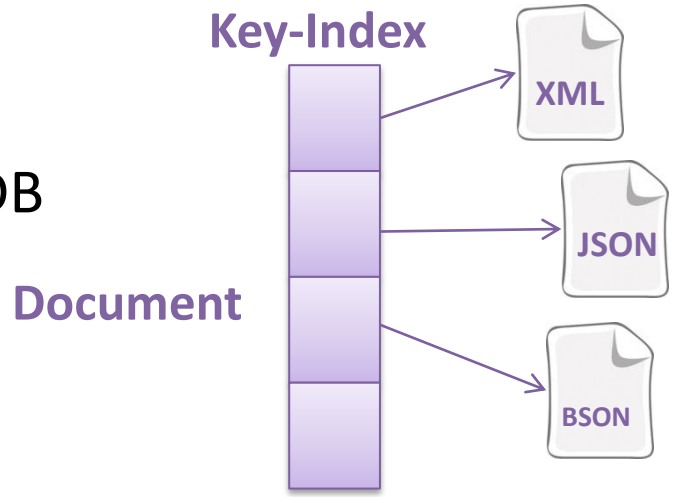
- Spesso sono presenti delle **API** o uno specifico **linguaggio di interrogazione (query)**, per realizzare il **recupero delle informazioni sulla base del contenuto**, esempio il valore di uno specifico campo.

Il **fine ultimo** di questa tipologia di database non è quello di garantire un'alta concorrenza, ma quello di **permettere la memorizzazione di grandi quantità di dati e di fornire un sistema di interrogazione sui dati performante**.

Document DB

Sono diventati i più popolari nella tipologia di NoSQL Database.

- **Modello di Dati:** raccolta di collezioni K-V (chiave-valore).
- **Utilizzano** le codifiche XML, YAML, JSON e BSON.
- **Ispirati** da Lotus Note (IBM).
- **Esempi:** CouchDB, **MongoDB**, SimpleDB



Tipologie di Database NoSQL

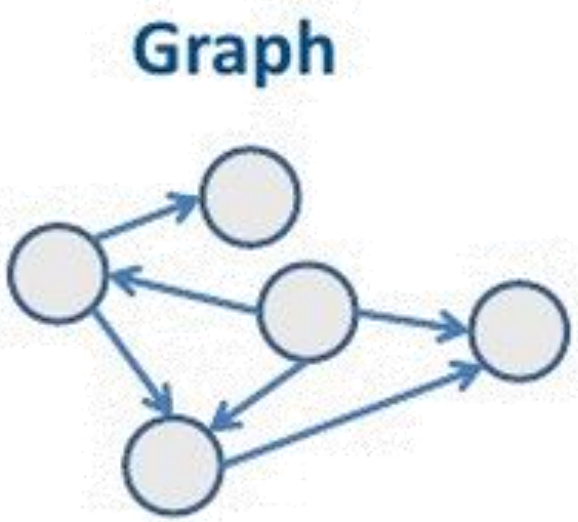
- Key-Value DB
- Col-Family/Big Table DB
- Document DB
- **Graph Database**
- XML DB
- Object DB
- Multivalue DB
- ACID NoSQL

Graph Database

- Nascono dall'esigenza di dover gestire **dati fortemente connessi** tra loro (specie non tabulari), in cui il concetto **di relazione tra i dati** ha un elevato potenziale informativo.

- **Occupano meno spazio** rispetto al volume di dati con cui sono fatti e **memorizzano molte informazioni sulla relazione tra i dati.**

- Questi DB si prestano molto bene alla rappresentazione di dati semistrutturati e interconnessi **come pagine Web, hiperlink e Social Network**.



Graph Database

- Le relazioni complesse presenti nei Social Network in realtà sarebbero rappresentabili anche con i **DB NoSQL** di *tipo key/value, column-oriented e document oriented*.

- **Non esiste il concetto di relazione**, ciò comporta *una maggiore complessità nelle interrogazioni e negli aggiornamenti*.

- Graph DB sono **ottimizzati alla rappresentazione e alla navigazione efficiente dei dati**:
  - Non hanno uno schema rigido.
  - Si adattano a cambiamenti nelle strutture dei dati.
  - Svolgono facilmente operazioni sui collegamenti, come ad esempio individuazione di “amici degli amici”.

Graph Database

- Uno dei **modelli** maggiormente **utilizzati per implementare i graph database** è quello basato sui ***"property graph"***.

- **I nodi**

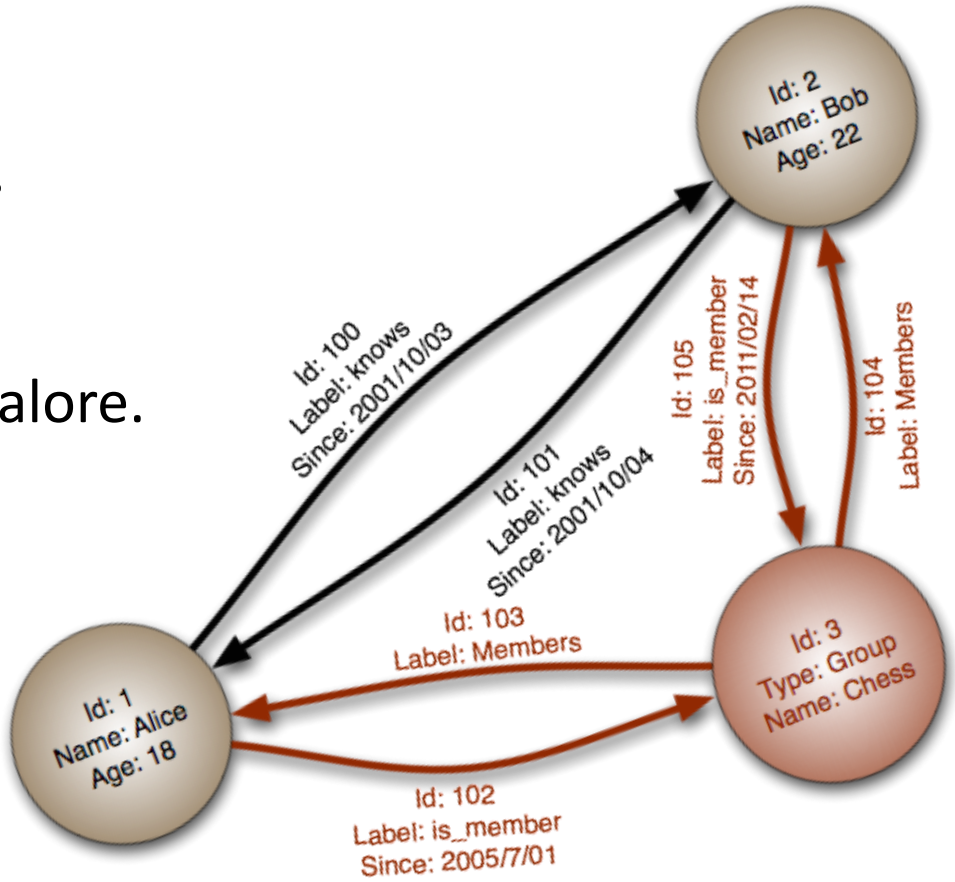
cioè dei contenitori di proprietà.

- **Le proprietà**

espresse sotto forma di chiave valore.

- **Le relazioni**

legano i nodi e a volte possono contenere proprietà.



Graph Database

- **Modello di Dati:** nodi-relazioni (chiave/valore).
- **Adatti** a gestire dati specifichi o che cambiano seguendo un certo schema.
- **Utilizzati** per la gestione di dati geospaziali, analisi di reti, bioinformatica, Social Network e motori di raccomandazioni.
- **Basati** sulla Teoria dei Grafi.
- **Esempi: Neo4J**, AllegroGraph, Objectivity.
- **(-)** Non è facile eseguire query su questo tipo di database.

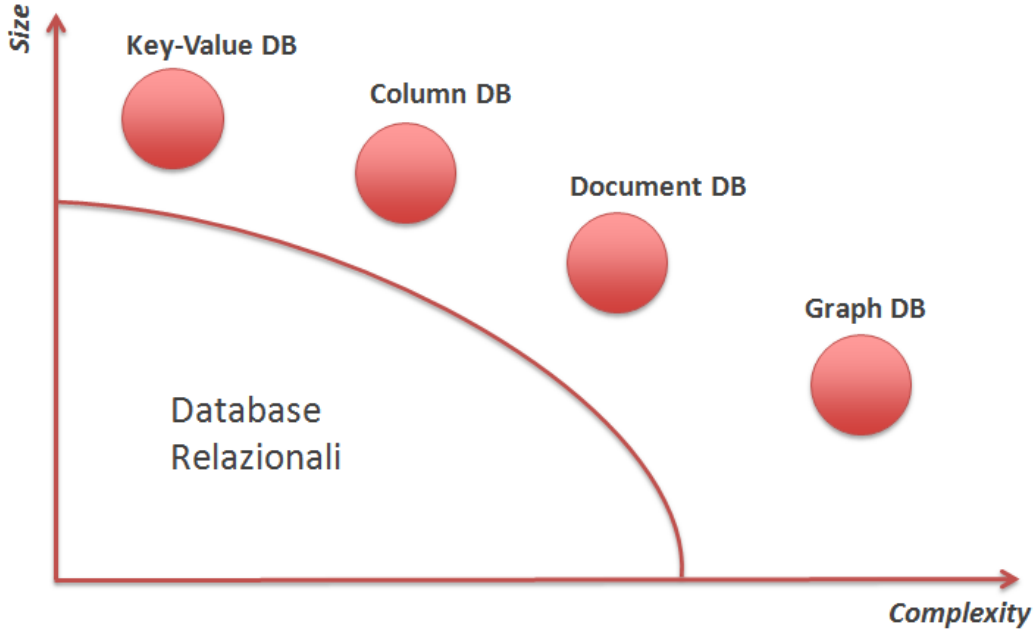
Comparazione

Tabella di riepilogo delle **caratteristiche principali** delle quattro classi fondamentali di **NoSQL Database.**

	Key-Value	Document-Oriented	Column-Family	Graph-Oriented
Schema	NO	NO	SI	NO
Correlazione dati	NO	NO	SI	SI
Caratteristica principale	Altissima Concorrenza	Query performanti, moli di dati altissime	Singola riga CF non separabile	Presenza delle relazioni tra i dati

Comparazione

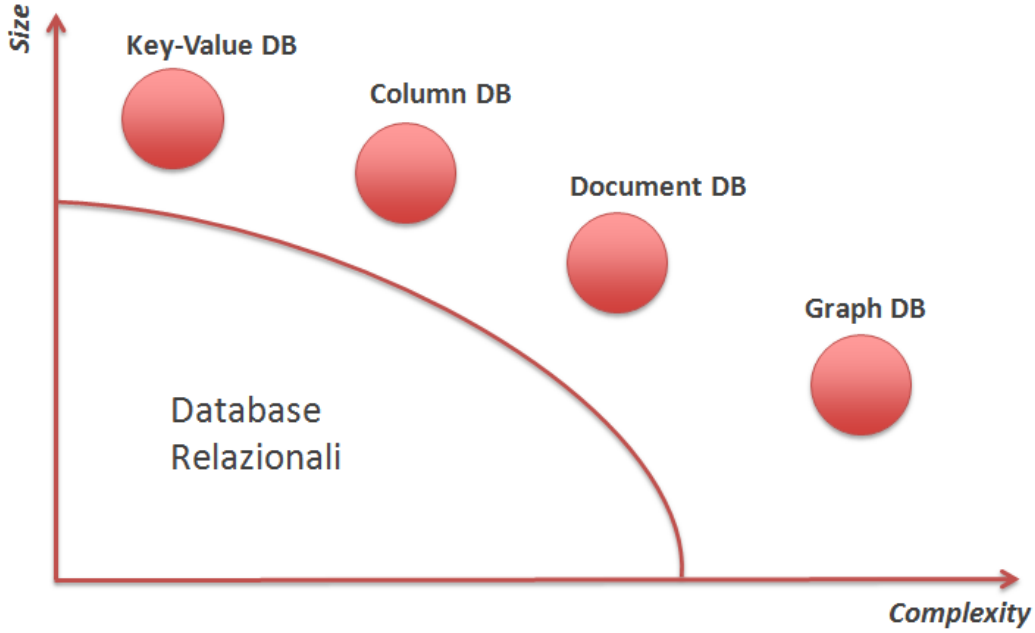
Valutazione delle soluzioni NoSQL DB sulla base dei parametri ***size*** e ***operational complexity***.



- I **Graph DB** hanno **una struttura di memorizzazione particolarmente complessa**, così come **complessi sono gli algoritmi** per navigarli (*algoritmi di routing*).
- **L'aumento della complessità** implica una **diminuzione della capacità di memorizzazione** (size).

Comparazione

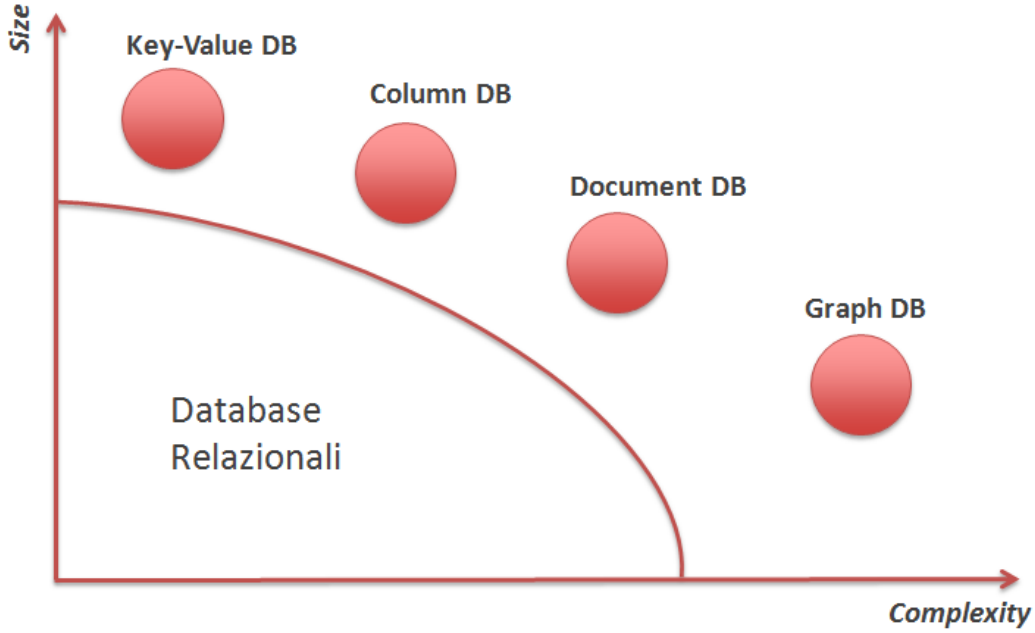
Valutazione delle soluzioni NoSQL DB sulla base dei parametri ***size*** e ***operational complexity***.



- La **famiglia dei DB a grafo** fornisce delle **routine di accesso ai dati molto performanti**, ma hanno il **limite di non poter trattare troppi dati**.

Comparazione

Valutazione delle soluzioni NoSQL DB sulla base dei parametri ***size*** e ***operational complexity***.



- La **famiglia dei DB chiave-valore** è invece adatta a **memorizzare grandi quantità di dati**, poiché utilizzano **l'*hash table*** come struttura dati di memorizzazione. Le **chiavi** sono quindi **scorrelate** tra loro e ciò garantisce una **buona scalabilità orizzontale**.

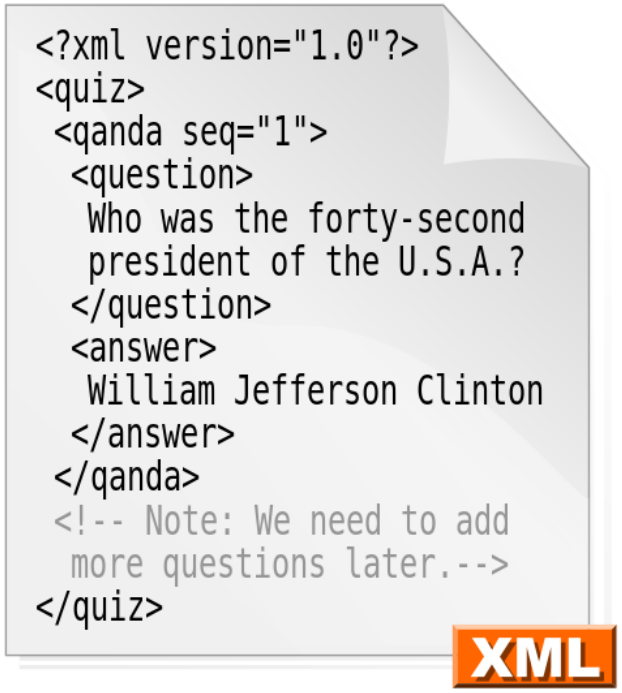
Tipologie di Database NoSQL

- Key-Value DB
- Col-Family/Big Table DB
- Document DB
- Graph Database
- **XML DB**
- Object DB
- Multivalue DB
- ACID NoSQL

XML Database

I **Database XML** sono nati come *ulteriore soluzione alla* ***gestione di dati difficilmente rappresentabili in formato tabellare***, infatti:

- Il **formato XML** è *molto utilizzato per lo scambio dei dati*; e la ***vista XML*** è quella *preferita da utenti e applicazioni*.
- I **documenti XML** si prestano bene a contenere **strutture dati gerarchiche**.
- Nasce quindi **l'esigenza di gestire dati XML all'interno dei database** tenendo ben presente la **struttura di origine**.



XML Database

Gli **XML-DB** *organizzano i dati in documenti XML* **ottimizzati** per contenere una **grande quantità di dati semi-strutturati.**

In questo contesto esistono tre possibili soluzioni:

- **Middleware**

Strati software che permettono lo **scambio di dati XML** *tra database(tradizionali) e applicazioni.*

- **Database con supporto XML** (o *XML-enabled*)
- **Database nativi XML**

Inizialmente lo sviluppo ***middleware*** ha permesso di far **dialogare i due mondi**. Poi sono stati *sviluppati diversi XML DBMS nativi*, e attualmente *diversi DMBS commerciali stanno investendo nel supporto XML*.

XML Database

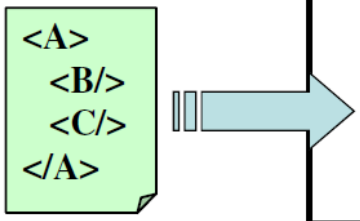
- **Database con supporto XML** (Oracle 9i con XDB) permettono di gestire documenti XML in 2 modalità:

1. **Document-Centric**
   - I **documenti XML sono i dati da inserire nel DB** (come campi).
   - Su questi documenti possono essere e**seguite semplici query tramite XPath** o espressioni regolari.

2. **Data-Centric**
   - La **struttura gerarchica del documento XML è parte fondamentale dei dati**, e *viene mappata su un insieme di tabelle relazionali.*
   - Dalle *relazioni tra le tabelle* è possibile **ricostruire la struttura originaria del documento XML.**

XML Database

Document Centric



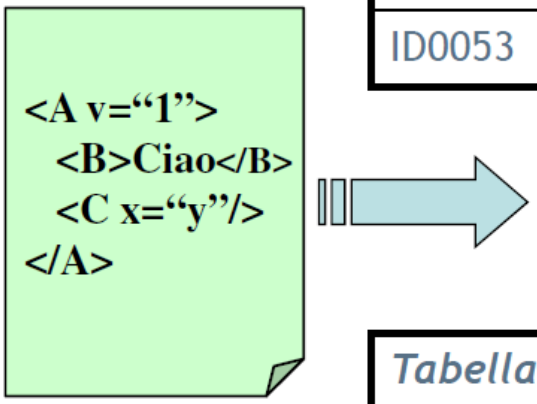
Tabella

Campo1	Campo2	Campo3	Campo4
<A> <B/> <C/> </A>	Xyxyxy	Xyxyxy	Xyxyxy
<D> Foobar </D>	Pippo	Pluto	Paperino
<E> <F> </F> </E>	Record3/1	Record3/2	Record3/3

Documento XML inserito nei campi del DB

XML Database

Data Centric



XML

Tabella B

B-ID	A-Ref	Text
ID0053	ID0011	Ciao

Tabella A

A-ID	v
ID0011	1

Tabella C

C-ID	A-Ref	x
ID0078	ID0011	y

Struttura del documento XML mappata sulle tabelle del DB

XML Database - Operazioni

Gli **XML-DB** *organizzano i dati in documenti XML* che possono essere:

- *Facilmente **interrogati tramite query XPath**.*
- *Trasformati mediante **XSLT in output tabellari**.*

XPATH

- **Linguaggio** che consente di **estrarre e manipolare nodi XML o valori**.
- Consente la **creazione di indici e query su dati di tipo XML**, gestiti nei DBMS relazionali.

XSLT (*eXtensible Stylesheet Language Transformation*)

- **Linguaggio** che permette la **trasformazione di documenti XML** in:
  - **Altri documenti XML**, ma con differente struttura.
  - In **HTML** o **documenti di testo** (*TXT, CSV…*)

XML Database - XPath

- **Xpath** è un linguaggio in cui compaiono delle ***path expression,*** *cioè una sequenza di passi (percorso) per raggiungere un nodo XML*. In queste espressioni gli elementi sono separati dal carattere '**/**'.
- **XPath** presenta **due possibili notazioni**:

Sintassi abbreviata
compatta consente la facile realizzazione di costrutti intuitivi.

```
/A/B/C
```

- Questa espressione seleziona gli elementi C che sono figli di elementi B a loro volta figli dell'elemento A (radice del file XML).

```
A//B/*[1]
```

- Espressione che seleziona il primo elemento figlio di B, senza tener conto di quanti elementi intercorrono tra B e A, //. Questa volta A non è nodo radice.

XML Database - XPath

Sintassi completa

Più complessa, contiene maggiori opzioni per specificare gli elementi.

- In questo caso in ogni punto del Xpath viene specificato esplicitamente l'***Axis*** seguito da **::**

Axis

- L'Axis è l'elemento che indica il **senso in cui deve essere percorso l'albero** del documento XML.
- Alcuni Axis sono: *child, attribute (@), descendent-or-self (//), parent(..), following* etc.

XML Database – Esempio XPath

`/listautenti/account//telefoni/*`

Restituisce la lista di tutti i nodi interni al nodo ***telefoni***, cioè ***fisso***, ***cellulare*** e ***fax***.

`/listautenti/account//indirizzo/..`

Restituisce tutti i nodi che contengono un nodo ***indirizzo***. L'uso dell'*Axis* **//** permette che vengano individuati anche *nodi di livelli differenti* purchè presenti all'interno di account.

`string(descendant::nome[1])`

Restituisce il valore della stringa del primo elemento ***nome*** che trova, in questo caso ***Gianni R***.

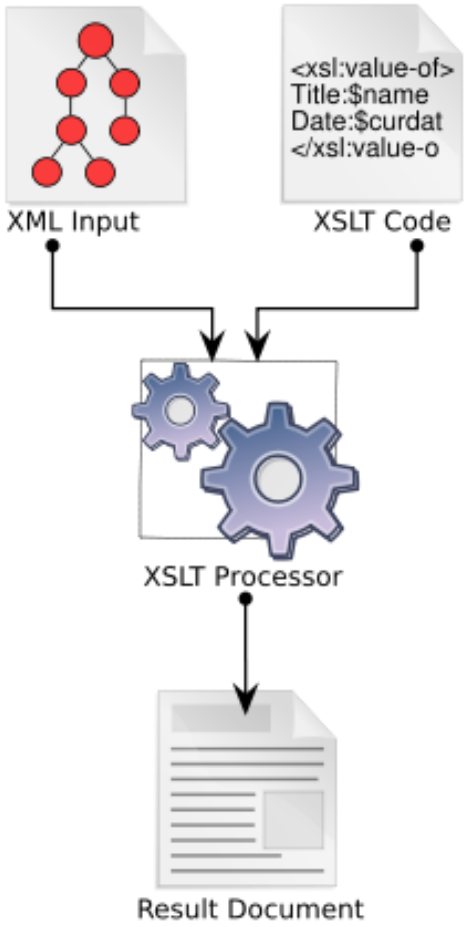
```
2  <?xml version="1.0" encoding="UTF-8"?>
3   <listautenti>
4    <account user="gianni">
5      <mail>gianni_red@mail.com</mail>
6      <nome>Gianni R.</nome>
7      <principale>
8        <indirizzo>
9          <via>Via vecchia 1</via>
10         <cap>98100</cap>
11         <citta>Siena</citta>
12       </indirizzo>
13       <telefoni>
14         <fisso>090123456</fisso>
15         <cellulare gestore="tim">3331214567</cellulare>
16       </telefoni>
17     </principale>
18     <altrirecapiti>
19       <ufficio>
20         <indirizzo>
21           <via>Via del lavoro, 2</via>
22           <cap>98100</cap>
23           <citta>Siena</citta>
24         </indirizzo>
25         <telefoni>
26           <fisso>09078901</fisso>
27           <fax>090345367</fax>
28         </telefoni>
29       </ufficio>
30     </altrirecapiti>
31   </account>
32  </listautenti>
```

XML Database – XSLT

XSLT (***eXtensible Stylesheet Language Transformations***) *è il linguaggio per trasformare l'XML* (diventato uno standard web con una direttiva W3C del 16 novembre 1999) *in un altro documento*.

- Per realizzare la trasformazione XSLT occorrono due file:

1. ***Il documento XML*** che deve essere trasformato.
2. **Il foglio di stile XSLT** che fornisce la *semantica della trasformazione*. Questo documento, infatti, *interpreta un XML* come una ***serie di nodi strutturati ad albero***.



XML Database – Esempio XSLT

Documento XML che deve essere trasformato.

```
<?xml version="1.0" ?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family-name>Smith</family-name>
  </person>
  <person username="MI1">
    <name>Morka</name>
    <family-name>Ismincius</family-name>
  </person>
</persons>
```

Lo si vuole trasformare in un ***nuovo documento XML*** con una ***struttura differente.***

XML Database – Esempio XSLT

Il **foglio di stile XSLT** definisce il **template** da seguire nel seguente modo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/persons">
    <root>
      <xsl:apply-templates select="person"/>
    </root>
  </xsl:template>

  <xsl:template match="person">
    <name username="{@username}">
      <xsl:value-of select="name" />
    </name>
  </xsl:template>

</xsl:stylesheet>
```

XML Database – Esempio XSLT

Il **nuovo documento XML** che si ottiene in **output** avrà la seguente struttura:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <name username="JS1">John</name>
  <name username="MI1">Morka</name>
</root>
```

Osservazione

Un **documento XML** può **essere associato a più fogli di stile XSLT**, ciascuno dei quali genererà *un output diverso*. Lo stesso procedimento vale anche al contrario: **uno stesso foglio di stile** può essere **applicato a più documenti XML**, allo scopo di *produrre documenti dal contenuto differente ma formato analogo*.

XML Database - Sintesi

- **Database XML Nativi** consentono la gestione di documenti XML come collezioni di dati indicizzati, che possono essere interrogati mediante XPath o Xquery.

- Possibilità di **integrazione con DB relazionali standard**, con schemi che mappano la struttura relazionale su documenti XML.

- **Esempi:** Tamino (SoftwareAG), BaseX, **eXsist** (Apache).

- Attualmente il progetto maggiormente sviluppato e stabile è **eXistdb.**

  Supporta le interrogazioni tramite XPath e XQuery 1.0
  Supporta gli aggiornamenti tramite il linguaggio Xupdate
  Accessibile da codice mediante l'interfaccia standard XML:DB

Tipologie di Database NoSQL

- Key-Value DB
- Col-Family/Big Table DB
- Document DB
- Graph Database
- XML DB
- **Object DB**
- Multivalue DB
- ACID NoSQL

Object Database (OODBMS)

- Nascono con **l'obiettivo di modellare dati complessi** (es. *programma cartografico per la gestione di mappe geografiche*) e **interconnessi in ambiti scientifico-tecnologici**.

- **Integrano** alle **basi di dati** i **concetti della programmazione ad oggetti**, superando i limiti del modello relazionale. *Ciò permette di realizzare applicazioni più efficienti.*

- **Uso di concetti del mondo OO**
  - Definizione di ***strutture complesse***.
  - ***Rappresentazione fedele e unitaria*** di oggetti della ***realtà*** (le caratteristiche non vengono suddivise su più tabelle).
  - Possibilità di definire ***relazioni/proprietà complesse*** come *ereditarietà, incapsulamento, overridding* etc.

Object Database (OODBMS)

- Le **strutture dati** definite possono essere **salvate direttamente nel database** senza subire alcuna modifica-adattamento.

- Il **modello dei dati è salvato in memoria di massa** e il DBMS assegna a **ciascun oggetto un identificatore univoco** (**Oid**), finchè l'oggetto sarà presente (ciclo di vita dell'oggetto).

- **L'Oid serve per recuperare un oggetto**, le sue *proprietà* e per la *gestione delle relazioni con altri oggetti*.

- I **DBMS** mettono a disposizione delle **librerie** per poter **accedere agli oggetti** (*modifiche o interrogazioni*). *Un oggetto viene "lavorato" in memoria centrale e poi riportato in memoria di massa.*

Object Database (OODBMS)

- Consentono la **creazione di un sistema di storage molto affidabile.**

- **PRO:** ***persistenza***, gestione dello ***storage secondario***, ***concorrenza***, ***sistemi ad hoc di query*** e di recupero delle informazioni.

- **CONS:** ***Mancanza di standardizzazione*** e di ***interoperabilità tra i diversi OODBMS.***

OSS. Una db Oracle potrà essere convertito facilmente per gestire dati MS SQL Server. La conversione di un DB ad oggetti è molto più improbabile e complessa.

- **Esempi:** Objectivity, Versant.

Tipologie di Database NoSQL

- Key-Value DB
- Col-Family/Big Table DB
- Document DB
- Graph Database
- XML DB
- Object DB
- **Multivalue DB**
- ACID NoSQL

Multivalue DB

- Sono una tipologia di *database NoSQL* e *multidimensionali* considerati **sinonimo di PICK** (o Pick Operating System).

- Supportano **l'utilizzo di attributi** che **sono** spesso **una lista di valori**, invece di valori singoli come nei classici RDBMS.

- Sono classificati come database NoSQL, ma **l'accesso ai dati è possibile anche utilizzando la sintassi SQL**.

- Non sono standardizzati ma semplicemente classificati in:
  - Pre-relazionale
  - Post-relazionale
  - Relazionale
  - Embedded

Multivalue DB

- In un sistema di database **MultiValue:**
  - Un ***Database*** è chiamato ***"Account"***
  - Una ***Tabella*** è chiamata ***"File"***
  - Una ***Colonna*** è un ***"Attributo"***, composto da **Multivalori e Sottovalori** in modo tale da poter **memorizzare al suoi interno più di un valore**.

- I dati sono memorizzati utilizzando due ***File*** separati:
  - Il primo ***File*** serve per **memorizzare I dati grezzi**.
  - Il secondo chiamato ***"Dictionary"*** serve per **memorizzare il formato di visualizzazione dei dati**.

Esempio

- Si consideri un ***File*** *(tabella)* **"Persona"**, con all'interno l'attributo **"email".** Questo campo potrà memorizzare un numero variabile di indirizzi mail all'interno di uno stesso record.

Multivalue DB

- In questo modo la **lista [mario.rossi@mail.com, m.rossi@mail.net, m_rossi@mail.org]** potrà essere **memorizzata e recuperata** con **una sola query o lettura su disco**, quando si accede al record associato.

- Per poter ottenere la stessa **relazione "1 a molti"** in un **RDBMS tradizionale**, si dovrebbe creare una **tabella addizionale** in cui memorizzare il numero degli indirizzi email legati ad una persona.

- Gli **RDBMS più moderni comunque supportano questo modello dati**, ad esempio in PostgreSQL una colonna può essere di tipo Array o baseType.

- **Modello dei Dati**: Simil Relazionale. Modello di dati che si adatta bene a XML.

- **Esempi:** OpenQM, Jbase

Tipologie di Database NoSQL

- Key-Value DB
- Col-Family/Big Table DB
- Document DB
- Graph Database
- XML DB
- Object DB
- Multivalue DB
- **ACID NoSQL**

ACID NoSQL

- Provano a **combinare le caratteristiche più interessanti** e desiderabili dei database **NoSQL e dei RDBMS**.

- Strumenti come **FoundationDB** e **OracleNoSQL** sono ad esempio dei key-value store replicati e distribuiti con un'architettura di tipo **shared-nothing.**

FoundationDB

- Tutte le operazioni di lettura/scrittura garantiscono il **rispetto delle proprietà** del paradigma **ACID** (*Atomicità – Consistenza – Isolamento - Durabilità*).

OracleDB

- Supporta **operazioni atomiche fatte sulla stessa chiave**, e permette **transazioni atomiche su insiemi di chiavi** che condividono lo stesso percorso chiave principale.

Soluzioni NoSQL Database

MongoDB (www.mongodb.org)

- E' un **NoSQL database**, ***Open Source***.

- E' di tipo **Document-oriented Database** con schemi dinamici (*schemaless*). I dati archiviati sotto forma di *document* in stile **JSON (BSON)**.

- Garantisce il **supporto di indici flessibili** e **query complesse**.

- **Replicazione integrata** di dati per **un'elevata disponibilità**.

- Modulo **auto-sharding** per la **scalabilità orizzontale**.

- Soluzione **sviluppata** per raggiungere **elevate performance** su operazioni di **lettura/scrittura**, e in grado si essere **altamente scalabile con il recupero automatico dei fallimenti**.

JSON

- Acronimo di ***JavaScript Object Notation***, è un formato adatto per lo scambio dei dati in applicazioni client-server.

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

BSON

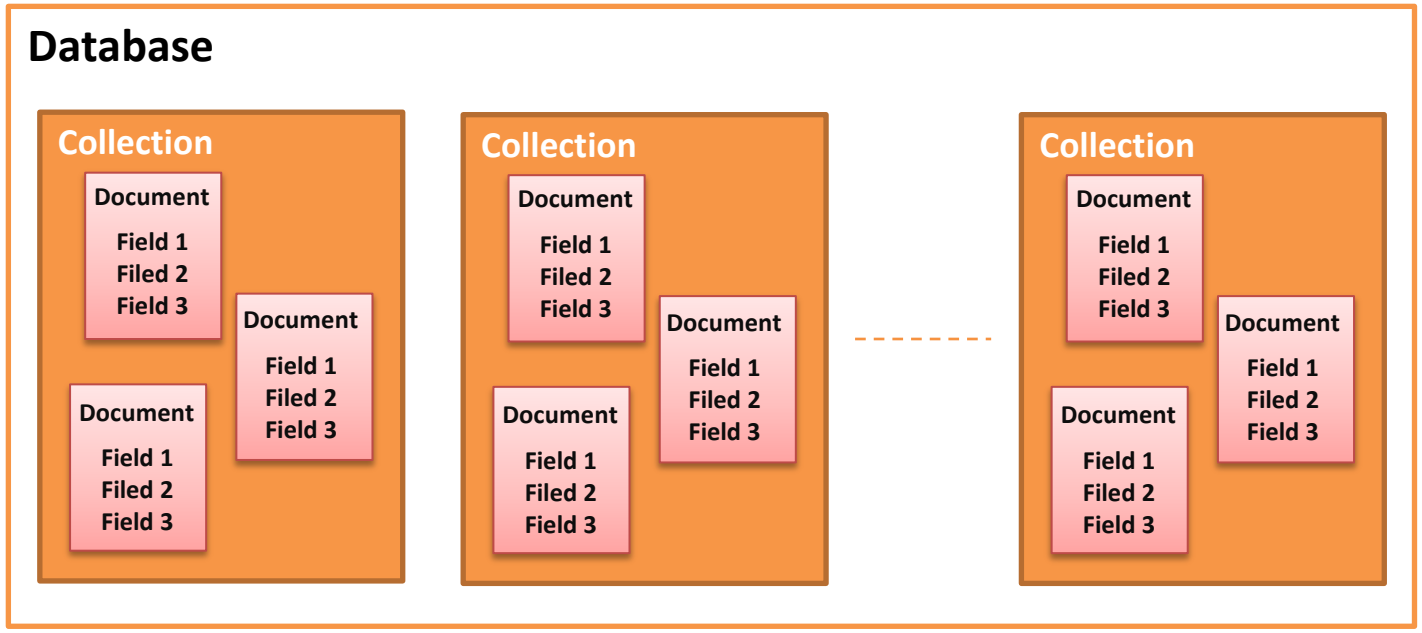
- ***Binary JSON***, è un'estensione del formato JSON che introduce anche il supporto di tipi di dati aggiuntivi: *double, string, array, binary, null, date, int32, int64, object id*…

```
{"hello": "world"}                → "\x16\x00\x00\x00\x02hello\x00
                                    \x06\x00\x00\x00world\x00\x00"

{"BSON": ["awesome", 5.05, 1986]} → "\x31\x00\x00\x00\x04BSON\x00\x26\x00
                                    \x00\x00\x020\x00\x08\x00\x00
                                    \x00awesome\x00\x011\x00\x33\x33\x33\x33\x33\x33
                                    \x14\x40\x102\x00\xc2\x07\x00\x00
                                    \x00\x00"
```

MongoDB – Data Model

- Un sistema basato su **MongoDB** contiene un **set di Database** cosi strutturati:



- Un ***database*** contiene un **set di *collections***, composte al loro interno da un **insieme di *documents***. Ogni ***documents*** è composto da un **set di campi** (***fields***), ciascuno dei quali è una **coppia chiave/valore**.

MongoDB e RDBMS

E’ possibile individuare una sorta di **analogia** tra i concetti dei ***database relazionali*** e quelli presenti in ***MongoDB***.

SQL	Mondo	Differenza
Database	***Database***	
Tabella	***Collection***	Le collection non impongono restrizioni sugli attributi (schemaless)
Riga	***Document***	Volori come oggetti strutturati
Colonna/Campo	***Field***	Maggior supporto ai tipi di dato
Indici	***Indici***	Alcuni indici caratteristici
Join	***Embedding/linking***	Gestione delle referenze lato applicazione
Primary Key	***Object ID***	

MongoDB – Data Model

- La ***chiave*** è una **stringa.**

- Il ***valore*** di un *field* può essere un tipo base (*string, double, date…*), oppure essere a sua volta un **document** o un *array di document*.

- Ciascun documento ha il **campo predefinito "_id"**, che può essere settato in fase di inserimento, altrimenti il sistema ne assegnerà uno univoco in modo automatico.

```
{
  _id: ObjectId ('243252351224df35435bd35wre3'),      //ID del documento
  nominativo: { nome: 'Gino', cognome: 'Bianchi'},    //valore di tipo documento
  data_nascita: new Date('Feb 12, 1975'),
  PIVA: '112334797634',                               //valori con tipo di dato base
  promozioni_utilizzate: ['PROMO 1', 'PROMO4'],       //array
}
```

MongoDB - Caratteristiche

- **Query ad hoc**

Supporta le **ricerche per campi**, **range queries** , le **ricerche tramite espressioni regolari** che restituiscono specifici campi dei documenti memorizzati o tramite funzioni definite dall'utente.

- **Indicizzazione**

Include il **supporto su molti tipi di indici** definiti su qualsiasi campo dei documenti in memoria. In un certo senso gli indici sono concettualmente simili ai RDBMS.

- **Replicazione** (*Paradigma Master-Slave*)

Il **Master** è in grado di realizzare **operazioni di scrittura-lettura**. Gli **Slave copiano i dati dal Master**, e possono essere utilizzati solo per realizzare **letture di dati o operazione di backup**.

MongoDB - Caratteristiche

- **Bilanciamento del carico**

Questo DB è in grado di essere **eseguito su server multipli**, **bilanciando il carico di dati e/o duplicandoli** con l'obiettivo di **mantenere attivo l'intero sistema** nel caso in cui si verificasse un **guasto hardware**.

- **Archiviazione dei file**

Può essere **usato come file system** (*GridFS*), i file possono essere distribuiti e copiati più e più volte sulle diverse macchine in modo trasparente.

- **Aggregazione**

Integra un framework MapReduce (non basato su Hadoop) utilizzato per le **elaborazioni in batch dei dati** e per le **operazioni di aggregazione** (GROUP BY).

MongoDB - Caratteristiche

- **Capped collections**

Supporta la **gestione di collezioni di dimensioni fisse** in cui è conservato **l'ordine di inserimento** degli oggetti. Hanno un **comportamento simile alle code circolari** (Sulla base di un nuovo inserimento, il dato più vecchio viene *"sovrascritto quando la coda è piena"*).

- **Aggiornamento documenti "in-place"**

lo **spazio** del disco riservato in cui è **memorizzato un documento rimane sempre lo stesso nello operazioni di update** *(< costi di scrittura su disco, di allocazione memoria e indicizzazione*).

- **MMAP function**

Tutti i **dati sono mappati in memoria**. Utilizzando i **memory-mapped file è possibile tener traccia di questa mappatura** ottenendo un *metodo facile e veloce per l'accesso ai dati e la loro manipolazione*.

MongoDB - Caratteristiche

- **Database Level Locks (da V.2.2)**

E' stata introdotta la possibilità di bloccare i singoli DB e non l'intero sistema nelle operazioni di scrittura o di lettura concorrente.

- **Ricerca di documenti**

Sono presenti comandi potenti e flessibili per ricercare documenti attraverso i loro attributi interni (che necessitano di essere indicizzati).

- **Sharding**

Ogni nodo secondario ha i dati completi presenti sul nodo primario. Aggiungere ulteriori partizioni di sharding in un momento successivo è un'operazione onerosa in termini di difficoltà e di tempo.

- **Formato JSON:** i dati non sono memorizzati in righe e colonne, ma in una forma binaria di documenti JSON (BSON). Lo schema non è fisso, cambia su diversi documenti è può essere variato in tempi molto rapidi.

MongoDB – Modellazione dei dati

La **modellazione dei dati** in MongoDB tiene conto in particolare di **tematiche** che riguardano:

- L'***indicizzazione***
- La ***normalizzazione*** o ***denormalizzazione*** dei dati
- Lo ***Sharding***

MongoDB - Indicizzazione

- Gli **indici** sono **simili** a quelli dei **DB relazionali**, cioè sono delle **strutture B-tree**.

  (+) Consentono il **recupero efficiente dei dati**.

  (-) Rallentano le operazioni di scrittura dei dati (ad ogni scrittura va aggiornato anche l'indice).

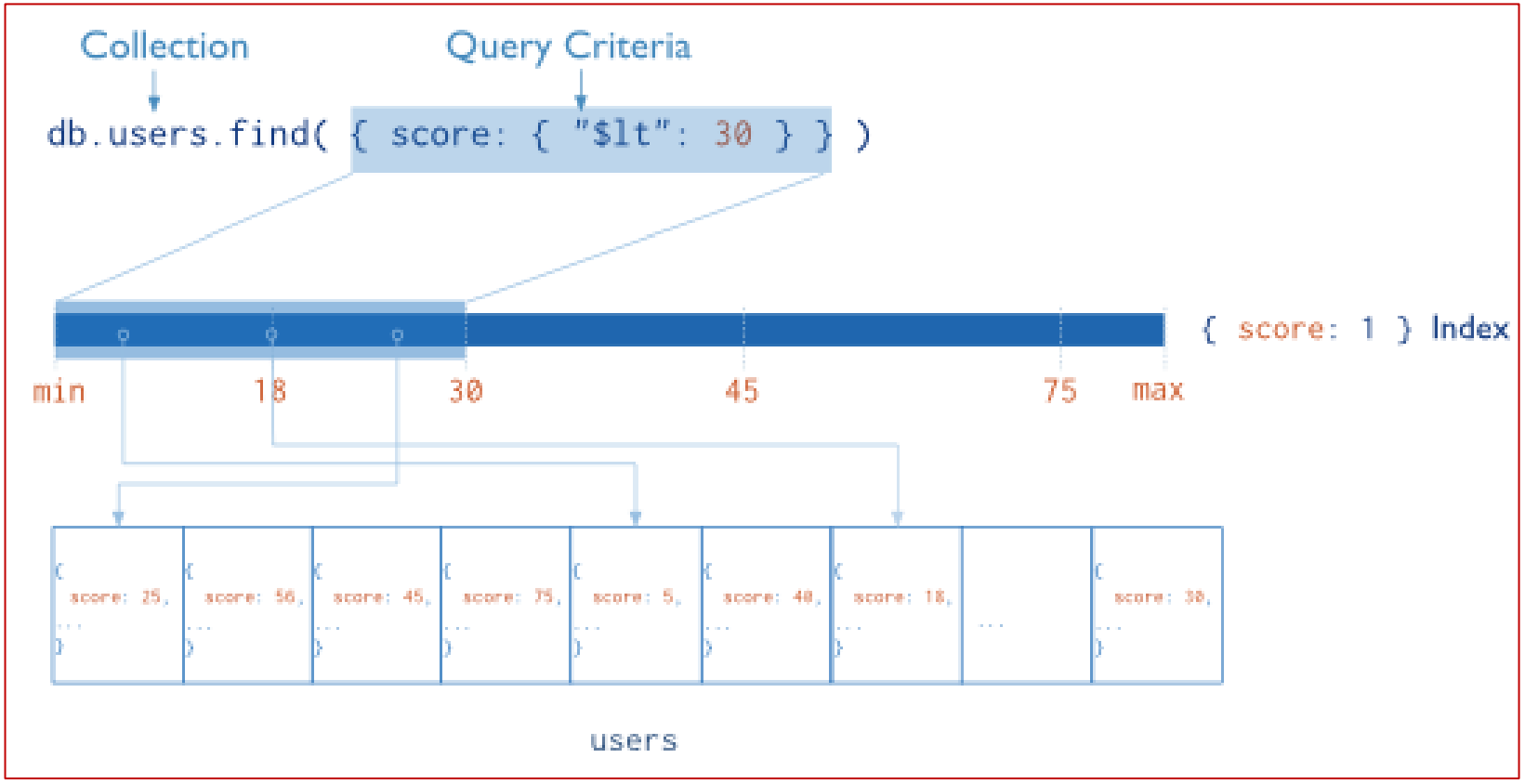
- E' possibile creare **indici secondari**, oltre a **quello univoco primario** creato in modo automatico sul campo ***"_id"***.

- Gli indici sono definiti a livello di ***collection***, e possono includere uno o più ***campi***.

- E' bene **costruire gli indici** sui **campi maggiormente utilizzati** come **chiave di ricerca nelle query**, o su quelli **più selettivi** (*restituiscono valori univoci o poche repliche*).

MongoDB - Indicizzazione

- Gli **indici** sono delle **strutture dati speciali** che ***memorizzano una piccola porzione dell'intero dataset*** di una collection in modo da ottimizzare l'esecuzione delle query.



MongoDB - Tipologie di Indici

Indici univoci

- **Non è consentito** l'inserimento di nuovi **documenti con un valore duplicato del campo** su cui è stata realizzata **l'indicizzazione**.

Indici composti

- Una **singola struttura di indicizzazione** mantiene il **riferimento a campi multipli** di un documento. Sono *utili* per le **query che combinano più campi** (*ad esempio catalogo di prodotti, o insieme di clienti*). L'ordine in cui i campi vengono specificati è importante ai fini della ricerca.

Indici Array o Multikey

- Usati per **indicizzare campi** che contengono un **array**, ogni valore dell'array viene memorizzato come elemento di uno specifico indice (*ad esempio per organizzare un **insieme di ricette**, in cui il campo ingredienti sarà quasi sicuramente un array*).

MongoDB - Tipologie di Indici

Indici Time To Live (TTL)

- Consentono agli utenti di **specificare un periodo di tempo dopo il quale i dati verranno automaticamente cancellati** (*esempio il clickstream di utenti web*).

Indici Sparsi

- Piccoli, sono gli **indici più efficienti quando i campi non sono presenti in tutti i documenti memorizzati** (*hanno buona flessibilità*).

Indici di ricerca testuali

- Sono **indici specializzati nelle ricerche testuali** che utilizzano **regole linguistiche** avanzate, specifiche della lingua per *serializzare, troncare o separare le parole*.

Indici geospaziali bidimensionali ***

- Ottimizzano le **query** relative a **dati posizionali in uno spazio bidimensionale**.

MongoDB – Modellazione dei dati

La **modellazione dei dati** in MongoDB tiene conto in particolare di **tematiche** che riguardano:

- L'***indicizzazione***
- La ***normalizzazione*** o ***denormalizzazione*** dei dati
- Lo ***Sharding***

MongoDB - Denormalizzazione

- **MongoDB** ha uno **schema flessibile**. I *documenti* **non hanno vincoli di struttura** imposti dalle *collection*, infatti:

    1. Stessa ***collection*** -> ***documenti*** con **differenti campi o tipi di dato**.

    1. Un **campo** di un documento può contenere una **struttura complessa** come un ***array di valori*** o un ***altro documento***.

In base alla seconda caratteristica è possibile:

- **Denormalizzare la struttura dati. Tutti i dati** che descrivono un entità e quelli ad essi collegati **sono inclusi all'interno del documento**.

- **Si mantengono i documenti separati**, aggiungendo però ad ognuno di essi un riferimento che permetta di legarli, es. *il campo "_id"*.

MongoDB - Denormalizzazione

Struttura Denormalizzata

```
db.autori.insert({
    "_id": 3,
    "Nome": "Gabriele",
    "Cognome": "D'Annunzio",
    "opere": [
                {   "titolo": "il piacere",
                    "anno": "1889",
                },
                {   "titolo": "il fuoco",
                    "anno": "1900",
                },
                {   "titolo": "la figlia di Iorio",
                    "anno": "1903",
                }
            ]
    })
```

Struttura Normalizzata

```
db.autori.insert({
    "_id": 3,
    "Nome": "Gabriele",
    "Cognome": "D'Annunzio",
})
db.libri.insert({
    "titolo": "il piacere",
    "anno": "1889",
    "id_autore": 3
})
db.libri.insert({
    "titolo": "il fuoco",
    "anno": "1900",
    "id_autore": 3
})
db.libri.insert({
    "titolo": "la figlia di Iorio",
    "anno": "1903",
    "id_autore": 3
})
```

- **MongoDB non supporta le *JOIN***. Se si adotta la *struttura normalizzata*, le **relazioni** tra i documenti si esplicitano tramite **query in cascata** che recuperano i documenti.

- **Convenzione DBRef** – specifica un ***documento*** tramite: ***database -> collection -> _id***

MongoDB – Modellazione dei dati

La **modellazione dei dati** in MongoDB tiene conto in particolare di **tematiche** che riguardano:

- L'***indicizzazione***
- La ***normalizzazione*** o ***denormalizzazione*** dei dati
- **Lo *Sharding***

MongoDB - Sharding

- Lo ***sharding*** è quel processo che permette il **partizionamento** di una *collection*, cioè di **suddividere i suoi documenti tra più istanze di MongoDB** (*architettura distribuita*).

- La **distribuzione dei documenti** tra le istanze è **basata su intervalli della chiave di partizionamento**, chiamata ***shard key.***

Shard Key

- Deve essere un **campo appartenente ad ogni documento**.
- **Campo utilizzato spesso nelle query** di ricerca.
- Scelta tra i **campi con un'alta cardinalità**.
- **Non** dovrebbe essere **legata al "tempo" (data) di inserimento**. Una maggiore casualità garantisce una distribuzione più uniforme.

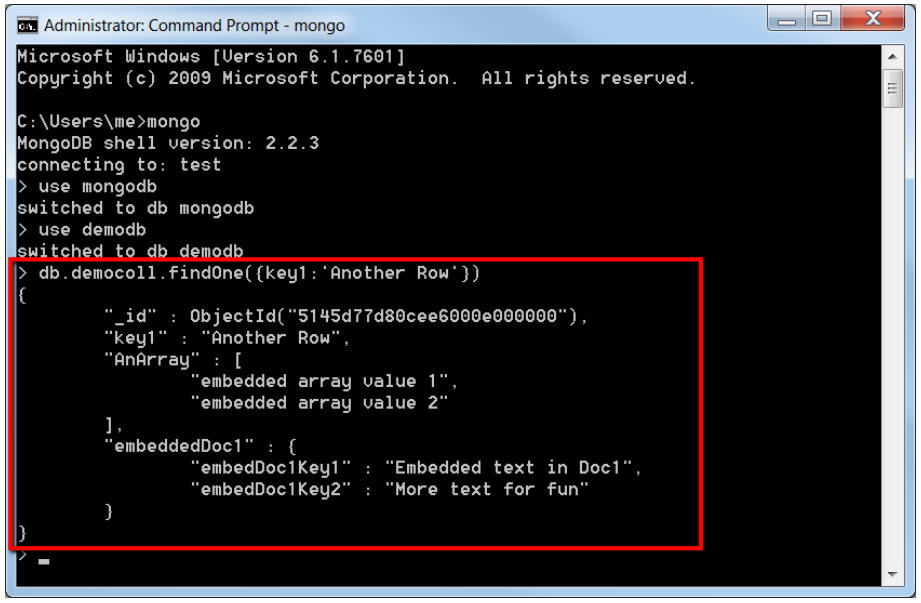
- Se la **scelta della chiave è complessa**, si può applicare ad essa una **funzione hash** (***hashed shard***) al campo candidato ad essere chiave (anche al campo ***_id***).

MongoDB - Interazione

L'**interazione con MongoDB** può avvenire in due modi differenti:

- Tramite l'utilizzo della **shell**, cioè lo strumento da riga di comando che supporta il linguaggio *JavaScript.*

- Tramite l'utilizzo delle **API** disponibili per **numerosi linguaggi di programmazione** come *C, C++, Java, C#, Ruby, Python, JavaScript* etc.



MongoDB - Interazione

- Da **riga di comando** è possibile *eseguire delle* ***semplici istruzioni***, oppure *lanciare* ***script*** *salvati su file*.

Istruzione	Esempio	Descrizione
db	`db`	Restituisce il database corrente.
use <nome db>	`use impiegati`	Cambia il db corrente in impiegati.
db.<collection>.insert()	`db.studenti.insert( { nome: 'Gianni', Cognome: 'Verdi'})`	Inserisce un utente nella collection studenti senza specificare l'ID
db.<collection>.find()	`dc.studenti.find()`	Elenca tutti i documenti della collection studenti
db.<collection>.drop()	`db.studenti.drop()`	Elimina l'intera collection dal DB

- Oltre ai comandi per effettuare ***inserimenti, modifiche e recupero*** *di documenti*, esistono anche i **cursori**, *cioè degli oggetti che permettono di* ***iterare su un set di documenti*** *restituiti come* ***risultato di una query***.

MongoDB - Interazione

```java
import com.mongodb.MongoClient;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBCursor;

public class MongoDB_test {
    public static void main(String args[]) throws Exception {
        //istanza studente
        MongoClient mongoClient = new MongoClient();
        //recuperiamo il database
        DB db = mongoClient.getDB("test");
        //creiamo una collection
        DBCollection coll = db.createCollection ("studenti", null);
        //creiamo un documento
        BasicDBObject doc =
            new BasicDBObject("nome", "Gianni").
            append("cognome", "Verdi").
            append("indirizzo", newBasicDBObject("comune", "Firenze")).
            append("provincia", "FI"));
        //aggiungiamo il documento alla collection
        coll.insert(doc);
        //creiamo un documento per l'esecuzione di una query
        BasicDBObject query = new BasicDBObject ("cognome", "Verdi");
        //utilizziamo un cursore per iterare sui risultati della
        //query e mostrare ogni documento
        DBCursor cursor = coll.find(query);
        try {
            while(cursor.hasNext()){
                System.out.println(cursor.next());
            }
        } finally {
                cursor.close();
        }
    System.out.println("");
    }
}
```

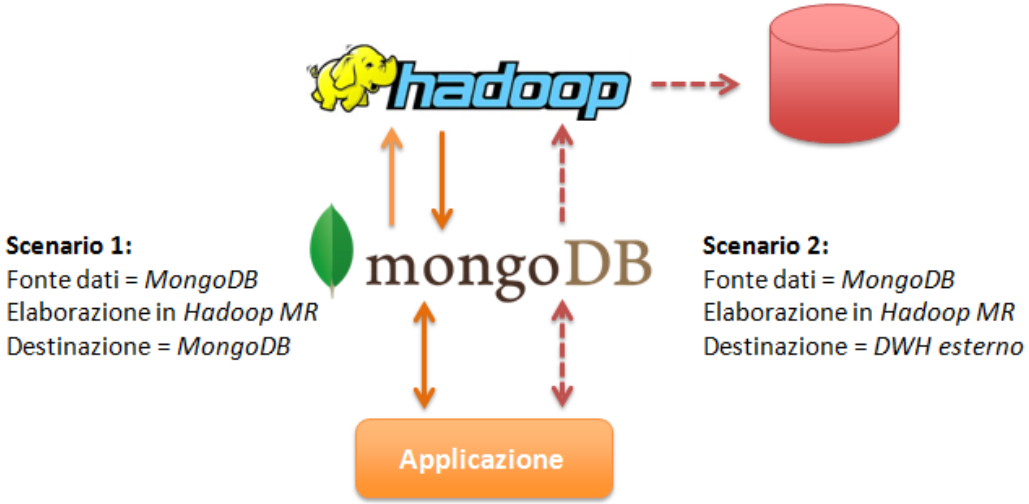
Nell' esempio con l'uso delle **API Java** si:

- **Creano le istanze** del **client *MongoDB*** e del **database *test***.
- **Crea** la nuova **collection *"studenti"***.
- **Crea** un **documento** che modella uno studente, con i campi ***nome, cognome e indirizzo*** (*documento annidato*).
- **Inserisce** il documento nella collection.
- **Crea** e si esegue la **query** che effettua una ricerca sul campo cognome.
- **Usa** un ***cursore*** per stampare a video i documenti recuperati dalla query.

MongoDB - Computazione

- Per **operazioni di calcolo** con un'***alta complessità*** e che coinvolgono una ***grande mole di dati*** è meglio ricorrere a **strumenti potenti** come **MapReduce**.

- In MongoDB si può ricorrere all'uso di MapReduce in due modi:
  1. Uso della funzione integrata ***mapreduce*** (da **shell** o tramite **API**).
  2. Uso dell'adapter per **l'integrazione con Hadoop**.



MongoDB - Vantaggi

Migliori prestazioni

- Presenza di **meccanismi di caching**.
- MongoDB fa **ampio uso di RAM** per velocizzare le operazioni del database.
- Tutti i **dati vengono letti e manipolati tramite i file mappati in memoria**. I dati a cui non si accede non vengono caricati in RAM.

Flessibilità

MongoDB fornisce grande flessibilità allo sviluppo grazie a:

- **Modello dati "document oriented".**
- **Implementazioni multi-datacenter.**
- **Livelli di coerenza regolabili**, con la possibilità di *personalizzare la disponibilità delle risorse* a diversi livelli operazionali.

MongoDB - Vantaggi

Modello di query semplice e ricco

- Possibilità di definire **diversi tipi di indici** che consentono di eseguire **molte tipologie di query** sui dati.
- Questa caratteristica rende MongoDB adatto per un'ampia varietà di applicazioni.

Diversi tipi di Sharding

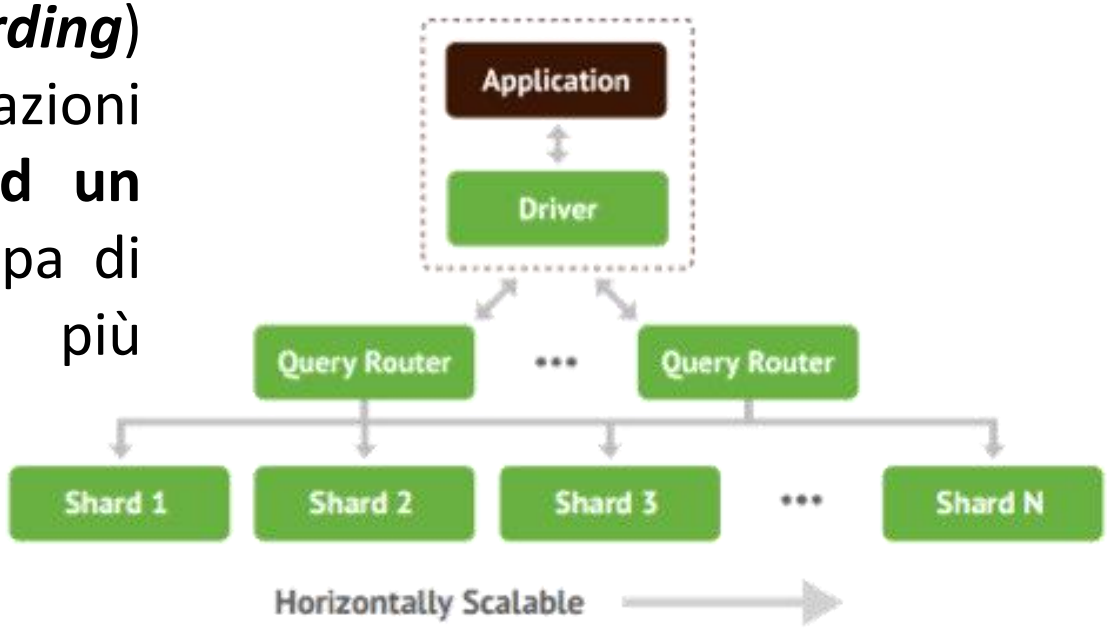
Per *ridurre i limiti* di un **unico server** (collo di bottiglia su RAM o disco I/O) senza aggiungere complessità. E' possibili implementare d**iverse tipologie di sharding**:

- **Range-based sharding** (*shard-key per vicinanza*)
- **Hash-based Sharding** (*MD5 hash*)
- **Tag-aware Sharding** (*configurazione utente*).

MongoDB - Vantaggi - Automatic Sharding

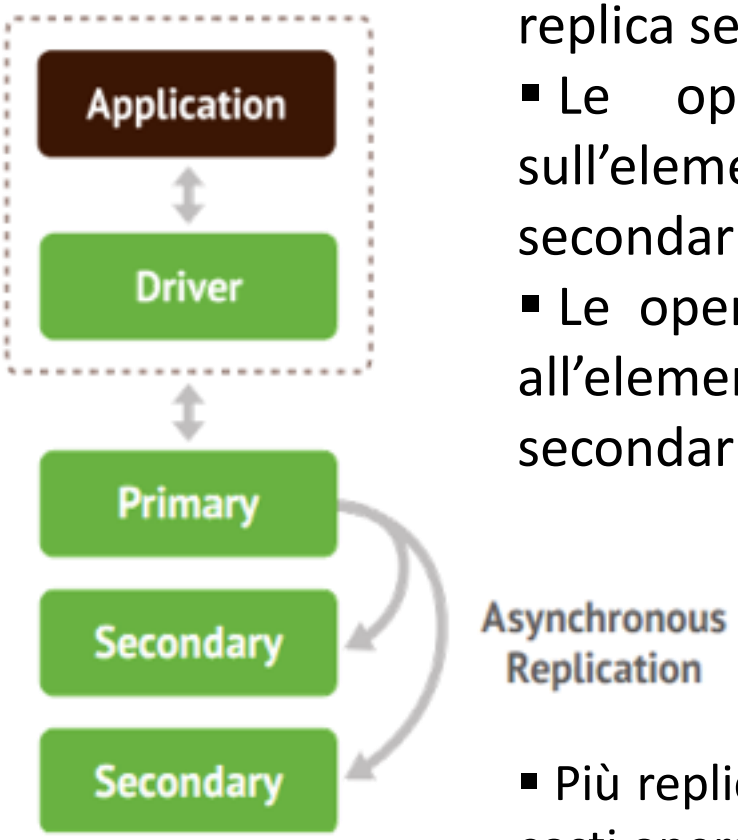
- La **distribuzione dei dati nel cluster** viene **bilanciata in modo automatico** col crescere del numero dei dati oppure quando la dimensione del cluster stesso aumenta o diminuisce.

- Il **Partizionamento** (***sharding***) è trasparente alle applicazioni che **inviano le query ad un *query router*** che si occupa di smistarle alle partizioni più appropriate a gestirle.



- *Risultati pertinenti*
- *Maggiore velocità di risposta*

MongoDB - Vantaggi - Facile Replicabilità



- Un elemento funge da replica primaria, gli altri da replica secondaria.
- Le operazioni che modificano il database sull'elemento primario vengono replicate anche sui secondari con un log (oplog).
- Le operazioni di lettura/scrittura sono indirizzate all'elemento primario; se questo fallisce uno dei secondari prende il suo posto.

- Più repliche aumentano la durabilità dei dati, riducono i costi operativi e migliorano la disponibilità del sistema.
- Il numero di repliche è configurabile dall'utente.

MongoDB – Startup e connessione al DB

Il pacchetto MongoDB (*http://www.mongodb.org/downloads*) contiene *due programmi eseguibili* (che su Windows sono):

- **mongod.exe** – database server
- **mongo.exe** – administrative shell

Per **lanciare il database** bisogna usare *mongod.exe*, mentre per accedere alla **shell interattiva**, va avviato *mongo.exe.*

```
1  # 'mongo' is shell binary. exact location might vary depending o
2  # installation method and platform
3  $ bin/mongo
```

MongoDB – Startup e connessione al DB

- Di default, la **shell** si connette al database ***"test"*** su *localhost*. Se si vuole *connettersi ad un altro database* bisogna utilizzare il *comando* ***use.***

```
1 > use mydb
2 switched to db mydb
```

- Se il **database** (es. *mydb*) **non esiste**, il comando ***use*** lo crea, *ma per mantenerlo registrato bisogna inserire almeno un dato*.
- Il comando ***show dbs***, *visualizza i database presenti in MongoDB*.
- ***help*** *mostra l'elenco dei comandi e relativa descrizione*.

MongoDB – Inserimento dati in una collezione

Creiamo *due oggetti*, ***j*** e ***t***, e li salviamo nella *collection* ***things***.

```
> j = { name : "mongo" };
{"name" : "mongo"}
> t = { x : 3 };
{ "x" : 3 }
> db.things.save(j);
> db.things.save(t);
> db.things.find();
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
{ "_id" : ObjectId("4c2209fef3924d31102bd84b"), "x" : 3 }
>
```

- **Non occorre** definire la *collection*, *il database la crea (se non esiste). in modo automatico* all'esecuzione della prima insert.
- I documenti memorizzati possono avere **campi differenti** (come nell'esempio), anche se è buona norma memorizzare documenti con la *stessa struttura* nella *stessa collection*.

MongoDB – Cursori

Ricordiamo che con i ***cursori*** è possibile *elencare/visualizzare* tutti i documenti di una data *collection*.

- Un ***cursore***, però, può essere utilizzato anche per **recuperare uno specifico *documento*** in una ***collection***.

```
> var cursor = db.things.find();
> printjson(cursor[4]);
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "x" : 4, "j" : 3 }
```

- Lo *stesso risultato si ottiene* anche con un approccio ***array-style.***

```
> var arr = db.things.find().toArray();
> arr[5];
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
```

MongoDB – Tipologie di Query

Query chiave-valore

- Restituiscono **risultati basati** su un **qualsiasi campo nella del documento** (chiave primaria).

Range query

- Restituiscono **risultato sulla base di intervalli di valori di ricerca** definiti come disuguaglianze (>, < o =).

Query geospaziali

- Restituiscono **risultati in base a criteri di prossimità**, di **intersezione** e **inclusione** specificati da un punto, linea, cerchio o un poligono.

MongoDB – Tipologie di Query

Query di ricerca testuale

- Restituiscono **risultati in ordine di rilevanza sulla base di argomenti di testo** (caratterizzati da operatori logici AND, OR, NOT).

Aggregation Framwork Query

- Danno come **risultato aggregazioni di valori** sulla base di specifiche caratteristiche o criteri di raggruppamento (GROUP BY).

MapReduce Query

- **Eseguono complesse elaborazioni dati** (magari utilizzando delle funzioni scritte in JavaScript) su **tutti i dati** presenti al'interno del **database**.

MongoDB – Esempi di Query

1. ***Selezione di tutti i documenti in una collezione***

```
db.inventario.find()
```

2. ***Corrispondenza esatta su un Array (campo multivalore)***

```
db.inventario.find({tags:['frutta',‘carne',‘pesce']}
)
```

3. ***Specifica condizione AND***

```
db.inventario.find({type:‘carne',prezzo:{$lt:9.95}})
```

4. ***Corrispondenza su un campo in un Subdocument***

```
db.inventario.find({'producer.company': 'ABC123’})
```

MongoDB – Esempi di Query (1)

SELECET * FROM things WHERE name=“mongo”

```
> db.things.find({name:"mongo"}).forEach(printjson);
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
```

**SELECET *
FROM things
WHERE x=4**

```
> db.things.find({x:4}).forEach(printjson);
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "x" : 4, "j" : 1 }
{ "_id" : ObjectId("4c220a42f3924d31102bd857"), "x" : 4, "j" : 2 }
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "x" : 4, "j" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85a"), "x" : 4, "j" : 5 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85b"), "x" : 4, "j" : 6 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85c"), "x" : 4, "j" : 7 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85d"), "x" : 4, "j" : 8 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85e"), "x" : 4, "j" : 9 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85f"), "x" : 4, "j" : 10 }
{ "_id" : ObjectId("4c220a42f3924d31102bd860"), "x" : 4, "j" : 11 }
{ "_id" : ObjectId("4c220a42f3924d31102bd861"), "x" : 4, "j" : 12 }
{ "_id" : ObjectId("4c220a42f3924d31102bd862"), "x" : 4, "j" : 13 }
{ "_id" : ObjectId("4c220a42f3924d31102bd863"), "x" : 4, "j" : 14 }
{ "_id" : ObjectId("4c220a42f3924d31102bd864"), "x" : 4, "j" : 15 }
{ "_id" : ObjectId("4c220a42f3924d31102bd865"), "x" : 4, "j" : 16 }
{ "_id" : ObjectId("4c220a42f3924d31102bd866"), "x" : 4, "j" : 17 }
{ "_id" : ObjectId("4c220a42f3924d31102bd867"), "x" : 4, "j" : 18 }
{ "_id" : ObjectId("4c220a42f3924d31102bd868"), "x" : 4, "j" : 19 }
{ "_id" : ObjectId("4c220a42f3924d31102bd869"), "x" : 4, "j" : 20 }
```

MongoDB – Esempi di Query (3)

```
SELECET *
FROM things
WHERE x="4"
AND j="true"
```

```
> db.things.find({x:4}, {j:true}).forEach(printjson);
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "j" : 1 }
{ "_id" : ObjectId("4c220a42f3924d31102bd857"), "j" : 2 }
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "j" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "j" : 4 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85a"), "j" : 5 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85b"), "j" : 6 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85c"), "j" : 7 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85d"), "j" : 8 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85e"), "j" : 9 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85f"), "j" : 10 }
{ "_id" : ObjectId("4c220a42f3924d31102bd860"), "j" : 11 }
{ "_id" : ObjectId("4c220a42f3924d31102bd861"), "j" : 12 }
{ "_id" : ObjectId("4c220a42f3924d31102bd862"), "j" : 13 }
{ "_id" : ObjectId("4c220a42f3924d31102bd863"), "j" : 14 }
{ "_id" : ObjectId("4c220a42f3924d31102bd864"), "j" : 15 }
{ "_id" : ObjectId("4c220a42f3924d31102bd865"), "j" : 16 }
{ "_id" : ObjectId("4c220a42f3924d31102bd866"), "j" : 17 }
{ "_id" : ObjectId("4c220a42f3924d31102bd867"), "j" : 18 }
{ "_id" : ObjectId("4c220a42f3924d31102bd868"), "j" : 19 }
{ "_id" : ObjectId("4c220a42f3924d31102bd869"), "j" : 20 }
```

MongoDB – Esempi di Query (4)

Corrispondenza esatta su un Subdocument

```
db.inventario.find(
    {
        producer: {
            company: 'ABC123',
            indirizzo: 'Via Verdi 4'
        }
    }
)
```

Limitazione dei risultati

- Il metodo collection `db.collection.findOne()` corrisponde a `db.collection.find()` con la limitazione a restituire un solo risultato.