Database NoSql

- **Entità**
- **Modelli**
- **Definizione di database**
- **Database NoSql**
- **Storia dei database NoSql**
- **Tipologia di database NoSql**
- **Campi di applicazione del database NoSql mongodb**
- **Applicazioni client e server**
- **Il database mongodb**
- **Il client mongodb Compass**
- **Un esempio di app client in python utilizzando tkinter**
- **URL (uniform resource locator)**
- **Framework**
- **Template**
- **La libreria Flask di python (creare server web)**
- **Un esempio di app client-server in python con mongodb e Flask**

Introduzione

1. I sistemi : definizioni ed esempi

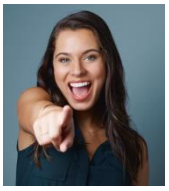
In informatica per sistema si intende, **un insieme di componenti interconnessi che lavorano insieme per raggiungere un determinato obiettivo.** E' una cosa reale che esiste e funziona.
Spesso si presenta come unico oggetto od entità, persona animale o concetto astratto

Esempi in informatica:

- Un personal computer, un cellulare, una stampante, un tablet, un server, una rete di computer, un sistema di gestione database, un sistema di elaborazione di transizione online, ecc..
- Una automobile, un robot, uno scooter, una lavatrice, una nave,un trattore, un distributore di bevande o di carburante , ecc..
- Una persona, un animale
- Un edificio, una scuola, una biblioteca, sistema impiantistico (idraulico, elettrico) ecc..







Riassumendo:

1. Un sistema è composto da diverse parti che hanno funzioni specifiche
2. Le parti sono interconnesse tra loro ed interagiscono per raggiungere l'obiettivo del sistema: ad esempio un'auto è composta da: motore, telaio, organi di trasmissione, carrozzeria, ruote, ecc..
3. Un sistema svolge nel suo complesso una funzione specifica: ad esempio l'auto trasporta le persone, il camion persone e merci, la stampante trasferisce le informazioni su carta

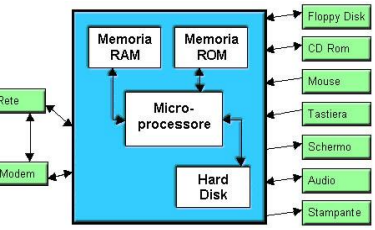
2. I modelli

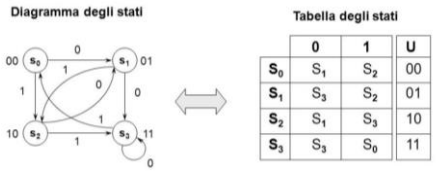
Un modello è una rappresentazione semplificata di un sistema :

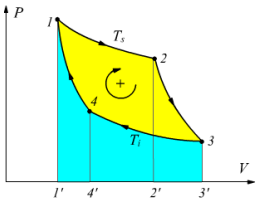
Esempi di modelli possono essere:

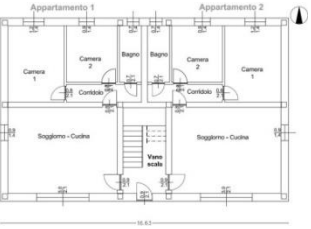
- Un diagramma, una tabella, un grafico, un grafo, una piantina di un edificio
- Un insieme di equazioni matematiche
- Un programma (software) con simulazione al computer
- Un disegno al cad, una simulazione al cad
- Un database, un modello OOP (classi ed oggetti)

L'obiettivo di un modello è di aiutarci a capire come funziona un sistema, a progettarlo, o a prevederne il comportamento

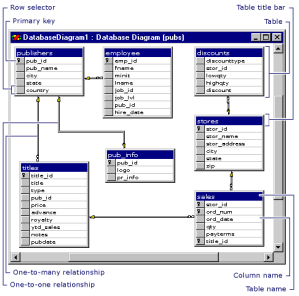


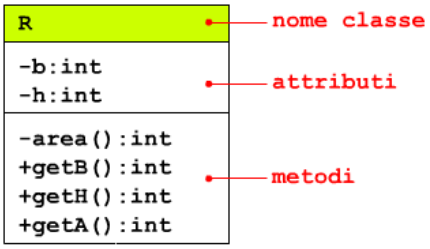






$$\begin{cases} y = 2x + 1 \\ 3x + y = 0 \end{cases}$$





Definizione di database (archivio elettronico)

Un database è un insieme di file in binario salvato su disco"?

No, la definizione di un database come "un insieme di file in binario salvato su disco" è una **descrizione molto semplificata e incompleta**. Sebbene i dati di un database alla fine vengano memorizzati su un supporto fisico come un disco rigido in formato binario, un database è **molto di più** di una semplice collezione di file.
Ecco una definizione più completa e precisa di cosa sia un database:
Un database è una collezione organizzata di dati strutturati, tipicamente memorizzata elettronicamente in un sistema informatico, in modo da poter essere facilmente accessibile, gestita e aggiornata.
Vediamo i punti chiave di questa definizione:

- **Collezione organizzata:** I dati in un database non sono semplicemente ammassati. Sono strutturati in modo logico per rappresentare entità, relazioni e informazioni del mondo reale. Questa organizzazione permette di trovare e gestire i dati in modo efficiente.
- **Dati strutturati:** I dati hanno un formato definito (ad esempio, tabelle con righe e colonne, documenti con campi specifici, grafi con nodi e archi). Questa struttura consente di interrogare e analizzare i dati in modo significativo.
- **Tipicamente memorizzata elettronicamente:** La maggior parte dei database moderni sono implementati su sistemi informatici e memorizzano i dati digitalmente.

- **Facilmente accessibile, gestita e aggiornata:** Un database è progettato per consentire agli utenti e alle applicazioni di interagire con i dati in modo controllato e sicuro. Questo include operazioni come l'inserimento, la lettura, la modifica e l'eliminazione dei dati.

Oltre alla semplice memorizzazione di file binari, un database include:

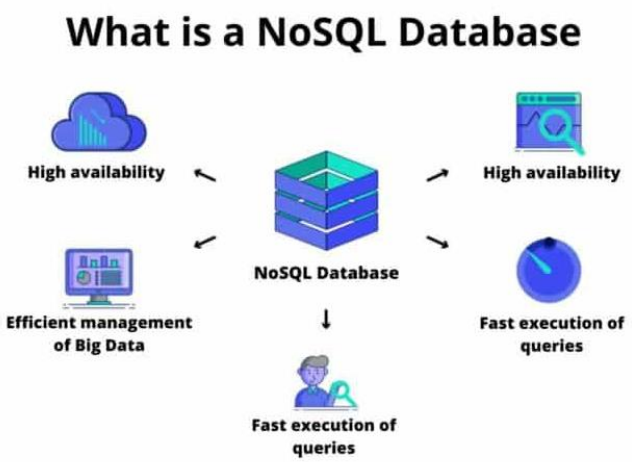
- **Schema del database:** La definizione della struttura del database, incluse le tabelle, le colonne, i tipi di dati, le relazioni tra le tabelle e i vincoli di integrità.
- **Sistema di gestione di database (DBMS):** Un software che permette di interagire con il database. Il DBMS fornisce funzionalità per definire, creare, interrogare, aggiornare e amministrare il database. Esempi di DBMS includono MySQL, PostgreSQL, Oracle, SQL Server e MongoDB.
- **Linguaggio di interrogazione:** Un linguaggio specifico (come SQL per i database relazionali) utilizzato per interagire con il database, recuperare informazioni e manipolare i dati.
- **Meccanismi di controllo e sicurezza:** Funzionalità per garantire l'integrità dei dati, la concorrenza degli accessi e la sicurezza contro accessi non autorizzati.
- **Indici e altre strutture di dati:** Ottimizzazioni per velocizzare le operazioni di ricerca e recupero dei dati.

In sintesi:

Un database è un sistema complesso che va ben oltre una semplice raccolta di file binari. È una struttura organizzata e gestita da un software specifico (DBMS) per archiviare, recuperare e manipolare dati in modo efficiente e affidabile. I file binari sono solo il modo in cui i dati vengono fisicamente memorizzati sul disco, ma la vera essenza di un database risiede nella sua organizzazione logica, nella sua struttura e nelle funzionalità offerte dal DBMS.

Database NoSQL

I database NoSQL non hanno una data di nascita precisa come un singolo prodotto. Il termine "NoSQL" è emerso più chiaramente **alla fine degli anni 90 inizio 2000**, con la crescente popolarità di diverse tipologie di database che non aderivano al modello relazionale tradizionale e al linguaggio SQL.
Tuttavia, le idee e le tecnologie alla base dei database NoSQL hanno radici più antiche. Già negli anni '70 e '80 esistevano sistemi di gestione dati che non erano puramente relazionali, come i database gerarchici e di rete. Inoltre, alcuni sistemi di archiviazione chiave-valore possono essere considerati precursori dei moderni database NoSQL.



L'**esigenza principale** che ha portato alla ribalta i database NoSQL è stata la necessità di gestire le sfide poste dalle **applicazioni web moderne su larga scala** e dalla **crescente quantità e varietà di dati**. In particolare:

- **Scalabilità orizzontale:** I database relazionali tradizionali spesso faticavano a scalare orizzontalmente (aggiungendo più server) in modo efficiente ed economico per gestire l'enorme traffico e i volumi di dati generati dalle applicazioni web e dai social media. I database NoSQL sono stati progettati per essere distribuiti su più nodi, offrendo una migliore scalabilità orizzontale.
- **Flessibilità dello schema:** Le applicazioni moderne spesso lavorano con dati eterogenei non strutturati o semi-strutturati (come documenti JSON, XML, grafi), che non si adattano facilmente agli schemi rigidi dei database relazionali. I database NoSQL offrono schemi più flessibili e dinamici, consentendo di gestire diversi tipi di dati senza dover definire tabelle e colonne fisse in anticipo.
- **Alte prestazioni:** Per molte applicazioni web in tempo reale, è fondamentale avere tempi di risposta rapidi. Alcuni database NoSQL sono ottimizzati per operazioni di lettura e scrittura veloci, spesso sacrificando alcune delle proprietà ACID (Atomicità, Coerenza, Isolamento, Durabilità) tipiche dei database relazionali in favore di una maggiore disponibilità e tolleranza alle partizioni (secondo il teorema CAP).
- **Agilità nello sviluppo:** La flessibilità dello schema dei database NoSQL si adatta meglio ai cicli di sviluppo agili, in cui i requisiti dei dati possono cambiare rapidamente. Gli sviluppatori possono modificare la struttura dei dati senza dover eseguire migrazioni complesse dello schema.

In sintesi, i database NoSQL sono nati come risposta alle limitazioni dei database relazionali nel gestire le esigenze di scalabilità, flessibilità e prestazioni delle applicazioni web moderne e della gestione di grandi volumi di dati eterogenei.

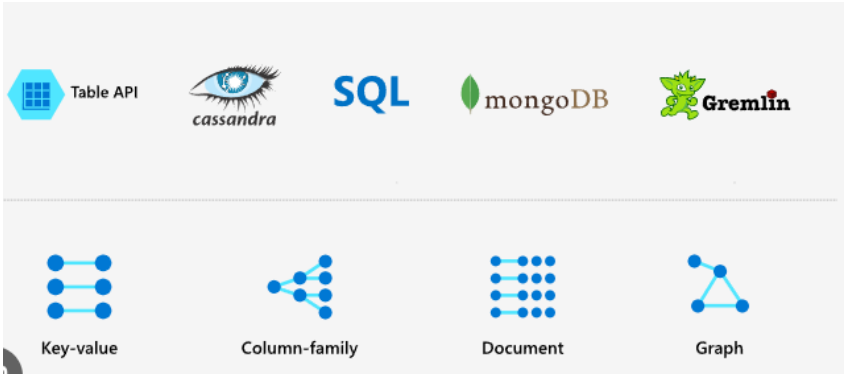
Storia dei database NoSql

La nascita dei database NoSQL è un argomento interessante con una storia un po' meno lineare di quanto si potrebbe pensare.
Sebbene il termine "NoSQL" abbia guadagnato popolarità verso la fine degli anni 2000, le idee e i sistemi che ne sono alla base hanno radici più lontane. Ecco alcuni punti chiave sulla loro nascita:

- **Antecedenti (anni '60 - '90):** Già negli anni '60 e '70 esistevano database che non seguivano il modello relazionale, come i database gerarchici e quelli a rete. Tuttavia, il modello relazionale con SQL divenne dominante. Nel **1998**, il termine **"NoSQL" fu coniato da Carlo Strozzi** per descrivere il suo database open-source che non utilizzava l'interfaccia SQL standard. Questo però non diede il via al movimento come lo conosciamo oggi.
- **La "rinascita" (fine anni 2000):** Con l'esplosione del web, dei big data e delle applicazioni con un alto numero di utenti, i limiti dei database relazionali tradizionali (soprattutto in termini di scalabilità e flessibilità dello schema) divennero più evidenti. Questo portò a una **nuova ondata di sistemi di database non relazionali**.
- **L'evento chiave del 2009:** Un momento cruciale fu un incontro organizzato da Johan Oskarsson di Last.fm nel **2009**. L'obiettivo era discutere i vantaggi dei database non relazionali. Fu in questo contesto che il termine "NoSQL" tornò in auge per etichettare questa crescente famiglia di database che spesso non aderivano alle proprietà ACID (Atomicità, Coerenza, Isolamento, Durabilità) tipiche dei database relazionali, preferendo invece un approccio noto come BASE (Basically Available, Soft state, Eventually consistent).
- **Il Teorema CAP:** Un concetto teorico importante che ha influenzato la progettazione dei database NoSQL è il **Teorema CAP**, formalizzato nel 2002 (basato su una congettura del 1999). Questo teorema afferma che in un sistema distribuito, è impossibile garantire contemporaneamente tutte e tre le seguenti proprietà: **Consistenza**, **Disponibilità** e **Tolleranza alle Partizioni**. I database NoSQL spesso scelgono di privilegiare la Disponibilità e la Tolleranza alle Partizioni rispetto alla Consistenza forte, optando per una consistenza eventuale.
- **I pionieri:** Alcuni dei primi e più influenti database NoSQL che emersero in questo periodo includono:
  - **Key-Value Stores:** Come Amazon Dynamo (le cui idee hanno influenzato molti altri).
  - **Document Databases:** Come MongoDB e CouchDB.
  - **Column-Family Databases:** Come Google Bigtable (e la sua implementazione open-source HBase) e Cassandra (originariamente sviluppato in Facebook).
  - **Graph Databases:** Come Neo4j.

In sintesi, la nascita dei database NoSQL è stata un processo evolutivo, con radici concettuali più antiche, ma con una vera e propria esplosione e definizione come movimento alla fine degli anni 2000 in risposta alle sfide poste dalle nuove esigenze di gestione dei dati nel mondo del web e dei big data. Il termine "NoSQL" è quindi diventato un ombrello per diverse tipologie di database che offrono approcci alternativi al modello relazionale tradizionale.

Tipologie dei database NoSQL



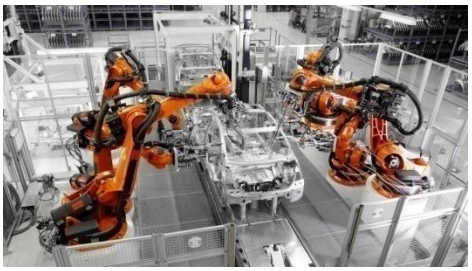
Esistono diverse tipologie di database NoSQL, ognuna ottimizzata per specifici casi d'uso e modelli di dati. Le principali categorie sono:

1. **Database Chiave-Valore (Key-Value Stores):**
   - Memorizzano i dati come coppie chiave-valore, dove ogni chiave è univoca e identifica un valore.
   - Sono altamente scalabili e offrono operazioni di lettura e scrittura molto veloci.
   - **Esempi:** Redis, Memcached, Amazon DynamoDB.
   - **Casi d'uso:** Caching, gestione delle sessioni utente, profili utente semplici.

2. **Database Orientati ai Documenti (Document Databases):**
    - Memorizzano i dati come "documenti", spesso in formato JSON o XML. Ogni documento è auto-descrittivo e può avere una struttura interna variabile.
    - Offrono flessibilità nello schema e sono adatti per dati semi-strutturati.
    - **Esempi:** MongoDB, Couchbase, Amazon DocumentDB.
    - **Casi d'uso:** Cataloghi di prodotti, gestione di contenuti, profili utente complessi.
3. **Database Orientati ai Grafi (Graph Databases):**
    - Memorizzano i dati come nodi (entità) e archi (relazioni) per rappresentare e interrogare dati altamente connessi.
    - Sono ideali per analizzare le relazioni tra i dati.
    - **Esempi:** Neo4j, Amazon Neptune.
    - **Casi d'uso:** Social network, sistemi di raccomandazione, rilevamento frodi, knowledge graph.
4. **Database a Colonne (Wide-Column Stores o Column-Family Databases):**
    - Memorizzano i dati in colonne anziché in righe. Le colonne correlate sono raggruppate in "famiglie di colonne".
    - Sono progettati per gestire enormi volumi di dati e offrono elevate prestazioni per query su specifiche colonne.
    - **Esempi:** Apache Cassandra, Apache HBase.
    - **Casi d'uso:** Serie temporali, big data analytics, archiviazione di dati di sensori.

È importante notare che alcuni database NoSQL sono **multimodello**, il che significa che supportano più di un tipo di modello di dati. Ad esempio, Azure Cosmos DB supporta documenti, chiavi-valore, grafi e colonne.
La scelta del tipo di database NoSQL dipende dalle specifiche esigenze dell'applicazione, dal tipo di dati da gestire e dai requisiti di scalabilità e prestazioni.

Campi di applicazione del database NoSql mongodb





MongoDB, grazie alla sua flessibilità e scalabilità, si insinua in molti aspetti della nostra vita quotidiana, spesso dietro le quinte delle applicazioni e dei servizi che utilizziamo. Ecco alcuni campi di applicazione di MongoDB nella vita di tutti i giorni:

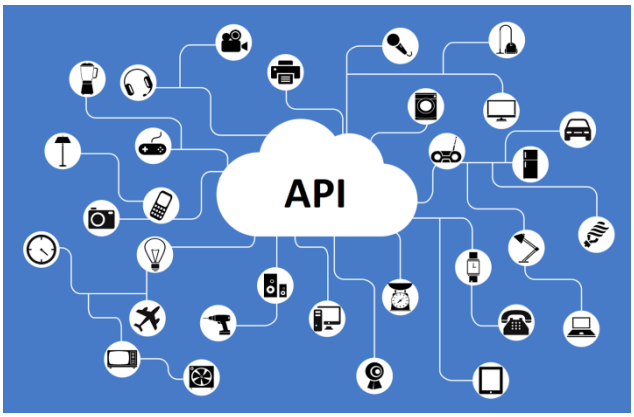
- **Social Media e Networking:** Molte piattaforme social utilizzano MongoDB per gestire i profili degli utenti, i post, i commenti, le connessioni e i feed di notizie. La sua capacità di gestire dati non strutturati e in continua evoluzione è ideale per questo tipo di applicazioni. Immagina di scorrere il tuo feed: le informazioni sui tuoi amici, i loro post con testo, immagini e video, i commenti e i "like" sono spesso memorizzati in database come MongoDB.
- **E-commerce e Vendita al Dettaglio Online:** I negozi online utilizzano MongoDB per cataloghi di prodotti flessibili (con attributi che possono variare da prodotto a prodotto), gestione degli ordini, carrelli degli acquisti temporanei e personalizzazione dell'esperienza utente (come raccomandazioni basate sulla cronologia di navigazione). Quando navighi su un sito di shopping e vedi le descrizioni dettagliate dei prodotti, le recensioni e le immagini, è probabile che queste informazioni siano gestite da un database NoSQL come MongoDB.
- **Gaming Online:** I giochi online con molti giocatori simultanei sfruttano la scalabilità di MongoDB per gestire i profili dei giocatori, le statistiche di gioco, gli inventari degli oggetti e le classifiche in tempo reale. La sua capacità di gestire un elevato volume di scritture e letture con bassa latenza è cruciale per un'esperienza di gioco fluida.

- **Applicazioni Mobili:** Molte app che usiamo quotidianamente, dai social network alle app di produttività, utilizzano MongoDB come backend per archiviare e sincronizzare i dati degli utenti tra diversi dispositivi. Pensa alle app per prendere appunti, alle app di gestione delle attività o alle app di fitness che memorizzano i tuoi progressi.
- **Internet of Things (IoT) e Smart Home:** I dispositivi IoT generano un'enorme quantità di dati in formati diversi. MongoDB è adatto per raccogliere, archiviare e analizzare questi dati provenienti da sensori, termostati intelligenti, telecamere di sicurezza e altri dispositivi connessi. Ad esempio, i dati sulla temperatura della tua casa, il consumo energetico o i video di sorveglianza potrebbero essere memorizzati in un database MongoDB.
- **Servizi di Streaming (Video e Musica):** Le piattaforme di streaming utilizzano database come MongoDB per gestire i metadati dei contenuti (titoli, descrizioni, tag, informazioni sugli artisti), le preferenze degli utenti, la cronologia di visualizzazione e le raccomandazioni personalizzate. Quando Netflix ti suggerisce un nuovo film basandosi su ciò che hai guardato in precedenza, è probabile che un database NoSQL stia lavorando dietro le quinte.
- **Logistica e Trasporti:** Le aziende di logistica utilizzano MongoDB per tracciare le spedizioni, gestire gli inventari in tempo reale e ottimizzare le rotte di consegna. La sua flessibilità è utile per gestire dati eterogenei provenienti da diversi sistemi di tracciamento.
- **Sanità:** In ambito sanitario, MongoDB può essere utilizzato per gestire le cartelle cliniche elettroniche (EHR) con informazioni variabili sui pazienti, i risultati dei test, la storia dei trattamenti e le note dei medici. La sua flessibilità consente di adattarsi alle diverse tipologie di dati medici.

In sintesi, MongoDB è spesso la tecnologia invisibile che supporta molte delle applicazioni e dei servizi digitali che rendono la nostra vita quotidiana più connessa, efficiente e personalizzata. La sua capacità di gestire grandi volumi di dati flessibili e in continua evoluzione lo rende una scelta popolare per un'ampia gamma di applicazioni moderne.

Le applicazioni client- server si compongono di tre parti

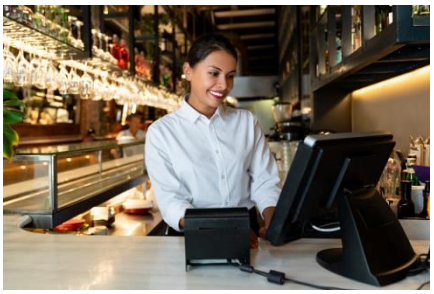
Il SERVER, il CLIENT e l' API

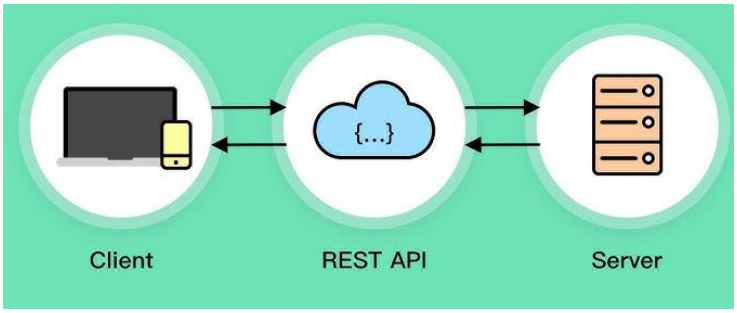


Viviamo in un mondo dove tutti sono interconnessi tra di loro, dove in pochi secondi riusciamo a mandare un messaggio ad una persona distante anche migliaia di kilometri e dove, comodamente dal letto con il nostro smartphone, potremmo anche programmare l'attivazione della macchina del caffè; ma **come avviene questa connessione**? Come fanno diversi dispositivi (tablet, pc, smartphone) e applicazioni a permetterlo?

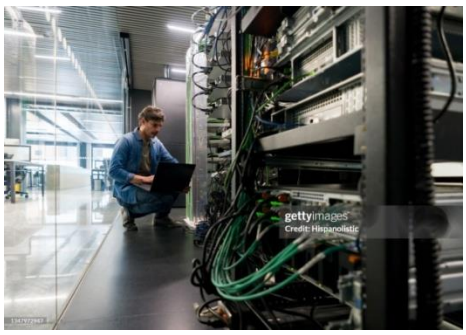
L'eroe silenzioso e poco citato che permette tutto questo sono le **Application Programming Interface** o meglio conosciute come **API.**

Immagina di andare al ristorante e di ordinare del cibo o chiedere informazioni in merito ad un piatto. Sicuramente, per fare una di queste azioni interagirai con un **cameriere,** rivolgendoti a lui anche per chiedere e pagare il conto oppure per altre informazioni sul menù.

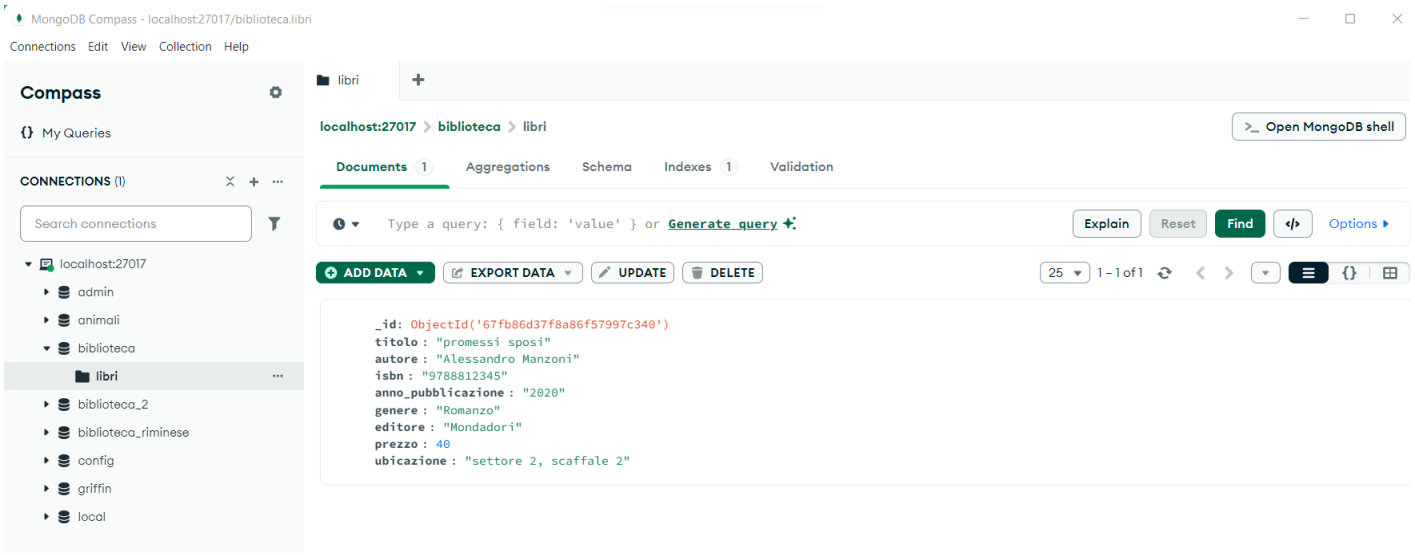




API

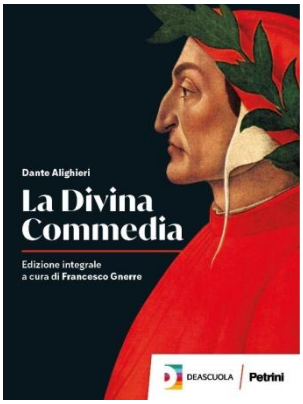


Il client MongoDB Compass



Il nostro programma client Database_Biblioteca realizzato in python con la libreria tkinter

I libri che inseriamo nel database



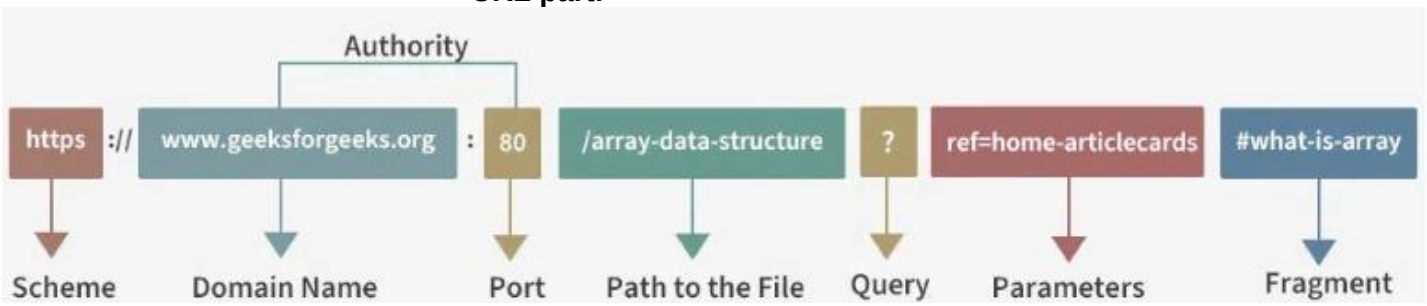

```
_id: ObjectId('67f77b397b9a283d5c1f83bb')
titolo: "promessi sposi"
autore: "alessandro manzoni"
anno_pubblicazione: 2025
genere: "romanzo"
isbn: 97888233799916
editore: "mondadori"
```

```
_id: ObjectId('67f77b397b9a283d5c1f83be')
titolo: "la divina commedia"
autore: "dante aligheri"
anno_pubblicazione: 2024
genere: "romanzo"
isbn: 97888233799917
editore: "deascuola-petrini"
```

URL

URL è l'acronimo di **Uniform Resource Locator**. In parole semplici, è l'**indirizzo web** di una specifica risorsa su Internet. Pensa all'URL come all'indirizzo di casa di una pagina web, un'immagine, un video o qualsiasi altro file online.

URL parti



Un URL indica al tuo browser:

1. **Il protocollo:** Come accedere alla risorsa (ad esempio, http:// o la sua versione sicura https://).
2. **Il dominio:** Il nome del server dove si trova la risorsa (ad esempio, www.google.com).
3. **Il percorso:** La posizione specifica del file o della pagina all'interno di quel server (ad esempio, /search).

A volte un URL può contenere anche:

4. **Una porta:** Un numero che specifica un "canale" di comunicazione specifico sul server (solitamente omesso se si usano le porte predefinite).
5. **Parametri:** Informazioni aggiuntive inviate al server (solitamente dopo un punto interrogativo ?).
6. **Un frammento:** Un riferimento a una specifica parte all'interno della pagina (solitamente dopo un cancelletto #).

Ecco alcuni esempi di URL:

- https://www.wikipedia.org/wiki/Italia: Questo è l'URL della pagina di Wikipedia sull'Italia.
  - https://: Protocollo sicuro per la navigazione web.
  - www.wikipedia.org: Il dominio di Wikipedia.
  - /wiki/Italia: Il percorso specifico della pagina sull'Italia.
- https://www.youtube.com/watch?v=dQw4w9WgXcQ: Questo è l'URL di uno specifico video su YouTube.
  - https://: Protocollo sicuro per la navigazione web.
  - www.youtube.com: Il dominio di YouTube.
  - /watch: Indica che si sta visualizzando un video.
  - ?v=dQw4w9WgXcQ: Un parametro che specifica l'ID univoco del video.
- https://www.amazon.it/gp/product/B0863JW2K7/ref=sxin_15_ac_d_rm?ac_md=0-0-Ym9va3MgZSBsaWJyYWk%3D-ac_d_rm-s-cf-sh&pd_rd_w=n79Y7&pf_rd_p=99a8b11c-4d9c-490a-89b9-1f851903066a&pf_rd_r=W6Y7W8H9J2K1M3N4P5Q6&qid=1678886400&sr=1-1-74d31540-1aa8-4b3f-ba71-90376752cd1c: Questo è un URL più complesso di una pagina prodotto su Amazon. Include vari parametri per tracciare la provenienza e le preferenze.
- mailto:nome.cognome@esempio.com: Questo è un tipo speciale di URL che apre il programma di posta elettronica predefinito dell'utente e indirizza una nuova email all'indirizzo specificato.

Framework

Un **framework** è come uno **scheletro predefinito** per costruire qualcosa di più complesso. Pensa a un **kit di costruzione** per una casa: ti fornisce le fondamenta, i muri portanti, il tetto (grezzo), lasciando a te il compito di decidere la disposizione delle stanze, le finiture e l'arredamento.

Nel mondo del software, un framework offre una struttura di base con librerie di codice, strumenti e linee guida da seguire per sviluppare un'applicazione. Invece di scrivere tutto il codice da zero (come gestire gli utenti, connettersi a un database, creare pagine web), il framework ti fornisce già queste funzionalità, permettendoti di concentrarti sulla logica specifica della tua applicazione.
Esempi di frame work sono Django e Flask

Con Flask, avrai a disposizione gli strumenti essenziali per strutturare la tua applicazione web in modo organizzato, pur mantenendo un maggiore controllo sui componenti che utilizzi.
Ecco come potresti affrontare alcuni aspetti del blog con Flask:

- **Gestione delle rotte (URL):** In Flask, definisci le diverse URL del tuo sito e le funzioni Python che devono essere eseguite quando un utente visita quella specifica URL.

- **Gestione dei template (HTML):** Flask ti permette di utilizzare motori di template (come Jinja2, che è incluso di default) per creare pagine HTML dinamiche. Puoi passare dati dalle tue funzioni Python ai template per visualizzare le informazioni (come l'elenco degli articoli o il contenuto di un singolo articolo).
- **Interazione con il database:** A differenza di Django che ha un ORM integrato, con Flask sei più libero di scegliere la libreria per interagire con il tuo database (ad esempio, SQLAlchemy o semplicemente librerie specifiche per il tuo database come psycopg2 per PostgreSQL o sqlite3 per SQLite). Dovrai scrivere tu il codice per definire le tabelle e interagire con i dati.
- **Gestione degli utenti e dei form:** Flask non fornisce nativamente un sistema completo per la gestione degli utenti o dei form. Tuttavia, puoi facilmente integrare librerie esterne per gestire queste funzionalità, come Flask-Login per l'autenticazione degli utenti e Flask-WTF per la gestione dei form.

In sintesi: Flask è come un **set di attrezzi ben fornito** e una **guida flessibile** per costruire la tua casa (il blog). Ti dà le basi per gestire le richieste web (le rotte), mostrare contenuti (i template) e connetterti ad altre parti del tuo sistema (come il database), ma ti lascia **molta più libertà** nella scelta degli strumenti specifici e nella struttura dettagliata della tua applicazione rispetto a Django.

Template

Un **template**, nel contesto dello sviluppo web e software, è un **modello predefinito** che definisce la **struttura di un documento o di un'interfaccia utente**, separando la sua **logica di presentazione** dai **dati effettivi** che verranno visualizzati.

Un template definisce un layout ed indica dove inserire i contenuti dinamici
Immagina di voler creare diverse pagine web con una struttura simile: un header con il logo e il menu di navigazione, un corpo centrale per il contenuto specifico della pagina e un footer con le informazioni di copyright.
Senza un template, dovresti scrivere l'intero codice HTML per ogni singola pagina, ripetendo la struttura dell'header e del footer ogni volta. Questo sarebbe inefficiente e difficile da mantenere: se volessi cambiare il logo, dovresti modificare ogni singolo file HTML.
Con un template, invece, crei un file (il template) che definisce questa struttura comune. All'interno del template, inserisci dei **segnaposto** o delle **variabili** che verranno poi riempiti con i dati specifici di ogni pagina.

La libreria Flask (creare server web)

Flask è un **micro-framework web** per Python. Immagina di voler costruire un sito web o un'applicazione web. Invece di dover scrivere tutto da zero (gestire le richieste degli utenti, le risposte, ecc.), Flask ti fornisce gli strumenti di base per farlo in modo semplice e veloce.
Ecco i concetti chiave:

- **Routing:** Definisci quali URL del tuo sito web devono attivare specifiche funzioni Python. Ad esempio, quando un utente visita / (la homepage), Flask sa quale funzione eseguire per mostrare il contenuto.

- **Funzioni di visualizzazione (View Functions):** Queste sono le funzioni Python che scrivi tu. Ricevono le richieste dagli utenti (tramite gli URL definiti nel routing) ed elaborano i dati.
- **Template:** Flask ti permette di usare file HTML dinamici (chiamati template) per presentare le informazioni agli utenti. Puoi inserire codice Python all'interno di questi template per mostrare dati in modo dinamico.
- **Richiesta e Risposta:** Flask gestisce le richieste che arrivano dai browser degli utenti e ti aiuta a costruire le risposte da inviare indietro (pagine web, dati JSON, ecc.).
- **Estensioni:** Flask è "micro" perché il suo nucleo è piccolo, ma puoi estenderne le funzionalità con diverse librerie (estensioni) per gestire database, autenticazione, form e molto altro.

In sostanza, Flask ti offre una struttura leggera e flessibile per costruire applicazioni web in Python, dandoti il controllo su come organizzi il tuo codice.

Le funzioni principali

Capiamo come funzionano render_template, request e redirect nella libreria Flask di Python. Sono strumenti fondamentali per costruire applicazioni web dinamiche.

render_template: Il tuo pennello per dipingere l'HTML
Immagina di dover mostrare una pagina web all'utente. Questa pagina spesso contiene informazioni dinamiche, magari il suo nome o una lista di prodotti, render_template è la funzione che ti permette di prendere un file HTML (che chiameremo "template") e di infonderlo con questi dati dinamici provenienti dal tuo codice Python.

In pratica:

1. **Crei un file HTML:** Questo file conterrà la struttura della tua pagina web e dei segnaposto per i dati dinamici. Flask cerca questi file di default in una cartella chiamata templates all'interno della tua applicazione. Ad esempio, potresti avere un file chiamato saluto.html:
```
<!DOCTYPE html>
<html>
<head>
    <title>Saluto!</title>
</head>
<body>
    <h1>Ciao, {{ nome }}!</h1>
    <p>Oggi è il {{ giorno_settimana }}.</p>
</body>
</html>
```
Nota le doppie parentesi graffe {{ ... }}. Queste indicano dove Flask inserirà i dati che gli passerai.

2. **Nel tuo codice Flask, usi render_template:** Quando la tua applicazione Flask riceve una richiesta (ad esempio, un utente visita un certo indirizzo web), la tua funzione di "view" (quella che gestisce quella richiesta) può usare render_template per generare la risposta HTML.
```
from flask import Flask, render_template
from datetime import datetime

app = Flask(__name__)

@app.route('/saluta/<string:nome>')
def saluta(nome):
    oggi = datetime.now()
    giorno = oggi.strftime("%A")
    return render_template('saluto.html', nome=nome, giorno_settimana=giorno)

if __name__ == '__main__':
    app.run(debug=True)
```
In questo esempio:
   - Quando un utente visita l'indirizzo /saluta/Alice, la funzione saluta('Alice') viene eseguita.
   - render_template('saluto.html', nome='Alice', giorno_settimana='Mercoledì') prende il file saluto.html e sostituisce {{ nome }} con 'Alice' e {{ giorno_settimana }} con il giorno corrente.
   - La funzione restituisce l'HTML così generato, che viene poi inviato al browser dell'utente.

request: La finestra sulle richieste dell'utente
Quando un utente interagisce con la tua applicazione web (ad esempio, clicca su un link, compila un form e lo invia), il suo browser invia una "richiesta" al tuo server. L'oggetto request di Flask è come una lente d'ingrandimento che ti permette di esaminare tutti i dettagli di questa richiesta.

Cosa puoi fare con request:

- **Accedere ai dati inviati tramite URL (query parameters):** Se un URL è del tipo /prodotti?categoria=libri&prezzo_max=20, puoi accedere a categoria e prezzo_max.

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/prodotti')
def prodotti():
    categoria = request.args.get('categoria')
    prezzo_massimo = request.args.get('prezzo_max')
    return f"Hai richiesto prodotti della categoria: {categoria} con un prezzo
massimo di: {prezzo_massimo}"

if __name__ == '__main__':
    app.run(debug=True)
```

- **Accedere ai dati inviati tramite form (dati POST):** Quando un utente compila un form e lo invia, i dati vengono inviati nel "corpo" della richiesta.

```
from flask import Flask, request, render_template

app = Flask(__name__)

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        return f"Hai inserito username: {username} e password: {password}"
    return render_template('login.html') # Mostra il form di login

if __name__ == '__main__':
    app.run(debug=True)
```

  In questo caso, se la richiesta è di tipo POST (l'utente ha inviato il form), request.form è un oggetto simile a un dizionario che contiene i dati del form.
- **Ottenere informazioni sull'utente:** Puoi accedere all'indirizzo IP dell'utente (request.remote_addr), all'user agent del suo browser (request.user_agent), ai cookie (request.cookies), ecc.

redirect: Il tuo navigatore virtuale

A volte, dopo che un utente ha compiuto un'azione (ad esempio, si è loggato con successo, ha inviato un form), non vuoi semplicemente mostrargli una nuova pagina, ma vuoi "reindirizzarlo" a un altro URL. La funzione redirect fa esattamente questo. Invia al browser dell'utente una risposta speciale che gli dice di caricare un URL diverso.

Come si usa:

```
from flask import Flask, redirect, url_for

app = Flask(__name__)

@app.route('/successo')
def successo():
    return "Login effettuato con successo!"

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # Simula un controllo delle credenziali
        if request.form['username'] == 'utente' and request.form['password'] ==
'password':
            return redirect(url_for('successo')) # Reindirizza alla pagina di
successo
        else:
            return "Credenziali errate."
    return '''
        <form method="POST">
```

```
            <label for="username">Username:</label><br>
            <input type="text" id="username" name="username"><br>
            <label for="password">Password:</label><br>
            <input type="password" id="password" name="password"><br><br>
            <input type="submit" value="Login">
        </form>
    '''

if __name__ == '__main__':
    app.run(debug=True)
```

In questo esempio:

- Se l'utente inserisce le credenziali corrette, redirect(url_for('successo')) fa sì che il browser dell'utente venga automaticamente reindirizzato all'URL associato alla funzione successo (che in questo caso è /successo).
- url_for('successo') è una funzione utile di Flask che genera l'URL per una determinata funzione di "view" in base al suo nome. Questo rende il tuo codice più robusto, perché se cambi l'URL /successo in futuro, non dovrai modificare tutti i redirect.

In sintesi:

- **render_template**: Prende un file HTML e lo popola con dati dinamici per mostrare una pagina web all'utente.
- **request**: Ti dà accesso a tutti i dettagli della richiesta inviata dal browser dell'utente.

redirect: Invia una risposta al browser dell'utente dicendogli di caricare un URL diverso.