UNICA
UNIVERSITÀ
DEGLI STUDI
DI CAGLIARI

sAIfer Lab
Joint lab on Safety and Security of AI

Salvataggio persistente dei dati

Maura Pintor
maura.pintor@unica.it

Basi di Dati

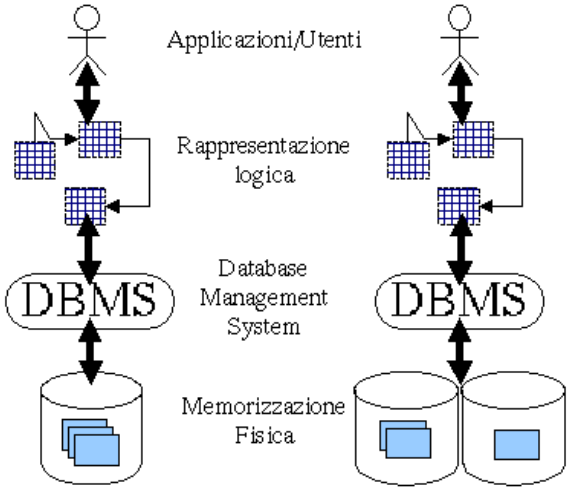
Le basi di dati sono collezioni di dati correlati che rappresentano una realtà

- la dimensione e complessità dipendono dalla realtà da rappresentare
- per esempio, le informazioni sugli acquisti degli utenti possono essere salvati in una opportuna base di dati, la cui complessità dipende dal numero di utenti, oggetti in vendita, e servizi

Sistemi per la gestione delle basi di dati

Attualmente, si utilizzano archivi gestiti da applicativi software

- **DataBase Management System (DBMS):** insieme di programmi per creare e gestire una base dati
  - riservatezza dei dati
  - robustezza a guasti hardware e software
  - manutenibilità (possibilità di evolvere la base di dati)
  - gestione della concorrenza (più utenti contemporanei che modificano i dati in modo consistente)



Descrizione di un sistema di basi di dati

Un DBMS è general-purpose

- Il DBMS fa riferimento ad un catalogo che contiene la definizione della struttura e dei vincoli della base di dati
- Le informazioni contenute nel catalogo sono dette metadati
  - Descrivono i dati di interesse
  - Le applicazioni non necessitano di conoscere la struttura fisica dei dati (es. come e dove sono memorizzati su disco) ma solo la struttura logica (cosa rappresentano)

Basi di dati relazionali

Le basi di dati relazionali sono utili per salvare dati **strutturati** e con **attributi in comune**

È un sistema per organizzare, conservare e accedere ai dati in **forma tabellare**
I dati sono organizzati in tabelle (**relazioni**), composte da righe (**record**) e colonne (**attributi**)

Si compone i seguenti elementi:
- Tabella: una struttura che rappresenta un'entità (es. Utenti, Ordini).
- Riga (Tupla): un'istanza dell'entità (es. un singolo utente).
- Colonna (Attributo): una proprietà dell'entità (es. nome, email).
- Chiave primaria (Primary Key): identifica univocamente ogni riga.
- Chiave esterna (Foreign Key): collegamento ad altre tabelle.

Esempio di base di dati relazionale

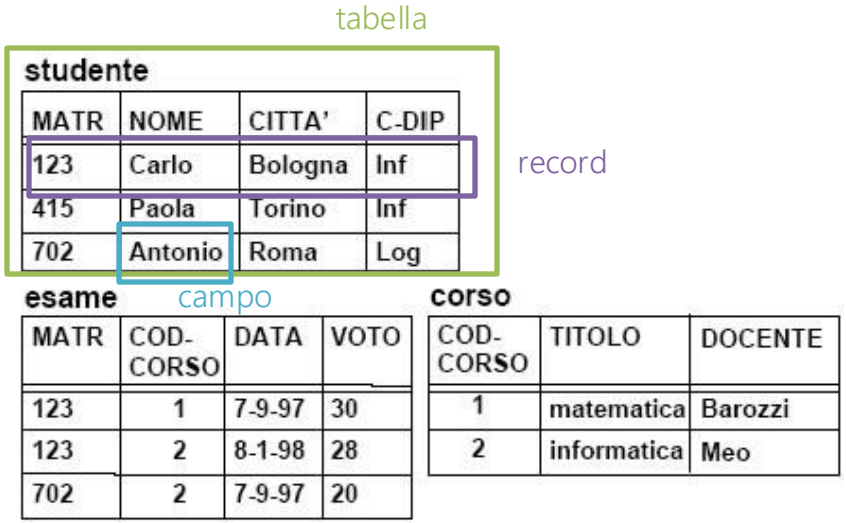
La base di dati è formata da 3 **tabelle**

Ogni tabella contiene un certo numero di **record** fra loro omogenei

Ciascun record è formato da un certo numero di **attributi**

Per ciascun attributo è definito il **tipo di dato**

I record di tabelle diverse sono tra loro **collegati logicam**ente da campi con identico significato (es. il campo *matricola*)



Viste dei dati

La vista è un sottoinsieme della basi di dati che soddisfa le esigenze di un gruppo di utenti

- Può contenere dati derivati dai dati presenti nella base di dati ma non memorizzati in essa
- Si parla di dati virtuali (per esempio, “tutti i nomi degli studenti che seguono il corso di matematica” è una vista che non esiste come tabella a se stante)
- Si possono avere viste diverse per diversi utenti/applicazioni

Query e Structured Query Language (SQL)

SQL è una delle ragioni che hanno determinato il successo commerciale dei DBMS

Structured Query Language

- Istruzioni per la definizione dei dati, formulazione di interrogazioni e aggiornamenti (DDL e DML)
- Sintassi standard usata da tutti i software di basi di dati (con incompatibilità minime)
- **Linguaggio dichiarativo**: descrive cosa si cerca, non come trovarlo

```
SELECT name FROM cities WHERE id = 17;
INSERT INTO countries VALUES ('SLD', 'ENG', 'T', 100.0);
```

La dichiarazione SELECT

La dichiarazione SELECT cerca nel database e restituisce un insieme di risultati

- l'utilizzo della wildcard * seleziona tutto

```
SELECT * FROM cities;
```

La clausola WHERE

Specialmente in basi di dati grandi, è fondamentale filtrare i risultati

La clausola WHERE filtra i risultati (righe) in base ai valori delle colonne

- si possono usare i comuni operatori <, >, =, <=, >=
- <> significa “diverso da”
- si possono creare costrutti logici con AND e OR
- BETWEEN min AND max
- IN (valore1, valore2, …) per selezionare “uno di questi valori possibili”
- LIKE per selezionare dei pattern (per esempio, “inizia con A” diventa “LIKE A%”)

```
SELECT * FROM cities WHERE country = “Italy”;
```

Il prodotto con JOIN

La JOIN genera il prodotto cartesiano, ovvero combina tutte le righe della prima tabella con tutte le righe della seconda tabella

- produce tantissime righe, in numero uguale al numero di righe della prima tabella per il numero di righe della seconda tabella
- per questo motivo è spesso associato a una clausola ON che seleziona le righe che hanno un match specifico
- si può poi aggiungere una clausola WHERE che filtra i risultati

```
SELECT * FROM students JOIN grades ON students.id = grades.student_id;
```

Creazione database e tabelle con CREATE

Crea un intero database nel server DMBS

```
CREATE DATABASE world;
```

Aggiunge una tabella nel database

- bisogna fare la lista completa dei nomi delle colonne e dei tipi di dato
- deve avere una colonna chiave per identificare le singole righe (e le chiavi devono essere univoche)

```
CREATE TABLE cities (
          cityname VARCHAR(30) NOT NULL PRIMARY KEY
          ZIP integer,
          REGION VARCHAR(30)
          country VARCHAR(30)
)
```

Aggiungere dati a tabelle con INSERT

Aggiunge una riga a una data tabella

- I valori delle colonne devono essere messi nello stesso ordine delle colonne della tabella

```
INSERT INTO cities
VALUES (Cagliari, 09100, Sardinia, Italy)
```

REPLACE e UPDATE

La REPLACE è come la INSERT, ma se la riga esiste già (identificata dalla **chiave univoca**), sarà rimpiazzata

```
REPLACE cities
VALUES (Cagliari, 09100, Veneto, Italy)
```

La UPDATE aggiorna la riga solo in determinati valori

- ATTENZIONE: se non si mette la clausola WHERE, aggiornerà tutte le righe

```
UPDATE cities
SET country = “Switzerland”
WHERE region = “Sardinia”
```

Cancellare righe con DELETE

Cancella una riga esistente dalla tabella, in base a un match

- ATTENZIONE: anche in questo caso, se si omette il WHERE, cancellerà tutte le righe

```
DELETE FROM cities
WHERE cityname = “Milano”
```

Create, Read, Update, and Delete (CRUD)

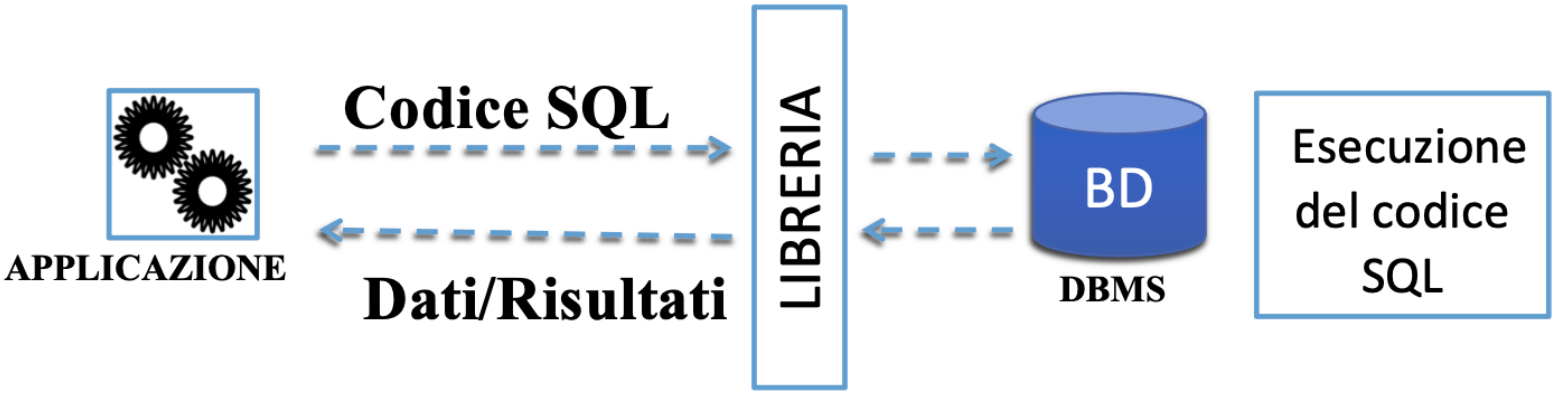
Create: creare righe in una tabella (INSERT)
Read: leggere righe in una tabella (SELECT)
Update: aggiornare righe in una tabella (UPDATE)
Delete: cancellare righe in una tabella (DELETE)

Concetti chiave:
- prima di leggere, bisogna creare
- prima di aggiornare, bisogna creare
- prima di cancellare, bisogna creare
- non si può leggere o aggiornare se si è cancellato

Interazione con un DBMS

Le applicazioni che si interfacciano con un DBMS integrano le query SQL all'interno del loro codice

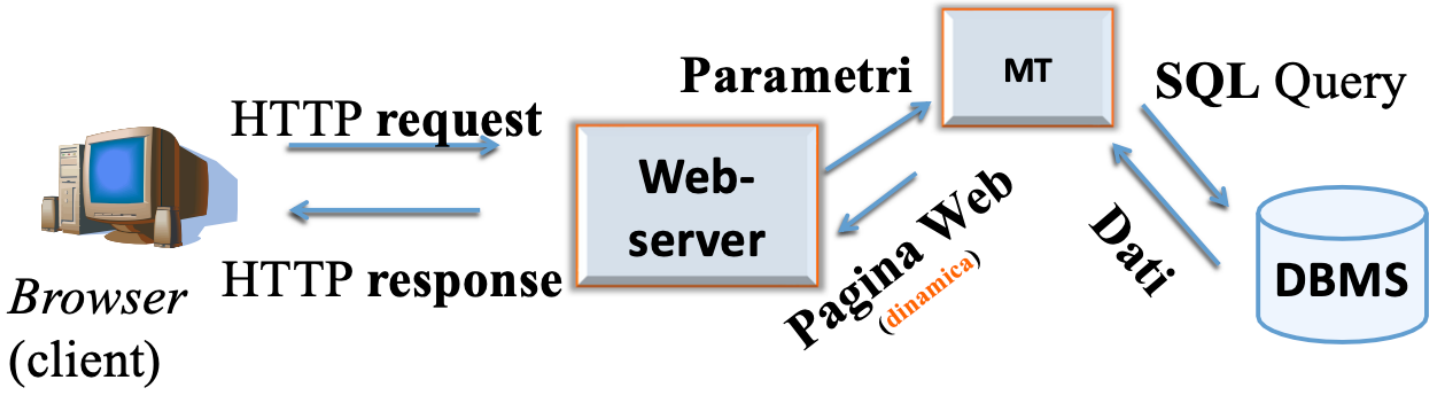
- usano opportune librerie per la connessione e i modelli dei dati



Web Information System (WIS)

Esempio di modello integrato DBMS/App molto usato

- servizio web che si occupa di selezionare i dati da restituire all'utente



Basi di dati non relazionali (cenni)

- “NoSQL” = Not Only SQL
- Famiglia di database che non usano il modello relazionale
- Nati per gestire grandi volumi di dati, dati non strutturati o in rapido cambiamento
- Ideali per sistemi distribuiti, web in tempo reale e Big Data

Tipologie di Database NoSQL

1. **Document Store**
   - Es: MongoDB, CouchDB
   - Dati organizzati in documenti (es. JSON, BSON)
   - Flessibili: ogni documento può avere struttura diversa
2. **Key-Value Store**
   - Es: Redis, DynamoDB
   - Ogni elemento è una coppia chiave-valore
   - Ottimizzati per prestazioni e semplicità
3. **Column-Family Store**
   - Es: Apache Cassandra, HBase
   - Dati memorizzati per colonne invece che per righe
   - Ottimi per letture/scritture massicce
4. **Graph Database**
   1. Es: Neo4j, ArangoDB
   2. Dati rappresentati come nodi e relazioni
   3. Ideali per social network, raccomandazioni, motori semantici

Caratteristiche Chiave dei Database NoSQL

- Schema-less: non richiedono uno schema fisso per i dati
- Scalabilità orizzontale: facili da distribuire su più server
- Alta disponibilità: progettati per tollerare guasti
- Performance: ottimizzati per letture e scritture veloci

Quando usare NoSQL:

- Quando i dati sono non strutturati o variano nel tempo
- In presenza di carichi di lavoro distribuiti o scalabilità elevata
- Per casi d'uso specifici:
  - Caching (Redis)
  - Logging (ElasticSearch)
  - Contenuti flessibili (MongoDB)
  - Relazioni complesse (Neo4j)

Quando usare DB relazionali e quando usare DB non relazionali

Caratteristica	Relazionali (SQL)	Non Relazionali (NoSQL)
Struttura dei dati	Tabelle con schema rigido	Schema flessibile (documenti, key-value, grafo, ecc.)
Integrità dei dati	Alta, grazie a vincoli (PK, FK, CHECK...)	Minore, spesso demandata all'applicazione
Standardizzazione	Linguaggio SQL standard	Linguaggi e query diversi tra motori
Coerenza	Coerenza forte (ACID)	Spesso consistenza eventuale (BASE)
Scalabilità	Scalabilità verticale (potenziare il singolo server)	Scalabilità orizzontale (più nodi distribuiti)
Performance (lettura/scrittura)	Ottime con dati strutturati e relazioni complesse	Ottime con grandi volumi e accessi rapidi
Flessibilità	Poco flessibile: modifiche allo schema complesse	Molto flessibile: adatto a dati non strutturati
Ideale per	Sistemi transazionali, ERP, dati consistenti	Big Data, applicazioni web, caching, real-time

Qualche esercizio su SQL

- https://sqlzoo.net/wiki/SELECT_basics
- https://sqlzoo.net/wiki/SELECT_.._JOIN
- https://sqlzoo.net/wiki/CREATE_TABLE
- https://sqlzoo.net/wiki/INSERT_.._VALUES

Qualche esercisio con NoSQL

Parte 1: Avvio e Verifica

- Avvio di Redis server (se non già attivo):
  - redis-server
- Apertura di una nuova finestra e connessione con la CLI:
  - redis-cli
- Verifica che Redis sia attivo:
  - PING
  - Risposta attesa: PONG

Qualche esercisio con NoSQL

Parte 2: Comandi Base Key-Value

- SET / GET
  - SET nome "Alice"
  - GET nome
- EXISTS
  - EXISTS nome
- DEL
  - DEL nome

Qualche esercisio con NoSQL

Parte 3: Liste (Lists)

- LPUSH compiti "studiare" "fare esercizio" "leggere"
- LRANGE compiti 0 -1
- RPOP compiti

Qualche esercisio con NoSQL

Parte 4: Operazioni Numeriche

- SET contatore 10
- INCR contatore
- DECR contatore
- INCRBY contatore 5

Qualche esercisio con NoSQL

TTL e Scadenze

- SET codice "abc123"
- EXPIRE codice 10
- TTL codice
- Dopo 10 secondi, chiedere di nuovo codice
  - GET codice