



Finanziato  
dall'Unione europea  
NextGenerationEU



# MySQL

## RDBMS



IMMAGINAZIONE  
E LAVORO DAL 1978



Cofinanziato  
dall'Unione europea



REGIONE  
PIEMONTE

# INDICE argomenti

Cos'è un database

DBMS

Il Modello RELAZIONALE

Normalizzazione

Tipi di dato

Installazione MySQL

Accesso DBMS (DBA)

Accesso DBMS (user)

SQL

DDL

1 parte

2 parte

DCL

DML

Query

Integrità referenziale

Union e JOIN

Funzioni

Raggruppamenti

Windows Function (Funzioni finestra)

Viste

Sub Query

Backup/Restoring

Indici

Transaction

Triggers

Stored Function

Stored Procedures

Events

# Cos'è un Database

**Un database è una raccolta organizzata di dati archiviati in un formato facilmente accessibile.**

- I **database** (o basi di dati) sono **collezioni di dati** tra loro correlati, utilizzate per rappresentare una porzione del mondo reale. Sono strutturati in modo tale da consentire la gestione efficiente dei dati, permettendo operazioni come **inserimento**, **aggiornamento**, **ricerca** e **cancellazione** delle informazioni.
- In informatica, il termine database indica una **struttura organizzata di dati**.  
I **database** (o più brevemente, DB) sono archivi dove le applicazioni **memorizzano i dati** in modo **persistente**<sup>1</sup> per poterli successivamente leggere, modificare o eliminare.

<sup>1)</sup> *In ambito informatico, la persistenza si riferisce alla capacità di un dato di essere conservato oltre la durata di esecuzione di un singolo programma, garantendo che i dati siano ancora disponibili in seguito.*

# Dati e Informazione

- **Informazione:** dato elaborato con significato e contesto.

In senso generale, l'informazione è qualsiasi contenuto significativo che possiamo trasmettere, raccogliere o interpretare. Tuttavia, perché l'informazione diventi utile, deve essere organizzata e strutturata in una forma che ne consenta una facile elaborazione.

- **Dati:** rappresentazioni grezze di fatti, numeri o eventi.

Un dato è un'unità elementare di informazione, che può essere un numero, una parola, un'immagine, ecc.

Ad esempio, una stringa di caratteri come "Maria", oppure un numero come "30", sono dati grezzi. Da soli, però, non raccontano molto.

Per renderli significativi dobbiamo inserirli in un contesto strutturato, ad esempio: "Maria ha 30 anni."

# Database file-server, client-server

## database file-server

Sono semplici files, a cui possono facilmente accedere i programmi che li usano per inserire, visualizzare, modificare o cancellare i dati in essi contenuti.

- ◉ il sistema accede fisicamente al file;
- ◉ più il file è di grandi dimensioni maggiore il tempo di accesso;
- ◉ accesso contemporaneo da più utenti rallenta notevolmente il db;
  - *MS Access,*
  - *Filemaker*
  - ...;

## database client-server

Rappresentano un servizio che mette a disposizione il software per interagire con i dati.

Viene gestito e mantenuto dai DBA (Database Administrator).

- *Microsoft SQL Server (RDBMS),*
- *Oracle (RDBMS),*
- *MySQL (RDBMS),*
- *DB2 (RDBMS),*
- *PostgreSql (ORDBMS object-relational database system),*
- *MongoDB (NoSQL Document Stores),*
- *Neo4j (NoSQL Graph),*
- ...;

# client-server

Esempio di richiesta dati attraverso un form via https

## CLIENT



*html  
form*

## WEBSERVER



*Php  
Pyton  
Java*

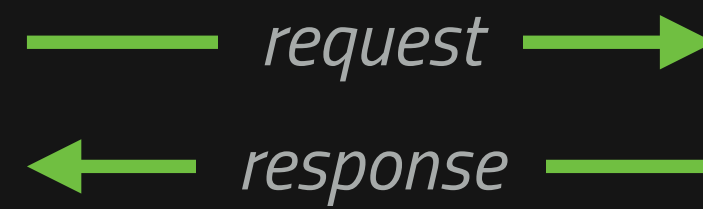
*...*

## DATABASE

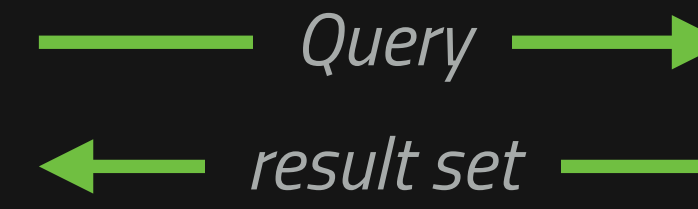


*MySQL  
Dati*

◀ HTTP ▶



◀ SQL ▶



## VANTAGGI DATABASE CLIENT-SERVER

1. I clients **non accedono fisicamente al file** sul database, inviano solamente la loro query al motore del database ed il server restituisce solamente i dati richiesti.
2. **Velocità**: al crescere delle dimensioni del database il tempo di una query rimane identico, perché attraverso la LAN viaggiano e continueranno a viaggiare solamente la richiesta (query) ed i dati restituiti, la dimensione del database diventa alla fine irrilevante per il client.
3. Il motore del database è in grado di gestire tutte le **connessioni simultanee** da parte degli utenti, ed utilizzare al meglio le prestazioni dell'hardware.
4. **Sicurezza**. Se su un sistema file-server potrebbe succedere che in determinate situazioni il file arrivi ad essere corrotto (termine tecnico), questo non deve potere succedere, mai e per nessuna ragione, su un sistema client-server.
5. La sicurezza *viene garantita anche grazie alle funzioni che i db client-server normalmente offrono*. Tutte le tabelle di un sistema gestionale aziendale sono tra loro collegate, la mancanza della gestione delle relazioni può portare a grossi problemi circa l' **integrità dei dati**.



# Strutture Dati e Modelli di Database

Le strutture dati rappresentano il modo in cui le informazioni vengono organizzate e memorizzate per facilitarne il recupero e la gestione.

Ogni struttura ha caratteristiche specifiche che la rendono più o meno adatta a determinati scenari applicativi.

Nei database, le strutture dati vengono implementate attraverso diversi modelli di organizzazione, ognuno pensato per esigenze specifiche.



# Modelli di database più diffusi

## Relazionale (RDBMS - Relational Database Management System)

- ◉ Organizza i dati in tabelle (righe e colonne).
- ◉ Relazioni tra le tabelle tramite chiavi primarie ed esterne.

In questo modello, i dati sono organizzati in tabelle (o relazioni), che possono essere pensate come fogli di calcolo.

Ogni tabella ha delle colonne (campi) e delle righe (record), dove le righe rappresentano entità specifiche e le colonne descrivono attributi delle entità.

Esempi: *MySQL, PostgreSQL, Oracle, SQL Server...*

## Time Series Database

Le serie temporali sono una struttura in cui i dati vengono organizzati in base a un intervallo di tempo. Vengono utilizzate per tracciare cambiamenti e osservare tendenze nel tempo.

- ◉ Ottimizzato per la memorizzazione di dati temporali.
- ◉ Formato: Timestamp + Valore.

Esempi: *InfluxDB*, *TimescaleDB*, *OpenTSDB*.

Esempio: sensori di temperatura che registrano la temperatura ogni ora:

```
{  
  timestamp: "2025-02-21T12:00:00Z",  
  temperatura: 22.5  
}
```

# Database Colonnare

- Memorizza i dati per colonne anziché per righe.
- Vantaggioso per analisi su grandi volumi di dati.

Esempi: *Apache Cassandra, Google Bigtable, Amazon Redshift.*

Esempio: Analisi di vendite con colonne separate per ogni mese:

*Prodotto | Gennaio | Febbraio | Marzo*

## NoSQL (Not Only SQL)

- ◉ Pensato per scalabilità e flessibilità.
- ◉ Tipologie:
  - ◉ Document-oriented: Archivia dati in documenti JSON o BSON (es. *MongoDB*, *CouchDB*).
  - ◉ Key-Value Store: Struttura chiave-valore (es. *Redis*, *DynamoDB*).
  - ◉ Wide-Column Store: Variante del database colonnare (es. *Apache HBase*, *ScyllaDB*).
  - ◉ Graph Database: Gestisce dati con nodi e connessioni (es. *Neo4j*).

Esempio: database documentale MongoDB che memorizza profili utente:

```
{  
  "nome": "Mario",  
  "cognome": "Rossi",  
  "email": "mario.rossi@example.com"  
}
```

# DATABASE

uso dei database relazionali

# Strumenti

- ◉ MySQL Community Server 8.4
- ◉ Terminale/Shell
  - ambiente a riga di comando per interagire con il database tramite istruzioni SQL
- ◉ MySQLWorkbench: ambiente grafico per interagire con il database
- ◉ Visual Studio Code o altro editor (Sublime Text, Atom, Brackets...)
- ◉ Documentazione ufficiale MySQL:
  - <https://dev.mysql.com/doc/refman/8.4/en>

# DBMS

database management system

- ◉ È il software per la creazione e la manipolazione di un database.  
È un **software di tipo server** (client-server) avente il compito di gestire uno o più database; questo vuol dire che il DBMS deve intervenire, **in qualità di intermediario**, in ogni operazione svolta sui database dai software che ne fanno utilizzo.
- ◉ Definisce gli utenti e gli amministratori di un database
- ◉ Fornisce **meccanismi di sicurezza, protezione** e controllo dell'**integrità dei dati**



# Cosa fa un DBMS? (riepilogo)

Funzione	Descrizione
Gestione dati	Crea, legge, modifica, elimina
Sicurezza	Controllo accessi, permessi
Integrità	Regole sui dati, relazioni
Concorrenza	Accessi simultanei
Persistenza	Salvataggio stabile nel tempo
Architettura	Funziona come server in ambienti client-server

Il DBMS è l'intermediario tra i programmi e i dati, garantendo coerenza, sicurezza e prestazioni.

# Componenti principali di un DBMS

Motore di archiviazione → gestisce come i dati sono salvati (es. InnoDB, MyISAM)

Motore delle query → interpreta ed esegue le istruzioni SQL

Gestore delle transazioni → garantisce che i dati siano consistenti

Controllo degli accessi → gestisce utenti, ruoli e permessi

Catalogo (metadati) → contiene le informazioni sulla struttura del database

# RDBMS

relational database management system

- MySQL è un software appartenente alla famiglia dei DBMS. All'interno di questo gruppo di software è possibile identificare dei sotto-insiemi più specifici tra cui, ad esempio, quello dei **DBMS NoSQL** (MongoDB) e quello dei **RDBMS** a cui appartiene tra gli altri, appunto, MySQL.
- Gli RDBMS non sono altro che dei sistemi di gestione delle banche dati che operano in aderenza alla teoria relazionale secondo la quale il sistema deve operare sui dati mediante relazioni tra le diverse tabelle in cui questi vengono suddivisi e ordinati.
- Nel modello relazionale, infatti, i dati all'interno di un database *sono organizzati in differenti tabelle le quali sono in relazione tra loro.*

# Storage Engine

Gli **storage engine** rappresentano delle librerie che determinano il modo in cui i dati di una tabella saranno salvati su disco.

Ciò sarà determinante per valutare le prestazioni, l'affidabilità, le funzionalità offerte dalla tabella stessa, rendendola più o meno adatta a particolari utilizzi.

In pratica, scegliere un particolare storage engine significa scegliere il modo in cui i dati vengono gestiti.

## MyISAM

Si tratta di un motore di memorizzazione veloce. Non supporta le transazioni. Non utilizza meccanismi di integrità referenziale.

- ◉ Adatto per le **ricerche full-text**;
- ◉ È **più veloce** poiché non è necessario tenere conto delle varie relazioni tra le tabelle;
- ◉ esegue il **lock sull'intera tabella**;
- ◉ ottimo se le tabelle vengono utilizzate principalmente in fase di lettura oppure se il **database è relativamente poco complesso**.

A partire dalla versione 5.5 di MySQL, *InnoDB è lo Storage Engine di default*. Prima era MyISAM.

## InnoDB

Lo scopo di InnoDB è quello di associare maggiore sicurezza (intesa soprattutto come consistenza ed integrità dei dati) a performance elevate. Funzionalità peculiari:

- ◉ **Transizioni**: per transazione si intende la possibilità di un DBMS di svolgere più operazioni di modifica dei dati, facendo sì che i risultati diventino persistenti nel database solo in caso di successo di ogni singola operazione. In caso contrario, verranno annullate tutte le modifiche apportate;
- ◉ **Integrità referenziale**: conferiscono la possibilità di creare una relazione logica tra i dati di due tabelle, in modo da impedire modifiche all'una che renderebbero inconsistenti i dati dell'altra;
- ◉ esegue il **lock a livello di riga**;
- ◉ *Ricerche full-text* a partire da MySQL 5.6.

# Charset

I **character set** (insiemi di caratteri) sono i diversi sistemi attraverso i quali i caratteri alfanumerici, i segni di punteggiatura e tutti i simboli rappresentabili su un computer vengono memorizzati in un valore binario. In ogni set di caratteri, ad un valore binario corrisponde un carattere ben preciso.

Con MySQL, a partire dalla versione 4.1, possiamo gestire i set di caratteri a livello di server, database, tabella e singola colonna, nonché di client e di connessione.

Ad ogni set di caratteri sono associate una o più **collation**, che rappresentano i modi possibili di confrontare le stringhe di caratteri facenti parte di quel character set.

Esempio: una determinata tabella utilizza il *utf8* (quello maggiormente usato in Europa Occidentale) e la *utf8\_general\_ci*.

Tale collation è multilingue e "case insensitive" (*\_ci*), cioè non tiene conto della differenza fra maiuscole e minuscole nell'ordinare o confrontare le stringhe.



Supponiamo di avere un alfabeto di quattro lettere:

A, B, a, b.

Assegniamo ad ogni lettera un numero:  $A = 0$ ,  $B = 1$ ,  $a = 2$ ,  $b = 3$ .

La lettera A è un *simbolo*, il numero 0 è la *codifica* per A, e la combinazione di tutte e quattro le lettere e la loro codifica è il **character set**.

Supponiamo che vogliamo confrontare due valori di stringa, A e B. Il modo più semplice per farlo è quello di guardare le codifiche: 0 per A e 1 per B.

Poiché 0 è minore di 1, diciamo A è inferiore a B.

Quello che abbiamo appena fatto è applicare un metodo di confronto per il nostro set di caratteri.

La **collation** è un insieme di regole (una sola regola in questo caso): "Confronta le codifiche".

# IL MODELLO RELAZIONALE

- La **tabella** è la struttura dati fondamentale di un database relazionale;
- Con le tabelle si rappresentano le **entità** e le **relazioni** del modello concettuale\*;
- La tabella è composta da campi (colonne o **attributi**) e da record (righe o **tuple**);
- Ogni *campo* rappresenta un **attributo** dell'entità/ relazione;
- Per ogni campo viene individuato un suo **dominio** (*tipo di dati*): alfanumerico, numerico, data...
- Ogni *record* rappresenta una *istanza* (o occorrenza o *tupla*) dell'entità/relazione;
- Garantisce l'**integrità referenziale**;

\***Modello concettuale**: trasformazione di **specifiche** in linguaggio naturale (che definiscono la realtà descritta dal DB) in uno schema grafico chiamato **Diagramma E-R** che utilizza due concetti fondamentali: *Entità* e *Associazione/Relazione*.

# Progettazione

## Modello Concettuale

**Definizione:** rappresenta i dati e le loro relazioni a un livello astratto, senza preoccuparsi di dettagli tecnici o implementativi.

**Obiettivo:** descrivere la struttura dei dati in modo comprensibile per utenti e analisti, senza vincoli tecnologici.

**Strumento tipico:** diagrammi Entity-Relationship (ER) o UML class diagrams (Unified modeling language).

**Indipendenza dal DBMS:** Questo modello non dipende dal database relazionale, NoSQL o altro.

## Modello Logico

**Definizione:** traduce il modello concettuale in una struttura più dettagliata e aderente alle regole di un particolare tipo di database (es. relazionale).

**Obiettivo:** definire tabelle, attributi, chiavi primarie e chiavi esterne, mantenendo ancora un livello di astrazione dalla tecnologia specifica.

**Strumento tipico:** Schema relazionale (tabelle, colonne, vincoli).

**Dipendenza dal tipo di database:** nel caso del modello relazionale, si definiscono relazioni tra tabelle con chiavi primarie e chiavi esterne.

## Modello Fisico

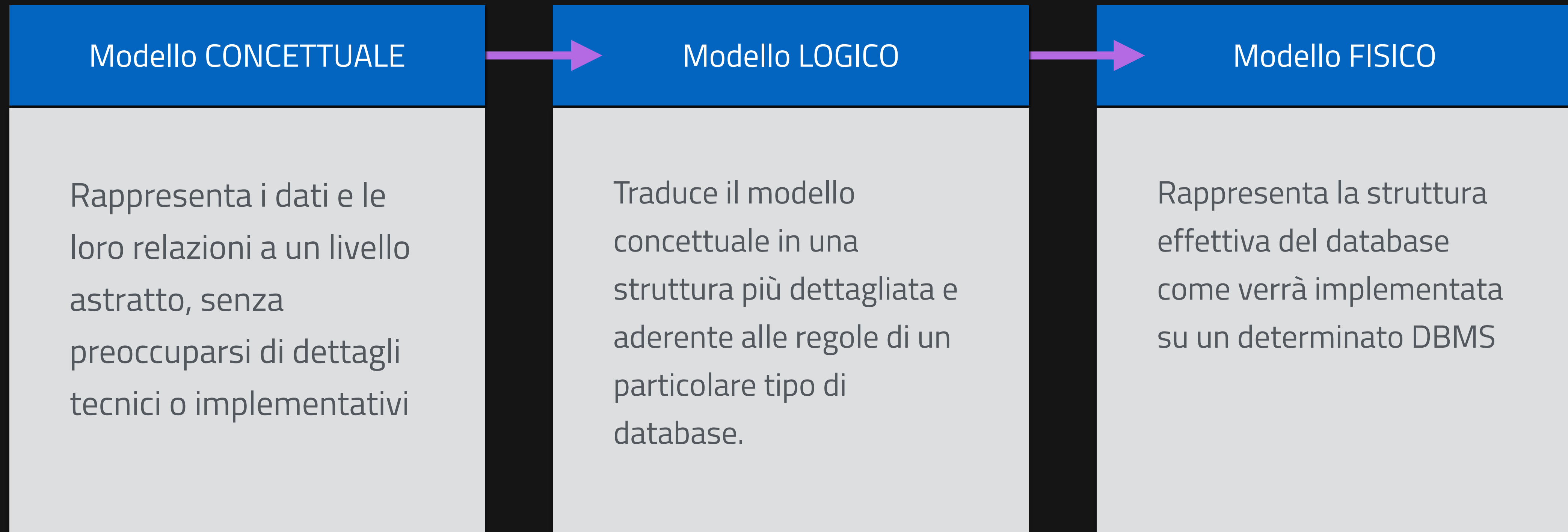
**Definizione:** rappresenta la struttura effettiva del database come verrà implementata su un determinato DBMS.

**Obiettivo:** ottimizzare il database per le prestazioni, definendo dettagli come tipi di dati specifici, indici, partizionamento, ecc.

**Strumento tipico:** DDL (Data Definition Language) SQL per creare tabelle, vincoli, indici.

**Dipendenza dal DBMS:** strettamente legato a un DBMS specifico (es. MySQL, PostgreSQL, Oracle).

# Fasi della progettazione



# Diagramma E-R, simboli

## Entità

Concetto fondamentale, generale, per la realtà che si sta modellando.

Rappresenta *classi di oggetti* (fatti, cose, persone, ...) che hanno *proprietà comuni* ed *esistenza autonoma* ai fini dell'applicazione di interesse.

- ◉ Identificata da un rettangolo



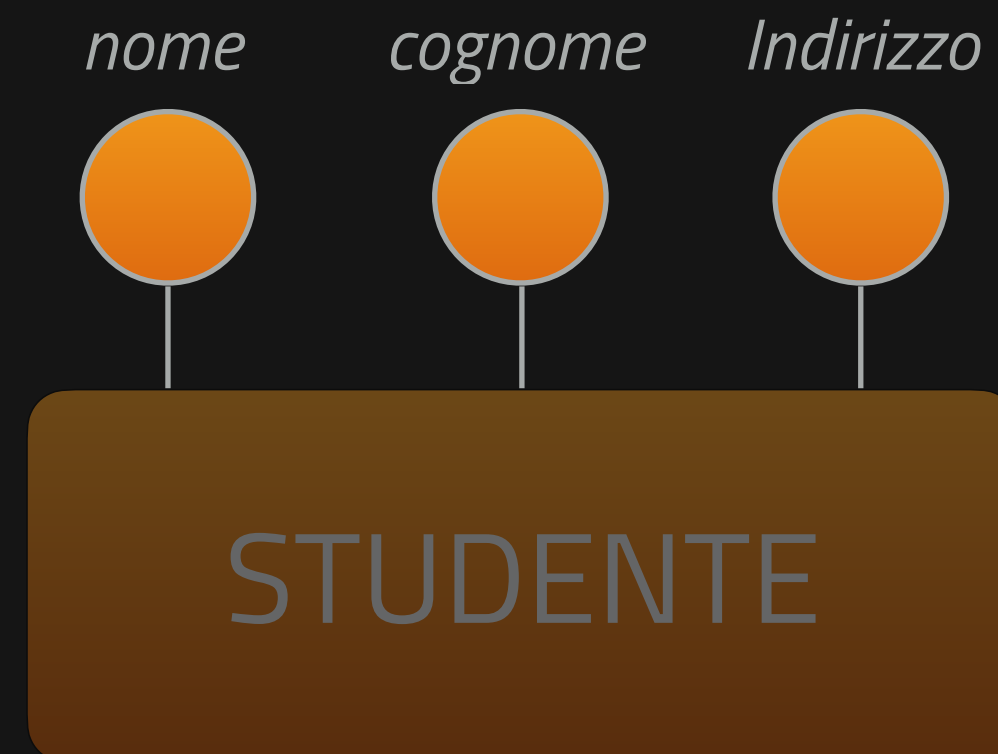
STUDENTE



## Attributi:

*Caratteristiche specifiche di un'entità*, utili (o necessarie) nella realtà da modellare

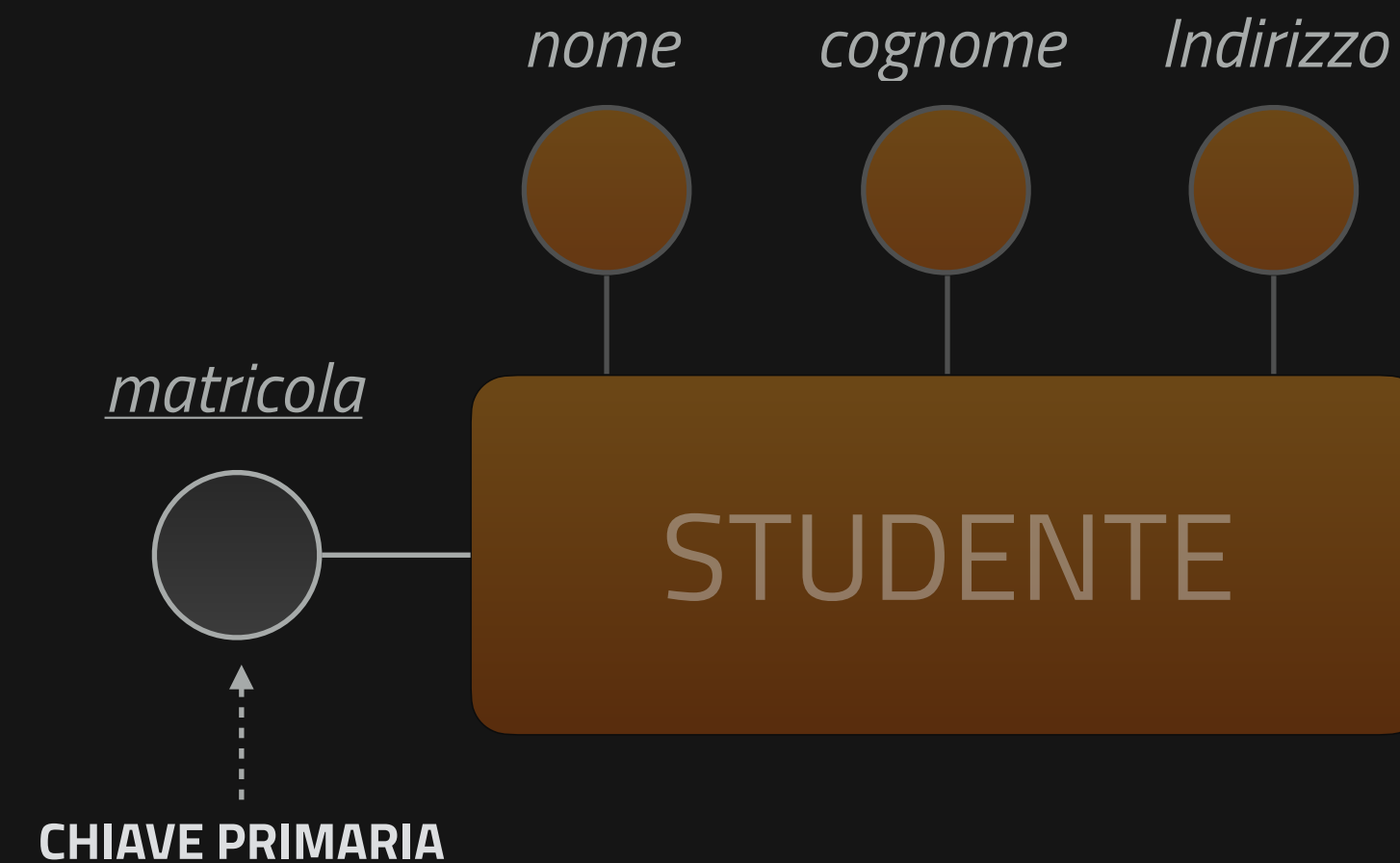
- identificata da un cerchio collegato all'entità



L'insieme di attributi che garantisce **l'univocità** delle istanze di un'entità è detta: **Chiave Primaria**.

È indicata come: **PRIMARY KEY** o **PK**

Identificata graficamente con un *cerchio pieno* collegato all'entità e relativo *nome attributo sottolineato*



## Caratteristiche

- L'insieme dei campi i cui valori identificano univocamente un record all'interno di una tabella è detto Chiave Primaria. Quando la *chiave primaria* è composta da un solo campo, si parla di *campo chiave*.
- Quando non è possibile trovare un campo chiave tra gli attributi di una entità, si definisce un campo univoco di tipo numerico che si auto-incrementa (contatore): ID (identifier).

Esempi di campo chiave: *matricola*, *codice fiscale*, etc.

## Istanze di un'Entità

Specifici dati, oggetti appartenenti ad un'entità

- non sono rappresentate nel Diagramma E-R ma si intendono contenute in ogni entità:
- Carlo Rossi, via Verdi* è un'*istanza* dell'**entità** *ALUNNO* (**attributi**: *Nome, Cognome, Indirizzo*)



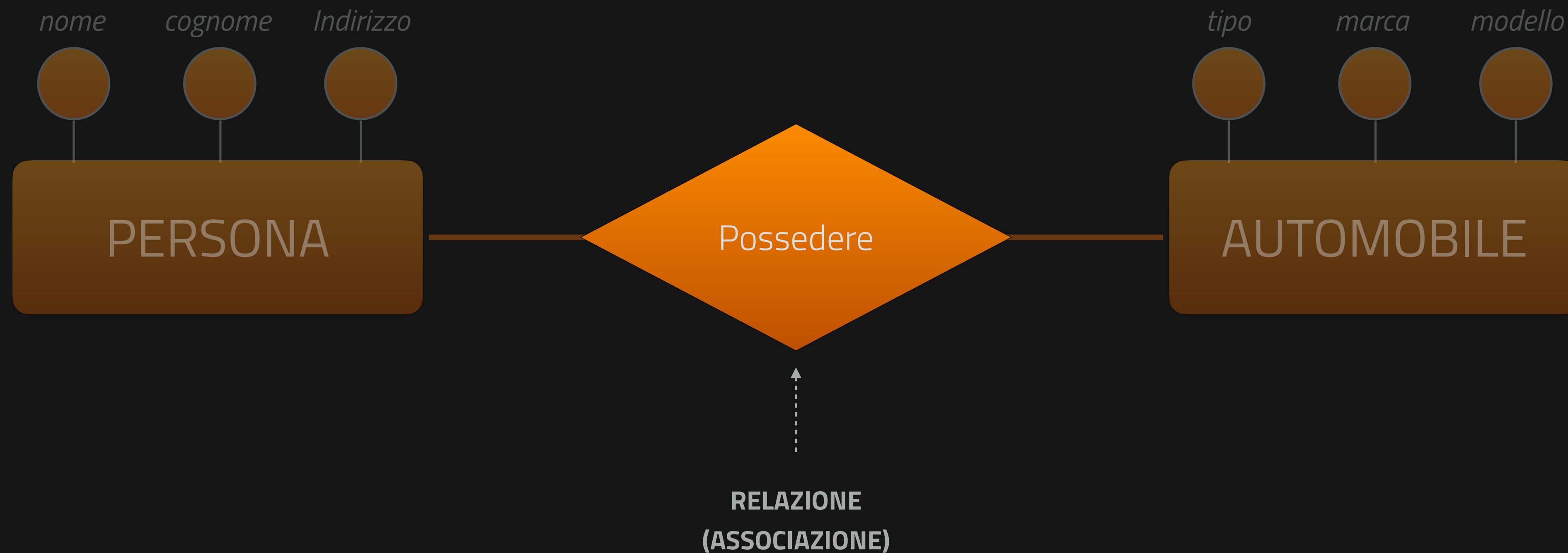
Possiamo considerare le entità come insiemi all'interno dei quali sono contenuti *oggetti* (le istanze) ciascuno con specifiche *caratteristiche* (valore degli attributi).



## Relazioni (Associazioni)

Collegamenti logici che uniscono due o più entità nella realtà descritta dal database

- identificata da un rombo collegato alle due entità:



# Cinema/teatro

## Relazione 1,1 (uno a uno)



- Uno spettatore *occupa* un singolo posto
- Ogni singolo posto può *essere occupato* solo da uno spettatore

## Liceo/Scuola superiore

### Relazione 1,N (uno a molti)



- Ad ogni classe *appartiene* più di un alunno
- Un alunno *appartiene* ad una singola classe

# Università

## Relazione N,N (molti a molti)



- Uno studente *frequenta* più corsi
- Ogni corso è *frequentato* da molti studenti



# Cardinalità delle relazioni

La relazione  $R$  che lega due entità **E1** ed **E2** può essere classificata in base alla sua **cardinalità** (quante istanze delle due entità sono coinvolte nella relazione):

- ◉  $1,1$  (**uno a uno**) se ad un elemento di **E1** può corrispondere un solo elemento di **E2**
- ◉  $1,N$  (**uno a molti**) se ad un elemento di **E1** possono corrispondere più di un elemento di **E2**, ad un elemento di **E2** può corrispondere un solo elemento di **E1**
- ◉  $N,N$  (**molti a molti**) se ad ogni elemento di **E1** possono corrispondere molti elementi di **E2** e viceversa

# Esempi



# Esempio

## **Database per una applicazione web che gestisce l'acquisto/iscrizione dei/ai corsi.**

Disegnare una base dati per la gestione dell'acquisto di corsi offerti da una piattaforma web.

Gli utenti/studenti devono essere registrati sulla piattaforma, per registrarsi occorre nome, cognome ed email. Viene memorizzata anche la data di registrazione.

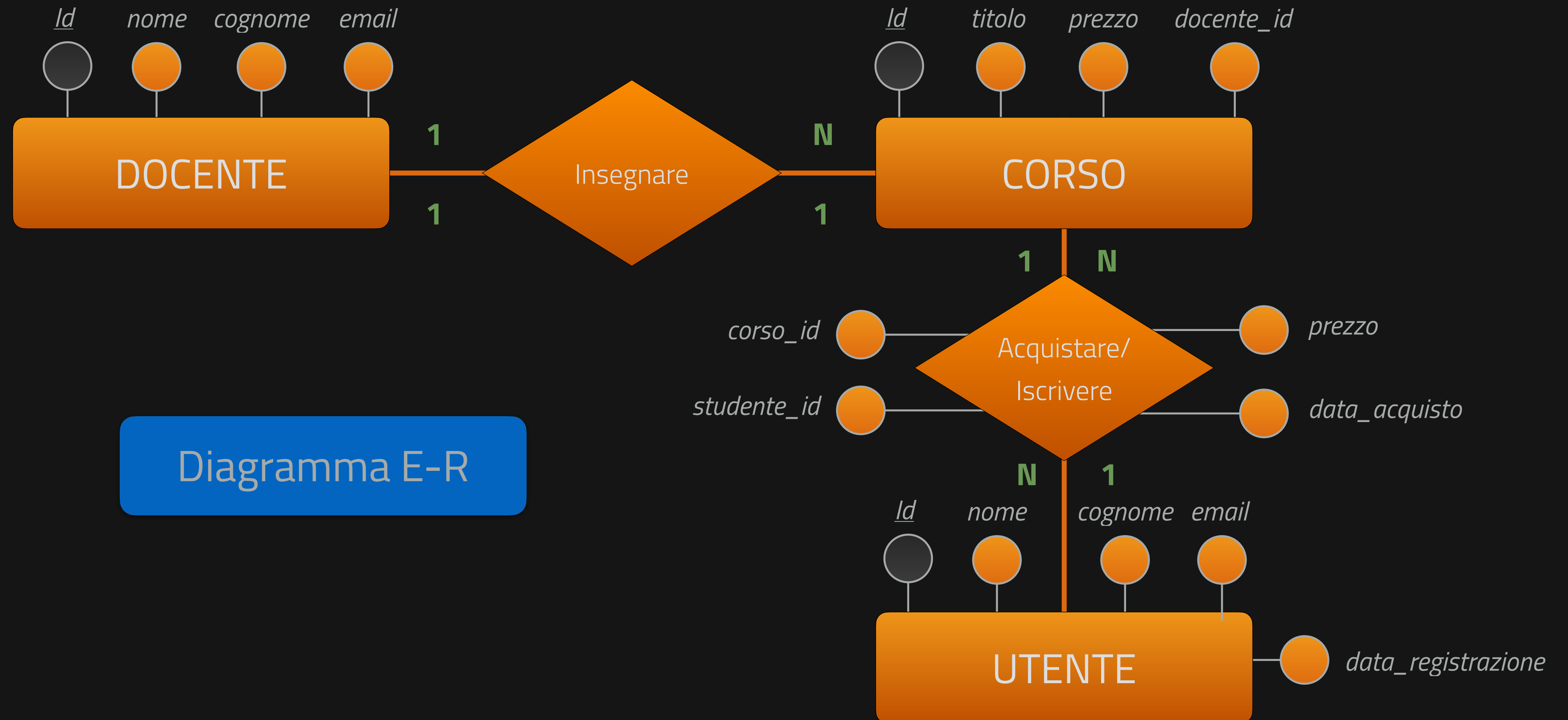
Gli studenti/utenti possono acquistare molti corsi. I corsi sono quindi a pagamento. Viene memorizzata anche la data di acquisto.

Il prezzo del corso può variare nel tempo.

I corsi si riferiscono a svariate materie quali: Java, Base di programmazione, Html, CSS, CMS, Javascript, React, PHP...

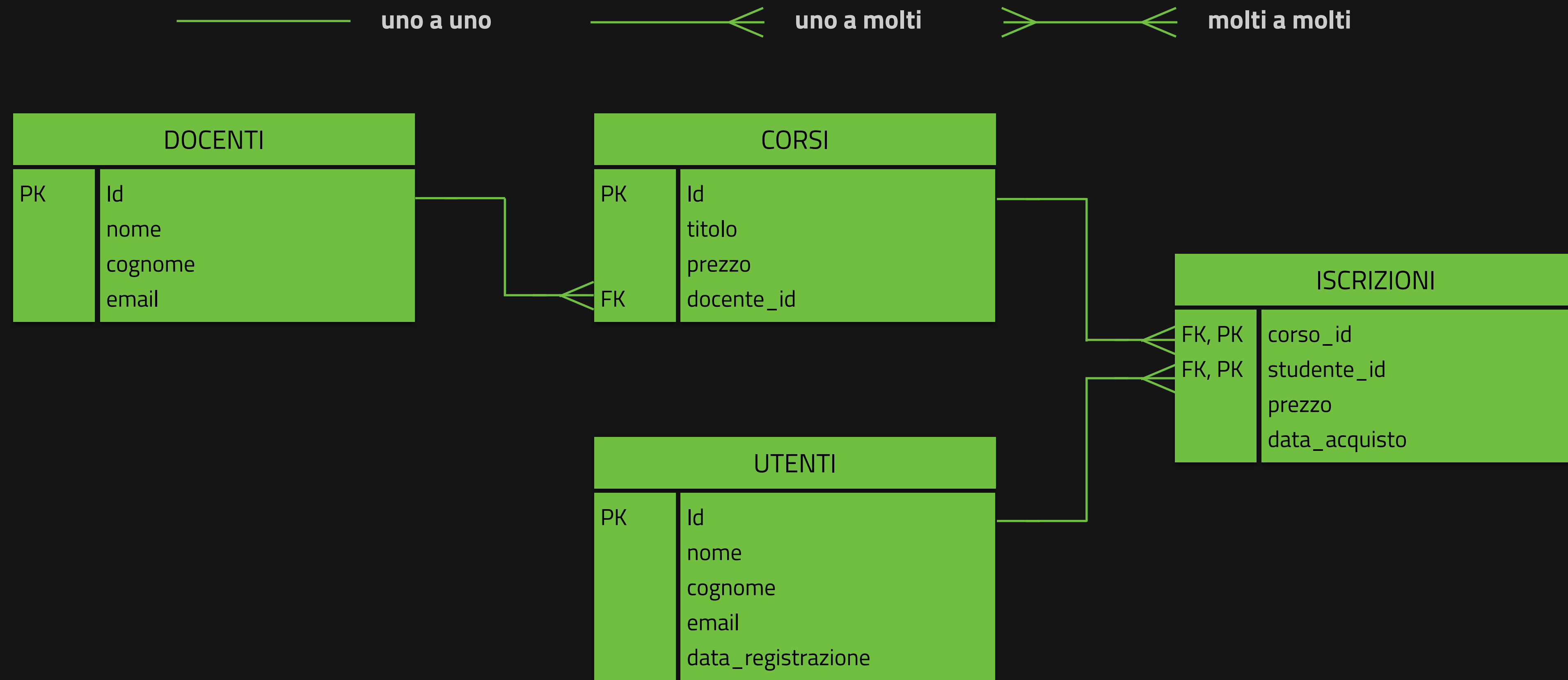
Ogni corso viene tenuto da un docente. Un docente può insegnare in molti corsi.

# Modello Concettuale



# Modello logico

## Simbologia delle relazioni



Rappresentazione dell'informazione in tabelle nel database secondo il modello relazionale dell'esempio relativo alla piattaforma corsi:

UTENTI

● id	cognome	nome	email
1	Esposito	Giuseppe	esposito.g@icloud.com
2	Pautasso	Carlo	cpautasso@gmail.com
3	Piras	Efisio	efpiras@gmail.com
4	Conti	Renzo	conti.renzo@icloud.com
5	Rigon	Paolo	prigon@gmail.com

ISCRIZIONI

corso_id	studente_id	prezzo	data
1	1	180.00	22/9/2022
1	2	180.00	23/9/2022
3	1	300.00	25/9/2022
2	3	120.00	28/9/2022

DOCENTI

● id	cognome	nome	email
1	Rossi	Mario	rossi.mario@icloud.com
2	Verdi	Paola	verdi.paola@gmail.com
3	Bruni	Marco	bruni.marco@gmail.com
4	Bianchi	Daniele	bianchi.daniele@icloud.com

CORSI

● id	titolo	prezzo	docente_id
1	Php	200.00	4
2	Javascript	120.00	2
3	Java	300.00	4

Legame logico tra tabelle

CORSI				DOCENTI			
●	titolo	prezzo	docente_i	● id	cognome	nome	email
1	Php	200.00	4	1	Rossi	Mario	rossi.mario@icloud.com
2	Javascript	120.00	2	2	Verdi	Paola	verdi.paola@gmail.com
3	Java	300.00	4	3	Bruni	Marco	bruni.marco@gmail.com
				4	Bianchi	Daniele	bianchi.daniele@icloud.com

Join tra tabelle

● id	titolo	prezzo	docente_id	● id	cognome	nome	email
1	Php	200.00	4	4	Bianchi	Daniele	bianchi.daniele@icloud.com
2	Javascript	120.00	2	2	Verdi	Paola	verdi.paola@gmail.com
3	Java	300.00	4	4	Bianchi	Daniele	bianchi.daniele@icloud.com

# Normalizzazione del database

Regole da rispettare nel definire le tabelle

## La prima forma normale (1NF)

Si dice che una database è in 1NF (prima forma normale) se per ogni tabella/relazione contenuta nella base dati:

- ◉ *tutte le tuple della relazione hanno lo stesso numero di attributi*
- ◉ *non presenta gruppi di attributi che si ripetono* (ossia ciascun attributo è definito su un dominio con valori atomici)
- ◉ *tutti i valori di un attributo sono dello stesso tipo* (appartengono allo stesso dominio)
- ◉ *esiste una chiave primaria* (ossia esiste un attributo o un insieme di attributi che identificano in modo univoco ogni tupla della relazione)
- ◉ *l'ordine delle righe è irrilevante* (non è portatore di informazioni)



Facciamo un esempio di una tabella che, seppur munita di una chiave primaria, non può essere considerata in forma normale:

● Codice Fiscale	Nome	Dettagli
LBRRSS79Y12T344A	Alberto	età: 30; professione: Impiegato
GNNBNCT84A11L61B	Gianni	età: 24; professione: Studente

La tabella qui sopra NON è in 1NF in quanto, pur avendo una chiave primaria, presenta un attributo (dettagli) che non contiene dati in forma atomica, ma un gruppo di attributi.

• Codice Fiscale	Nome	Dettagli
LBRRSS79Y12T344A	Alberto	età: 30; professione: Impiegato
GNNBNCT84A11L611B	Gianni	età: 24; professione: Studente

Modifichiamo la tabella aggiungendo gli opportuni attributi:

• Codice Fiscale	Nome	eta	professione
LBRRSS79Y12T344A	Alberto	30	Impiegato
GNNBNCT84A11L611B	Gianni	24	Studente

## Altro esempio

Id	Nome	Corsi
1	Maria	Matematica, Fisica
2	Giovanni	Biologia

La tabella di fianco NON è in 1NF in quanto, ogni colonna deve assumere un solo valore, ovvero non può essere una matrice o un'array di valori.

Id	Corso	Nome
1	Fisica	Maria
1	Matematica	Maria
2	Biologia	Giovanni

In questo caso la normalizzazione consiste nel riportare le celle che originariamente erano raggruppate in una unica colonna in più righe replicando gli altri valori

## La seconda forma normale (2NF)

Perché una base dati possa essere in 2NF è necessario che:

- si trovi già in 1NF;
- tutti gli *attributi non chiave* dipendano dall'intera chiave primaria (e non solo da una parte di essa).

Ora la tabella precedente che rispetta la 1FN non rispetta però la seconda forma normale.

Id	Corso	Nome
1	Fisica	Maria
1	Matematica	Maria
2	Biologia	Giovanni

La chiave primaria è la combinazione (Id, Corso).

L'attributo "Nome" dipende solo da Id, non da tutto l'insieme (Id, Corso).

Quindi c'è una dipendenza parziale  $\Rightarrow$  violazione della 2NF.

Soluzione: scomposizione in due tabelle

Id_insegnante	Nome
1	Maria
2	Giovanni

Id_corso	Corso	Id_insegnante
1	Fisica	1
2	Matematica	1
3	Biologia	2

Altro esempio: si supponga di avere a che fare con il database di una scuola con una chiave primaria composta dai campi "Codice Matricola" e "Codice Esame":

● Codice Matricola	● Codice Esame	Nome Matricola	Voto Esame
1234	M01	Rossi Alberto	6
1234	L02	Rossi Alberto	7
1235	L02	Verdi Mario	8

Il database qui sopra si trova in 1NF ma non in 2NF

Perché il campo "Nome Matricola" non dipende dall'intera chiave ma solo da una parte di essa ("Codice Matricola").

• Codice Matricola	• Codice Esame	Nome Matricola	Voto Esame
1234	M01	Rossi Alberto	6
1234	L02	Rossi Alberto	7
1235	L02	Verdi Mario	8

Per rendere il nostro database in 2NF dovremo scomporlo in due tabelle:

• Codice Matricola	• Codice Esame	Voto Esame	• Codice Matricola	Nome Matricola
1234	M01	6	1234	Rossi Alberto
1234	L02	7	1235	Verdi Mario
1235	L02	8		

## La terza forma normale (3NF)

Un database è in 3NF se:

- è già in 2NF (e quindi, necessariamente, anche 1NF);
- tutti gli attributi non chiave dipendono direttamente dalla chiave, quindi *non ci sono attributi "non chiave" che dipendono da altri attributi "non chiave"*.



Supponiamo di avere una base dati di una palestra in cui il codice fiscale dell'iscritto al corso frequentato è associato all'insegnante di riferimento.

Si supponga che il nostro DB abbia un'unica chiave primaria ("Codice Fiscale") e sia così strutturato:

• Codice Fiscale	Codice Corso	Insegnante
LBRRSS79Y12T344A	BB01	Marco
GNNBNCT84A11L61B	BB01	Marco
LBRMNN79E64A112A	BB01	Marco
GLSTMT59U66P109B	AE02	Federica

Il nostro database non è in 3NF in quanto il campo "insegnante" non dipende dalla chiave primaria ma dal campo "Codice Corso" (che non è chiave).

• Codice Fiscale	Codice Corso	Insegnante
LBRRSS79Y12T344A	BB01	Marco
GNNBNCT84A11L611B	BB01	Marco
LBRMNN79E64A112A	BB01	Marco
GLSTMT59U66P109B	AE02	Federica

Per normalizzare il nostro DB in 3NF dovremo scomporlo in due tabelle:

• Codice Fiscale	Codice Corso
LBRRSS79Y12T344A	BB01
GNNBNCT84A11L611B	BB01
LBRMNN79E64A112A	BB01
GLSTMT59U66P109B	AE02

• Codice Corso	Insegnante
BB01	Marco
AE02	Federica

# Tipi di dato

In una tabella MySQL per ciascuna colonna possiamo definire diversi tipi di dato (dominio):

- ◉ **Numerics** (numeri interi e a virgola mobile)
- ◉ **String** (stringa)
- ◉ **Date , Time** (data e ora)
- ◉ **JSON**

# Dati numerici: interi

Tipo	Intervallo di valori	Solo se positivi (UNSIGNED)
BIT(M)	Da 1 a 64	/
TINYINT (1 byte)	da -128 a +127	da 0 a 255
SMALLINT (2 byte)	da -32 768 a +32 767	da 0 a 65 535
MEDIUMINT (3 byte)	da -8 388 608 a +8 388 607	da 0 a 16 777 215
INT (4 byte)	da -2 147 483 648 a +2 147 483 647	da 0 a 4 294 967 295
BIGINT (8 byte)	da -9 223 372 036 854 775 808 a +9 223 372 036 854 775 807	da 0 a 18 446 744 073 709 550 615

E' importante precisare che se all'interno di un campo di tipo numerico si cerca di inserire un valore maggiore di quanto ammesso dal tipo prescelto, MySQL produrrà un errore.

# Tipi di numeri in virgola mobile (valore approssimativo)

Tipo	Tipo (sintassi)	Tipo (sintassi deprecata)*	spazio
FLOAT	FLOAT[(p)], con p compreso tra 0 e 23	FLOAT, FLOAT(M,D)*	4byte
DOUBLE	FLOAT[(p)], con p compreso tra 24 e 53	DOUBLE, DOUBLE(M,D)*	8byte
DECIMAL	DECIMAL(M,D)		Dipende da M*

Nel caso di FLOAT, puoi specificare opzionalmente la precisione, usando una sintassi tipo FLOAT(p), dove "p" è la precisione in bit.

Se la precisione è tra 0 e 23, la colonna sarà considerata di tipo FLOAT e userà 4 byte.

Se la precisione è tra 24 e 53, sarà considerata di tipo DOUBLE e userà 8 byte.

MySQL permette anche una sintassi non standard, del tipo FLOAT(M,D) (o DOUBLE(M,D)).

M è il numero totale di cifre che puoi memorizzare (prima e dopo il punto decimale).

D è il numero di cifre dopo il punto decimale.

Per esempio, se dichiari una colonna come FLOAT(7,4), puoi memorizzare fino a 7 cifre in totale, con 4 cifre dopo il punto decimale.

Quindi i numeri vanno da -999.9999 a 999.9999.

\* <https://dev.mysql.com/doc/refman/8.4/en/precision-math-decimal-characteristics.html>

# Dati stringa

Tipo e lunghezza massima consentita

```
nome VARCHAR(20)
codiceFiscale CHAR(16)
titolo TINYTEXT
messaggio TEXT
```

Tipo	Lunghezza massima
CHAR(n)	255 caratteri
VARCHAR(n)	65.535 caratteri
BINARY(b)	255 byte
VARBINARY(b)	65.535 byte
TINYTEXT	255 caratteri
TINYBLOB	255 byte
TEXT	65.535 caratteri
BLOB	65.535 byte
MEDIUMTEXT	16.777.215 caratteri
MEDIUMBLOB	16.777.215 byte
LONGTEXT	4.294.967.295 caratteri
LOB	4.294.967.295 byte
ENUM('value1','value2',...)	2byte (65.535 valori distinti)
SET('value1','value2',...)	64 valori distinti

I tipi **CHAR** e **VARCHAR** sono sicuramente i tipi più utilizzati.

La differenza tra questi due tipi è data dal fatto che CHAR ha *lunghezza fissa*, VARCHAR ha *lunghezza variabile*.

Questo significa che in una colonna **CHAR**(10) tutti i valori memorizzati occuperanno lo spazio massimo anche se costituiti da 3 soli caratteri.

I tipi **TEXT** e **BLOB** (Binary Large Object) consentono di memorizzare grandi quantità di dati:

- ◉ TEXT è utilizzato per dati di tipo testuale,
- ◉ BLOB è utilizzato per ospitare dati binary (ad esempio il sorgente di un'immagine)

## BINARY e VARBINARY

I tipi **BINARY** e **VARBINARY** sono simili a CHAR e VARCHAR, tranne per il fatto che *memorizzano stringhe binarie* anziché stringhe non binarie: memorizzano stringhe di byte anziché stringhe di caratteri.

Per questi campi il *set di caratteri* e la *collation*, il *confronto* e l'*ordinamento* si basano sui valori numerici dei byte memorizzati.



## TEXT vs VARCHAR()

### TEXT

- ◉ dimensione massima fissa di 65535 caratteri (non è possibile limitare la dimensione massima)
- ◉ prende  $2 + c$  byte di spazio su disco, dove  $c$  è la lunghezza della stringa memorizzata.
- ◉ indice: può essere indicizzato solo con un indice: prefix index.

### VARCHAR (M)

- ◉ dimensione massima variabile di caratteri M
- ◉ M deve essere compreso tra 1 e 65535
- ◉ prende  $1 + c$  byte (per  $M \leq 255$ ) o  $2 + c$  (per  $256 \leq M \leq 65535$ ) byte di spazio su disco dove  $c$  è la lunghezza della stringa memorizzata
- ◉ può essere parte di un indice

Se è necessario memorizzare stringhe più lunghe di circa 64 KB, utilizzare MEDIUMTEXT o LONGTEXT.

VARCHAR non supporta la memorizzazione di valori così grandi.

## Tipi ENUM e SET

I tipi ENUM e SET sono un tipo di dato di testo in cui le colonne possono avere solo dei valori predefiniti.

**ENUM:** Tipo di dato ENUMerazione.

Contiene un insieme di valori prefissati tra cui scegliere: **si può inserire solamente uno dei valori previsti.**

I valori sono inseriti tra parentesi(elenco separato da virgola) dopo la dichiarazione ENUM.

```
genere ENUM( 'F', 'M', 'NB' )
```

La colonna genere accetterà solamente i valori F , M o NB. Se proviamo a mettere un valore diverso con il comando INSERT, MYSQL restituirà errore.

**SET:** è una estensione di ENUM.

```
interessi SET( 'a', 'b', 'c', 'd' )
```

Come per ENUM i valori sono fissi e disposti dopo la dichiarazione SET; tuttavia, le colonne SET possono assumere più di un valore tra quelli previsti.

# DateTime

Tali tipi di dati sono molto utili quando si ha a che fare con informazioni riguardanti la data e l'orario.  
Di seguito una tabella riepilogativa

Tipo	Formato	Intervallo
DATETIME	YYYY-MM-DD HH:MM:SS	'1000-01-01 00:00:00' a '9999-12-31 23:59:59'
DATE	YYYY-MM-DD	1000-01-01' a '9999-12-31'
TIME	HH:MM:SS	-838:59:59' a '838:59:59
YEAR	YYYY	un anno compreso fra 1901 e 2155, oppure 0000.
TIMESTAMP	YYYY-MM-DD HH:MM:SS	'1970-01-01 00:00:01'UTC' a '2038-01-19 03:14:07' UTC'

I campi di tipo DATETIME contengono sia la data che l'orario. I valori all'interno di questi campi possono essere inseriti sia sotto forma di stringhe che di numeri.

MySQL interpreta i valori dell'anno a 2 cifre utilizzando queste regole

- I valori dell'anno nell'intervallo 00-69 diventano 2000-2069.
- I valori dell'anno nell'intervallo 70-99diventano 1970-1999.

Sia *DATETIME* sia *TIMESTAMP* possono memorizzare in automatico la data.

Per ottenere ciò in fase di definizione del campo bisogna impostare il valore di *default di memorizzazione* (es):

```
ins TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP  
data DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
```

## DATETIME o TIMESTAMP?

### TIMESTAMP

Il tipo TIMESTAMP memorizza data e ora utilizzando il fuso orario del server per le conversioni automatiche da e verso UTC.

Quando memorizzi un valore in una colonna TIMESTAMP, MySQL lo converte automaticamente nel fuso orario UTC (Coordinated Universal Time) e lo memorizza in quel formato. Quando lo leggi, il valore viene riconvertito nel fuso orario del server o in quello del client (se configurato).

Questo rende il TIMESTAMP utile quando devi mostrare gli stessi dati di data e ora a utenti in fusi orari diversi, poiché MySQL gestisce automaticamente la conversione da e verso UTC.

TIMESTAMP copre solo il range di date da 1970-01-01 00:00:01 UTC fino a 2038-01-19 03:14:07 UTC.

Se hai bisogno di memorizzare date al di fuori di questo intervallo, non è adatto.

## DATETIME o TIMESTAMP?

### DATETIME

Il tipo DATETIME memorizza la data e l'ora così come sono, senza conversioni di fuso orario. Il valore viene memorizzato esattamente come viene inserito, senza alcuna considerazione del fuso orario del server o del client.

Questo significa che DATETIME è utile quando vuoi che la data e l'ora restino fisse, indipendentemente dal fuso orario dell'utente o del server.

È una buona scelta se i dati di data e ora devono essere gli stessi per tutti, ovunque si trovino.

DATETIME ha un range molto più ampio, da 1000-01-01 00:00:00 a 9999-12-31 23:59:59, quindi può essere usato per date molto più lontane nel passato o nel futuro rispetto a TIMESTAMP.

# JSON

MySQL supporta JSON nativo come tipo di dati per gli oggetti nella notazione JSON.

Rende facile l'archiviazione, l'interrogazione e il recupero di documenti di tipo JSON piuttosto che archiviarli come stringhe di testo o BLOB binari (vedi MariaDB).

Per fare ciò mette a disposizione una serie di funzioni\*

Sintassi per la definizione di un attributo di tipo JSON

```
columnName JSON
```

\* <https://dev.mysql.com/doc/refman/8.0/en/json-function-reference.html>



# Attributi

## Per i campi numerici

Si tratta di un'istruzione ulteriore che viene passata al DBMS in fase di creazione (o modifica) di una colonna di tabella.

- ◉ **AUTO\_INCREMENT** - Aumenta automaticamente il valore di una colonna aggiungendo 1 al valore massimo già registrato nella tabella. E' opportuno utilizzarlo in combinazione con NOT NULL. Può essere applicato a tutti i tipi numerici interi.
- ◉ **UNSIGNED\*** - un campo contrassegnato con UNSIGNED non accetta valori negativi.
- ◉ **ZEROFILL\*\*** - viene utilizzato per memorizzare i numeri con degli zeri davanti nel caso in cui la lunghezza sia inferiore a quella massima prevista.

Se per esempio si dichiara un campo INT ZEROFILL e si desidera memorizzare il valore "1234", esso verrà visualizzato come "00000001234" con M(11) - sempre UNSIGNED.

\*A partire da MySQL 8.0.17 è deprecato per FLOAT, DOUBLE e DECIMAL; \*\*A partire da MySQL 8.0.17 è deprecato; [vedi documentazione mysql](#).



## AUTO\_INCREMENT (sequence)

In MySQL, una *sequenza* è un elenco di interi generati nell'ordine crescente, vale a dire 1,2,3...

Impostare l'attributo AUTO\_INCREMENT in una colonna, tipicamente una colonna chiave primaria, crea automaticamente una sequenza in MySQL.

Le seguenti regole vengono applicate quando si utilizza l'attributo AUTO\_INCREMENT:

- ◉ Ciascuna tabella ha **solo una colonna** AUTO\_INCREMENT il cui tipo di dati è **"intero"**.
- ◉ La colonna AUTO\_INCREMENT **deve essere indicizzata**, il che significa che può essere:
  - PRIMARY KEY o UNIQUE.
- ◉ La colonna AUTO\_INCREMENT deve avere un vincolo NOT NULL.

Quando si imposta l'attributo AUTO\_INCREMENT in una colonna, **MySQL aggiunge automaticamente il vincolo NOT NULL** alla colonna implicitamente.

## Attributi per i campi di tipo stringa

### BINARY:

- ◉ L'unico vincolo/opzione che può essere utilizzato per i campi destinati ad ospitare dati stringa è **BINARY** il quale può essere utilizzato con **CHAR** o **VARCHAR** qualora questi campi siano destinati ad ospitare dati binari (pur non rendendosi necessario utilizzare un campo della famiglia BLOB)
- ◉ **CHAR(n) BINARY** e **VARCHAR(n) BINARY** differiscono dal *tipo di dato* **BINARY** e **VAR BINARY** per la codifica e la collation utilizzata\*

\* <https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html>

## Attributi universali (sia per campi numerici sia per quelli testuali)

Possono essere utilizzati tanto con campi numerici quanto con campi di tipo stringa.

- ◉ **DEFAULT** – Può essere utilizzato con tutti i tipi di dati ad eccezione di TEXT e BLOB. Serve per indicare un valore di default per il campo qualora questo venga lasciato vuoto.
- ◉ **NULL / NOT NULL** – Può essere utilizzato con tutti i tipi di campi e serve per definire se un dato campo può avere un valore NULL oppure no.

## Vincoli

- ◉ **CHECK** (expression) – consente di imporre un vincolo al dato da inserire.
- ◉ **FOREIGN KEY** – consente di imporre un vincolo riferito alla chiave esterna.

## Attributi/Indici

- ◉ **UNIQUE** – Con UNIQUE si imposta una regola di unicità, questo significa che nessun dato contenuto nella colonna può essere ripetuto: ogni dato deve, quindi, essere unico e se si cerca di inserire un dato duplicato si riceve un errore. Può essere nullo.
- ◉ **PRIMARY KEY** – Può essere utilizzato con tutti i tipi di dati (numerici e stringa) ed è una sorta di variante di UNIQUE che consente di creare un indice primario sulla tabella (campo chiave).
- ◉ **INDEX** [KEY] (*colonne*) – E' utilizzato per creare un'indice nella tabella ai fini di migliorare le performances di accesso ai dati.

# INDEX (indici)

(introduzione al concetto, verranno trattati in modo approfondito in questa sezione)

- Servono ad ottimizzare le performance del database.
- Un indice è una struttura dati ausiliaria che consente di recuperare più velocemente i dati di una tabella, evitandone la lettura dell'intero contenuto (full table scan), tramite una selezione più mirata.
- devono essere usati consapevolmente per non ottenere l'effetto contrario ovvero rallentare il db.

# SQL – Structured Query Language

**SQL** – Structured Query Language, è il linguaggio che permette di effettuare le operazioni per estrarre e manipolare i dati da un database.

E' lo standard tra i sistemi relazionali: viene usato in tutti i prodotti DBMS come set di comandi per l'utente della base di dati

Tipi di istruzioni SQL

- ◉ **DCL** (Data control language): permette di gestire il controllo degli accessi e i permessi per gli utenti
- ◉ **DDL** (Data Definition Language): permette di definire la struttura del database
- ◉ **DML** (Data manipulation language): permette di modificare i dati contenuti nel db, con le operazioni di inserimento, variazione e cancellazione
- ◉ **TCL** (Transaction Control Language): queste operazioni gestiscono le transazioni nel database
- ◉ **Query Language**: permette di porre interrogazioni al db

## DCL

gestire il controllo degli accessi e i permessi per gli utenti:

```
CREATE USER 'vecchione'@'host' IDENTIFIED BY 'password';
```

```
GRANT ALL  
ON nomedb.*  
TO 'user'@'host';
```

## DDL

permette di definire la struttura del database

```
CREATE DATABASE dbName; -- crea un nuovo database
DROP DATABASE dbName; -- cancella il database
CREATE TABLE tableName(...); -- crea una nuova tabella nel DB
ALTER TABLE tableName ... ; -- modifica la struttura di una tabella
DROP TABLE tableName ... ; -- cancella una tabella dal DB
CREATE INDEX indexName ... ; -- crea un indice su una certa tabella
ALTER TABLE tableName DROP INDEX indexName ... ; -- elimina l'indice specificato
```



## DML

permette di *modificare i dati contenuti nel db*, con le operazioni di inserimento, variazione e cancellazione

- ◉ Inserimento:

```
INSERT INTO tableName(field1, field2, ...)  
VALUES ('value1', 'value2', '...');
```

- ◉ Cancellazione:

```
DELETE FROM tableName  
WHERE column_name = some_value;
```

- ◉ Aggiornamento:

```
UPDATE tableName  
SET column_name = new_value  
WHERE column_name = some_value;
```

## TCL

gestiscono le transazioni nel database

```
COMMIT; -- rende definitive le operazioni sul database  
ROLLBACK; -- ripristina i dati eliminando le modifiche temporanee  
SAVEPOINT save_point_name(...); -- crea un punto di salvataggio
```

## Query Language

permette di porre interrogazioni al db

```
SELECT field(s)  
FROM table(s)  
WHERE condition(s);
```

Attraverso **SELECT** vengono selezionati dei campi (*attributi*) da una o più tabelle e restituiti all'utente sotto forma di una "nuova tabella" (*resultset*)

Attraverso la clausola **WHERE** è possibile filtrare il *resultset* sulla base di alcune regole

# Installazione MySQL 8

Dal sito di mysql scaricate il pacchetto relativo al vostro sistema operativo (Linux, Windows o MacOS).

<https://dev.mysql.com/downloads/mysql/>

...

Il *client mysql* verrà installato nella cartella...



► C:\Program Files\MySQL\MySQL Server 8.0\bin



► /usr/local/mysql-8.0.26-macos10.15-x86\_64/bin

dove si trova l'eseguibile di mysql

# Avvio del servizio

MySQL viene eseguito come servizio o come *daemon*.

Un servizio o demone è un *programma in esecuzione continua* nel sistema operativo, il cui compito è quello di rimanere in attesa di richieste per la fruizione di specifiche funzionalità.

Il demone si chiama *mysqld*:

*mysqld* viene avviato in automatico all'avvio del sistema

(a seconda della configurazione applicata durante l'installazione)

Possiamo verificare che il servizio sia attivo con **mysqladmin**

```
mysqladmin -u root -p ping
```

Fornendo i corretti dati di autenticazione, se il DBMS è attivo, sarà stampato il messaggio "mysql is alive";

```
mysqld is alive
```

se invece non è attivo, verrà mostrato un errore di connessione.

```
mysqladmin: connect to server at 'localhost' failed  
error: 'Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)'  
Check that mysqld is running and that the socket: '/tmp/mysql.sock' exists!
```

# Avvio/Arresto del servizio

## MacOSX

Su mac è installato un controllo tra le Preferenze di sistema per gestire il servizio di MySQL.

Qui avviate/arrestate il servizio cliccando sul pulsante:

Stop / Start MySQL Server

## Windows

Apri la finestra *Esegui/Start* usando il tasto: Win + R

Digita *services.msc*

Ora cerca il servizio MySQL in base alla versione installata.

Fare clic su *interrompi*, *avvia* o *riavvia* il servizio.



# PATH MySQL

Il *client mysql* si trova nella cartella...

- ▶ C:\Program Files\MySQL\MySQL Server 8.0\bin

Ogni volta che di deve accedere al servizio, una volta aperto il terminale, bisogna portarsi nel punto in cui è stato installato mysql:

```
cd C:\Program Files\MySQL\MySQL Server 8.X\bin
```

A questo punto possiamo accedere al servizio avviando mysql.

Per evitare di digitare ogni volta il percorso bisogna creare una scorciatoia per la shell

In Windows bisogna *aggiungere una variabile di ambiente* relativa al path di MySQL.





# PATH MySQL

Il *client mysql* si trova nella cartella...

- ▶ `/usr/local/mysql-8.0.23-macos10.15-x86_64/bin`

Ogni volta che di deve accedere al servizio, una volta aperto il terminale, bisogna portarsi nel punto in cui è stato installato mysql:

```
cd /usr/local/mysql-8.0.26-macos10.15-x86_64/bin/
```

A questo punto possiamo accedere al servizio avviando mysql (vedi slide 70)

Per creare una scorciatoia per la shell (impostare la shell bash) su mac bisogna creare un file nascosto ".bash\_profile" con la scorciatoia scritta in questo modo:

```
export PATH=${PATH}:/usr/local/mysql/bin
```

# Accesso al DBMS

Amministratore del servizio (DBA - Data Base Administrator)

Da terminale accedere a MySql:

```
mysql -u root -p
```

Vi verrà chiesto di inserire la password ( è quella creata in fase di installazione)

```
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 16  
Server version: 8.0.23 MySQL Community Server - GPL  
  
Copyright (c) 2000, 2021, Oracle and/or its affiliates.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
mysql>
```

Visualizzate elenco database disponibili:

```
mysql> show databases;
```

Essendo una prima installazione dovrete vedere i seguenti db:

```
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
+-----+  
4 rows in set (0.01 sec)  
  
mysql>
```

I database elencati sono funzionali al servizio, sono disponibili solo all'amministratore del sistema;

*information\_schema* e *performance\_schema* sono disponibili anche all'utente per ottenere informazioni sul proprio database e sulle prestazioni delle query.

# DDL

Data Definition Language (1parte)

# Creare un database

Una volta effettuato l'accesso possiamo eseguire l'istruzione CREATE DATABASE (CREATE SCHEMA) seguita dal nome del database da creare.

```
CREATE DATABASE databaseName;
```

Se il database è già presente mysql ve lo segnala attraverso un messaggio di errore:

```
ERROR 1007 (HY000): Can't create database 'nomedatabase'; database exists
```

Usando la sintassi seguente

```
CREATE DATABASE IF NOT EXISTS databaseName;
```

mysql verifica l'esistenza del db: se non esiste lo crea, se esiste vi da ok ma segnala un *warning*.

```
Query OK, 1 row affected, 1 warning (0.01 sec)
```

In fase di creazione di un database con MySQL è anche possibile specificare charset e collation; ad esempio:

```
CREATE DATABASE IF NOT EXISTS databaseName  
CHARACTER SET utf8  
COLLATE utf8_general_ci;
```

Specificando questi valori è possibile "sovrascrivere" quelli impostati di default a livello server.

Per visualizzare come è stato creato il database

```
SHOW CREATE DATABASE databaseName;
```

Per elencare rispettivamente i set di caratteri disponibili e le "collezioni" disponibili.

```
SHOW CHARACTER SET; SHOW COLLATION;
```

# Cancellare un database

Per eliminare un DB basta scrivere l'istruzione DROP DATABASE (DROP SCHEMA) seguita dal nome del database da rimuovere.

```
DROP DATABASE databaseName;  
DROP DATABASE IF EXISTS databaseName;
```

Se si usa l'istruzione opzionale **IF EXISTS** si evita di ricevere l'errore qualora il database sia già stato eliminato.

# DCL

Data Control Language



## DCL

gestire il controllo degli accessi e i permessi per gli utenti:

Istruzione **CREATE USER**: crea l'utente e assegna una password

```
CREATE USER 'app_ep'@'localhost' IDENTIFIED BY 'ep_2025!';
```

**password** : password associata all'utente che stiamo creando.

La password va scritta "in chiaro".

# GRANT

Assegna i permessi e/o privilegi

```
GRANT --istruzioni_consentite  
ON namedatabase.*  
TO 'nameuser'@'hostuser';
```

**istruzioni\_consentite:** CREATE, SELECT, UPDATE, DELETE, ALTER, DROP,...

Per dare all'utente permessi completi utilizzare la parola chiave ALL.

**database.\*** : nome del database sul quale l'utente potrà eseguire le istruzioni consentite.

Per tutte le tabelle del db: .\* . Si può specificare il nome di una o più tabelle. Per tutti i database: \*.\* .

**user** : specifica il nome dell'utente che vogliamo creare o al quale vogliamo assegnare nuovi permessi.

**host** : specifica il/gli host da cui è ammessa la connessione. Se voglio indicare più IP devo usare la wild card %: 130.192.200.%

## REVOKE

```
REVOKE --istruzioni_revocate  
ON databaseName.*  
FROM 'user'@'host';
```

per la quale valgono le stesse regole sopra viste per **GRANT** .

Per eliminare tutti i privilegi:

```
REVOKE ALL PRIVILEGES, GRANT OPTION -- istruzioni_revocate  
FROM 'user'@'host';
```

Quest'ultima sintassi elimina ogni permesso dell'utente su qualunque database del sistema.

## Cambiare/aggiornare la password MySQL degli utenti

Per cambiare una normale password utente devi digitare:

- ◉ *Cambia password per l'utente (da root):*

```
ALTER USER 'userName'@'host' IDENTIFIED BY 'newpass';
```

- ◉ Cambiare la propria password:

```
ALTER USER USER() IDENTIFIED BY 'newpass';
```

## Verificare i permessi utente

Verificare i privilegi di uno specifico utente:

```
SHOW GRANTS FOR 'user'@'localhost';
```

Verificare i privilegi dell'utente attualmente loggato a MySQL:

```
SHOW GRANTS FOR CURRENT_USER;
```

## Eliminare un utente da MySQL

```
DROP USER 'user'@'localhost';
```

questo comando rimuove l'utente e i suoi permessi.

Visualizzare elenco utenti mysql (solo utente root)

```
SELECT user, host FROM mysql.user;
```

# Accesso al DBMS

Utente

Da terminale accedere a MySql utilizzando le credenziali dell'utente creato:

```
mysql -u nomeUser -p
```

Vi verrà chiesto di inserire la password ( quella assegnata all'utente)

```
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 16  
Server version: 8.0.23 MySQL Community Server - GPL
```

```
Copyright (c) 2000, 2021, Oracle and/or its affiliates.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql>
```

```
mysql> show databases;
```

## Accesso al DBMS - Utente

A questo punto dovete prendere possesso del database per poterci lavorare:

L'istruzione è **USE** più nome del database assegnato (senza **;** a chiusura del comando)

```
use nomeDatabase
```

Da terminale potete accedere *anche direttamente* al database per i quali avete i privilegi.

Dovete passare a mysql tutti i parametri (tranne la password) più il nome del database:

```
mysql -u nomeUser -p nomeDatabase
```

Non dovete più usare **USE**, siete nel database e potete lavorarci\*

\*le applicazioni, di solito, hanno un file dedicato alla connessione al db in cui vengono passati tutti i parametri di connessione compreso l'host e in alcuni casi anche la porta (dipende dal provider del servizio).

# DDL

Data Definition Language (2 parte)



# Creare le tabelle

Per creare una tabella usiamo il comando `CREATE TABLE tableName()`.

Quando creiamo una tabella dobbiamo definire anche tutti i campi ad essa associati, argomenti della parentesi.

- ◉ Per ogni campo verrà definito il dominio che indica al sistema quale tipo di dati verrà memorizzato nel campo.
- ◉ Per ogni campo definiamo anche gli eventuali attributi

```
CREATE TABLE IF NOT EXISTS nome_tabella(  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  fieldName1 VARCHAR(60) NOT NULL,  
  fieldName2 DATE,  
  fieldName3 TINYINT  
)  
[ENGINE InnoDB CHARACTER SET utf8 COLLATE utf8_general_ci];
```

Esempio di creazione di una tabella denominata studente, con il campo id come chiave primaria.

```
CREATE TABLE IF NOT EXISTS studenti(  
  id INT AUTO_INCREMENT,  
  nome VARCHAR(20),  
  cognome VARCHAR(30) NOT NULL,  
  genere ENUM('m', 'f'),  
  indirizzo VARCHAR(100),  
  citta VARCHAR(30),  
  provincia CHAR(2) DEFAULT 'To',  
  regione VARCHAR(30) DEFAULT 'Piemonte',  
  email VARCHAR(100) NOT NULL UNIQUE,  
  data_nascita date,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
  PRIMARY KEY(id)  
);
```

Verificare se e come è stata creata la tabella:

- ◉ Per verificare se la nostra tabella è stata creata si usa l'istruzione che mostra le tabelle presenti:

```
SHOW TABLES;
```

- ◉ Per verificare che la struttura della tabella sia corretta:

```
SHOW CREATE TABLE tableName;
```

- ◉ Per visualizzare come è stata creata una tabella:

```
DESCRIBE tableName; DESC tableName;
```

Rinominare una tabella:

```
ALTER TABLE tableName RENAME newtableName;
```

```
RENAME TABLE tableName TO newtableName;
```

## INFORMAZIONI SULLE TABELLE

Per conoscere la struttura della tabella con più o meno informazioni (valore dell'auto\_increment, data di creazione, collation)

```
DESCRIBE tableName; DESC tableName;
```

```
SHOW COLUMNS FROM tableName;
```

```
SHOW FULL COLUMNS FROM tableName;
```

```
SHOW INDEX FROM tableName;
```

```
SHOW TABLE STATUS LIKE 'tableName'1; -- mostra valore auto_increment
```

→ *Mostrano le stesse informazioni*

→ *Mostra ulteriori informazioni (privilegi e commenti)*

Per conoscere solo il valore dell'auto\_increment di una tabella possiamo interrogare anche il db INFORMATION\_SCHEMA

```
SELECT table_name, auto_increment  
FROM information_schema.tables  
WHERE table_schema = 'databaseName'1;  
-- AND TABLE_NAME = 'tableName';
```

1) Le statistiche della tabella vengono memorizzate nella cache. Per disabilitare la cache e avere sempre l'ultima versione è necessario modificare la variabile del server che indica la durata del cache-clear a 0:

```
SET PERSIST information_schema_stats_expiry = 0 -- bisogna usare l'utente DBA (root)
```

## INFORMATION\_SCHEMA

INFORMATION\_SCHEMA fornisce l'accesso ai metadati del database, le informazioni sul server MySQL come il nome di un database o una tabella, il tipo di dati di una colonna, o privilegi di accesso...

### Note sull'utilizzo INFORMATION\_SCHEMA

- ◉ INFORMATION\_SCHEMA è una banca dati all'interno di ciascuna istanza di MySQL, il **luogo che memorizza le informazioni su tutti gli altri database che il server MySQL mantiene**.  
Il database INFORMATION\_SCHEMA contiene diverse tabelle di sola lettura (in realtà viste).
- ◉ Anche se è possibile selezionare INFORMATION\_SCHEMA come database predefinito con un'istruzione USE, *è possibile SOLO leggere il contenuto delle tabelle*, non eseguire INSERT, UPDATE o DELETE.

## INFORMATION\_SCHEMA

Vediamo un esempio di utilizzo di INFORMATION\_SCHEMA

```
SELECT table_name, table_type, engine, table_collation  
FROM information_schema.tables  
WHERE table_schema = 'databaseName'  
ORDER BY table_name;
```

Questa select ci restituirà l'elenco delle tabelle del nostro database indicando il tipo di tabella e l'engine associato.

# Modificare le tabelle

L'istruzione **ALTER TABLE** viene utilizzata per aggiungere, eliminare o modificare le colonne di una tabella esistente.

Per *aggiungere un campo a una tabella*, utilizzare la seguente sintassi:

```
ALTER TABLE tableName  
ADD fieldName [DATATYPE];
```

Per modificare *nome e datatype* di un campo di una tabella

```
ALTER TABLE tableName  
CHANGE fieldName newFieldName [DATATYPE];
```

Per modificare *solo datatype* di un campo di una tabella

```
ALTER TABLE tableName  
MODIFY fieldName [DATATYPE];
```

Per modificare *solo nome* di un campo di una tabella

```
ALTER TABLE tableName  
RENAME COLUMN oldName TO newName;
```

Per cancellare un campo di una tabella

```
ALTER TABLE tableName  
DROP fieldName;
```

Potete combinare le istruzioni separandole con una virgola

```
ALTER TABLE tableName  
CHANGE fieldName newFieldName [DATATYPE],  
ADD fieldName3 [DATATYPE] AFTER fieldName2,  
DROP fieldName4;
```



Per aggiungere un campo in una data posizione

```
ALTER TABLE tableName  
ADD fieldName2 [DATATYPE]  
AFTER fieldName;
```

Usando *FIRST* al posto di *AFTER* si aggiunge il campo in prima posizione, *FIRST* non vuole il nome del campo

Per spostare un campo in una tabella

```
ALTER TABLE tableName  
MODIFY fieldName2 [DATATYPE]  
AFTER fieldName;
```

Per spostare un campo in prima posizione

```
ALTER TABLE tableName  
MODIFY fieldName2 [DATATYPE]  
FIRST;
```

se il campo è una chiave primaria non dovete indicarlo nel datatype altrimenti ricevete un errore di mysql

- ◉ Aggiungere alla tabella la PRIMARY KEY (se non già impostata):

```
ALTER TABLE tableName ADD PRIMARY KEY (field1[, field2, ...]);
```

- ◉ Aggiungere alla tabella la PRIMARY KEY aggiungendo campo nuovo apposito:

```
ALTER TABLE tableName  
ADD id INT AUTO_INCREMENT PRIMARY KEY;
```

- ◉ Eliminare la PRIMARY KEY

```
ALTER TABLE tableName DROP PRIMARY KEY;
```

Attenzione, per eliminare la primary key di un campo id con auto\_increment, bisogna prima eliminare l'attributo auto\_increment

```
ALTER TABLE tableName MODIFY id INT;
```

## Duplicare tabelle

Se abbiamo necessità di duplicare una tabella possiamo utilizzare l'istruzione **CREATE TABLE** combinata con l'istruzione **LIKE**.

Per duplicare una tabella possiamo scrivere:

```
CREATE TABLE tableNameCopy LIKE tableName;
```

Questa istruzione duplica solo la struttura della tabella.

# Cancellare una tabella dal database MySQL

Vediamo l'operazione inversa alla creazione di tabelle, la loro eliminazione.

- Per eliminare una tabella utilizzeremo il comando DROP TABLE.

```
DROP TABLE tableName;
```

L'eliminazione di una tabella, come per il database, è un'operazione irreversibile

- È possibile eliminare più di una tabella contemporaneamente:

```
DROP TABLE tableName, tableName2, tableName3;
```

# DML

Data Manipulation Language

# Creazione, lettura, aggiornamento e eliminazione dei record (CRUD)

Una volta creata la struttura del nostro database ci ritroveremo, ovviamente, con una serie di tabelle vuote.

Prima di aggiungere record a una tabella bisogna conoscere il tipo di dati previsto per ogni campo, quali campi non possono avere valore nullo, quali campi hanno l'incremento automatico...

Quando si inseriscono i dati bisogna usare le *virgolette* o gli *apici* per i dati *tipo stringa* (compresa la data), *senza virgolette* o *apici* per i dati di *tipo numerico*.

Non si inseriscono i valori per i campi definiti con l' *auto\_increment*.

## Consistenza dei dati

La consistenza dei dati è una proprietà fondamentale nei database che garantisce che i dati rispettino regole e vincoli predefiniti, mantenendo uno stato valido e coerente.

In altre parole, **i dati devono sempre essere corretti e rispettare le regole definite dal database.**

```
CREATE TABLE studenti (  
    id INT PRIMARY KEY,  
    nome VARCHAR(50),  
    età TINYINT UNSIGNED CHECK (età >= 18) -- Vincolo di consistenza  
);
```

La consistenza fa parte delle proprietà ACID (Atomicità, Consistenza, Isolamento, Durabilità), fondamentali per garantire la corretta esecuzione delle transazioni nei database relazionali.

## INSERT INTO

INSERT INTO è l'istruzione utilizzata per inserire nuovi record in una tabella. Ha due parti

**INSERT INTO** seleziona la tabella e i campi per i quali effettuare l'inserimento

**VALUE/VALUES** elenca i valori dei campi da inserire

```
INSERT INTO tableName(field1, field3)
VALUES(value1, value3);
```

È possibile inserire più record con un solo INSERT separando l'elenco dei valori di ogni record con la: ,

```
INSERT INTO tableName(field1, field2, field3,...)
VALUES(r1_value1, r1_value2, r1_value3, ...),(r2_value1, r2_value2, r2_value3, ...);
```

Altra sintassi per singolo record con istruzione **SET**:

```
INSERT INTO tableName
SET field1 = 'value1', field2 = 'value2', field3 = 'value2';
```



## INSERT INTO

È possibile usare il comando **INSERT INTO** senza l'uso di nomi di campo se si inserisce un record rispettando l'ordine dei campi della tabella

```
INSERT INTO tableName  
VALUES(value1, value2, value3);
```

in questo caso devono essere inseriti i valori di tutti i campi, anche i valori *AUTO\_INCREMENT* o *TIMESTAMP* (passare default per inserimento automatico).

Per i campi che accettano i valori nulli potete passare `null`.

```
INSERT INTO studente  
VALUES(default, 'fabio', 'rossi', 'fbr@gmail.com', null, default);
```

## Mostrare i record di una tabella

È possibile visualizzare i record di una tabella utilizzando l'istruzione **SELECT**.

Per visualizzare tutti i record da una tabella si usa il carattere jolly \*.

Dobbiamo anche utilizzare l'istruzione **FROM** per identificare la tabella che vogliamo interrogare:

```
SELECT * FROM tableName;
```

Di solito si visualizzano campi specifici, piuttosto che l'intera tabella.

Dopo l'istruzione SELECT elencare i campi che interessano, separati da una virgola.

```
SELECT fieldName, fieldName2, fieldName3 FROM tableName;
```

## INSERT INTO ... SELECT

L'istruzione INSERT INTO ... SELECT consente di inserire automaticamente dati in una tabella copiandoli da un'altra:

```
INSERT INTO amici(nome, cognome)  
SELECT nome, cognome  
FROM studenti;
```

Ad esempio, possiamo popolare la tabella *amici* copiando i dati già presenti nella tabella *studenti*.

- La tabella di destinazione (amici) deve esistere.
- I tipi di dato tra i campi selezionati devono essere compatibili.
- L'ordine dei campi nel SELECT deve corrispondere all'ordine dei campi specificati in INSERT.

## CREATE TABLE ... SELECT

Si possono creare delle tabelle già popolate di dati mediante l'uso congiunto delle istruzioni **CREATE TABLE** e **SELECT**.

**CREATE TABLE** crea la nuova tabella,  
**SELECT** carica i dati prelevandoli da un'altra tabella.

Il suo funzionamento, in pratica, è analogo a quello di **INSERT INTO ... SELECT**.

```
CREATE TABLE parenti (  
    id INT AUTO_INCREMENT,  
    nome VARCHAR(20),  
    cognome VARCHAR(30),  
    PRIMARY KEY(id)  
) SELECT nome, cognome FROM amici;
```

Questo comportamento è **supportato da MySQL** e permette di definire la struttura a mano e popolare subito la tabella. Non è STANDARD SQL.

## Duplicare tabelle e suoi contenuti

Se abbiamo necessità di copiare il contenuto di una tabella in altra tabella, possiamo utilizzare l'istruzione **CREATE TABLE** combinata con **LIKE** e le istruzioni **SELECT**.

Per duplicare esattamente una tabella(con indici e chiavi) e i suoi contenuti bisogna usare due istruzioni separate:

```
CREATE TABLE studenti_bk LIKE studenti;  
INSERT INTO studenti_bk SELECT * FROM studenti;
```

Si può usare anche un'istruzione sola: in questo caso gli indici non vengono ricreati, cioè le strutture delle tabelle sono diverse:

```
CREATE TABLE studenti_bk2 [AS]  
SELECT * FROM studenti;
```

## UPDATE

Aggiornamento dei record in una tabella.

Questa istruzione modifica il valore presente in una colonna di un record già esistente.

Viene utilizzata insieme all'istruzione **SET**:

```
UPDATE tableName  
SET field1 = value1, field2 = value2  
WHERE field3 = value3;
```

- ◉ dopo **UPDATE** indichiamo quale tabella è interessata
- ◉ con **SET** specifichiamo quali colonne modificare e quali valori assegnare
- ◉ con **WHERE** (opzionale) stabiliamo le condizioni che determinano quali righe saranno interessate dalle modifiche (Attenzione: **se WHERE è omissso, tutti i record saranno aggiornati** per le colonne indicate).

Per operare simultaneamente su più campi è sufficiente suddividere le coppie chiave/valore con una virgola.

Quando si inseriscono i dati in una tabella rammentate sempre come sono state definite le colonne per evitare errori di inserimento.

Se si inserisce un valore *troppo lungo*, o *non compreso* dalla definizione della colonna, MySQL restituisce un errore\* e non effettua alcuna modifica.

```
UPDATE studenti SET genere = 's' WHERE id = 1;  
ERROR 1265 (01000): Data truncated for column 'genere' at row 1
```

Il campo *genere* della tabella *studenti* è un campo definito come:

`ENUM( 'f', 'm', 'nb' )`, accetta quindi solo i valori *f*, *m* o *nb*.

In questo caso stiamo tentando di inserire un valore non ammesso.

\*dipende dall'impostazione della variabile globale @@sql\_mode: di default mysql lavora in strict mode.



## SQL Mode: STRICT MODE

Il server MySQL può funzionare in diverse modalità SQL e può applicare queste modalità in modo diverso per client diversi, a seconda del valore della variabile di sistema: `SQL_MODE`.

I DBA possono impostare la modalità SQL globale in modo che corrisponda ai requisiti operativi del server del sito e ogni applicazione può impostare la modalità SQL della sessione in base ai propri requisiti.

Le modalità influiscono sulla sintassi SQL supportata da MySQL e dai controlli di convalida dei dati che esegue. Ciò semplifica l'utilizzo di MySQL in ambienti diversi e l'utilizzo di MySQL insieme ad altri server di database.

```
SELECT @@SQL_MODE;
+-----+
| @@SQL_MODE |
+-----+
| NO_ZERO_IN_DATE,NO_ZERO_DATE,NO_ENGINE_SUBSTITUTION |
+-----+
```

Questo significa che se volete operare con un sessione che lavori in STRICT MODE (che attivi i controlli sui campi per esempio) dovete impostare la variabile ad inizio sessione di connessione:

```
SET SQL_MODE = 'TRADITIONAL';
```

<https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html>



## Eliminazione dei record in una tabella

L'istruzione **DELETE** viene utilizzata per eliminare gruppi di record in una tabella.

È necessario utilizzare la parola chiave condizionale **WHERE** per scegliere quali record si desidera eliminare, altrimenti si eliminano tutti i record.

La sintassi di base per l'istruzione è la seguente :

```
DELETE  
FROM tableName  
WHERE field = value;
```

esempio:

```
DELETE  
FROM studenti  
WHERE genere = 'm';
```

## Eliminare tutti i record della tabella

Per svuotare una tabella si usa l'istruzione **TRUNCATE**

```
TRUNCATE [TABLE] tableName;
```

Questa soluzione è la più veloce perché elimina la struttura della tabella per poi ricrearne una uguale vuota azzerando il valore di eventuali campi AUTO\_INCREMENT.

Usando **DELETE** si eliminano tutti i record presenti nella tabella specificata *record per record*.

```
DELETE FROM tableName;
```

Un simile modo di operare, seppur assolutamente funzionante, è poco efficiente perché dipende dalla quantità di righe presenti in tabella.

Inoltre usando DELETE il valore di un eventuale AUTO\_INCREMENT rimane inalterato; per azzerarlo:

```
ALTER TABLE tableName AUTO_INCREMENT = 1;
```

## INFORMAZIONI SULLE TABELLE

Per conoscere la struttura della tabella con più o meno informazioni (valore dell'auto\_increment, data di creazione, collation)

```
DESCRIBE tableName; DESC tableName;
```

```
SHOW COLUMNS FROM tableName;
```

```
SHOW FULL COLUMNS FROM tableName;
```

```
SHOW INDEX FROM tableName;
```

```
SHOW TABLE STATUS LIKE 'tableName'1; -- mostra valore auto_increment
```

Per conoscere solo il valore dell'auto\_increment di una tabella possiamo interrogare anche il db INFORMATION\_SCHEMA

```
SELECT table_name, auto_increment  
FROM information_schema.tables  
WHERE table_schema = 'databaseName'1;  
-- AND TABLE_NAME = 'tableName';
```

1) Le statistiche della tabella vengono memorizzate nella cache. Per disabilitare la cache e avere sempre l'ultima versione è necessario modificare la variabile del server che indica la durata del cache-clear a 0:

```
SET PERSIST information_schema_stats_expiry = 0 -- bisogna usare l'utente DBA (root)
```

# Commenti

MySQL Server supporta tre stili di commento:

- `#` da questo simbolo a fine riga.
- `--` da questo simbolo a fine riga.  
lo stile di commento (doppio trattino) richiede che il secondo trattino sia seguito da almeno uno spazio bianco o un carattere di controllo (come uno spazio, tab, nuova riga e così via).
- `/* commento */` come nel linguaggio C.  
Questa sintassi consente un commento su più righe e l'inserimento del commento *inline*.

L'esempio seguente mostra tutti e tre gli stili di commento:

```
SELECT 1 + 1; # Questo commento continua fino alla fine della riga
SELECT 1 + 1; -- Questo commento continua fino alla fine della riga
SELECT 1 /* questo è un commento in linea */ + 1;
SELECT 1 +
/*
questo è un commento
su più linee
*/
1;
```