MODELLO RELAZIONALE E DATA BASE RELAZIONALE: DATI E VISTE

Il **modello relazionale** rappresenta i dati nella forma di tabelle bidimensionali, dove ogni tabella "descrive qualcosa" che esiste nel mondo reale su cui si raccolgono informazioni (p.e. una persona, un luogo, una cosa, un evento).

Un **database relazionale** è quindi una collezione di tabelle bidimensionali.

La **vista logica del database** è l'organizzazione dei dati nelle tabelle relazionali, cioè è la forma con cui un database presenta all'utente i dati che contiene

La **vista interna** è il modo con cui il software del database salva fisicamente i dati sul disco fisso.

IL CONCETTO DI RELAZIONE

Il **modello relazionale** si basa sul concetto matematico di **RELAZIONE**

Una **RELAZIONE** è una tabella relazionale

Il database è quindi una collezione di tabelle relazionali, ognuna descritta mediante un proprio **schema logico**.

Lo **schema logico del database** descrive la struttura complessiva del database, cioè l'insieme delle tabelle, dei campi, delle chiavi e delle relazioni tra i dati; in pratica rappresenta come i dati sono organizzati a livello logico, indipendentemente da come sono memorizzati fisicamente.

Lo **schema logico della tabella** è una parte dello schema logico complessivo del database, e descrive la struttura di una singola tabella: il suo nome, gli attributi che la compongono (colonne), i tipi di dato e altre informazioni.

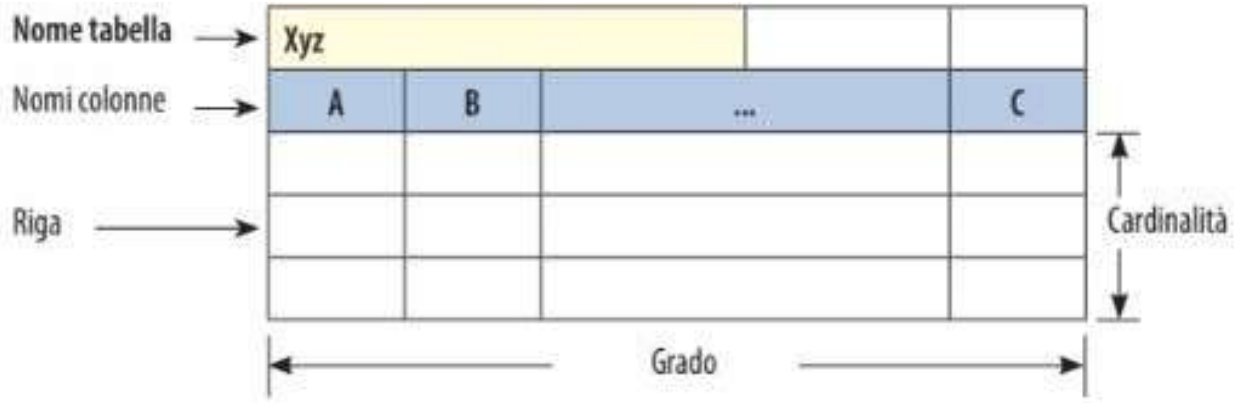
ESEMPIO

Riportiamo a titolo di esempio due semplici schemi di tabelle relazionali:

docenti [matricola, cognome, nome, materia, telefono]
libri [isbn, titolo, autore, editore, materia]

Una relazione viene rappresentata con una **tabella relazionale**, in cui i dati vengono salvati sotto forma di **colonne** (a cui viene assegnato un **nome**) e un certo numero di **righe**:

- **le colonne** rappresentano gli **attributi** o **campi della relazione**
- **una riga** (**record** o **tupla**) rappresenta un singolo esemplare dell'entità che da il nome alla tabella
- **la cella della tabella** è l'intersezione di una riga (record) con una colonna (campo) dove viene salvato un singolo **dato**



NOMENCLATURA

La relazione è rappresentata dalla **TABELLA**

L'intestazione o nome della colonna è l'**ATTRIBUTO** o **CAMPO**

La riga è la **TUPLA** o il **RECORD**

L'intersezione tra ATTRIBUTO/CAMPO e **TUPLA/RECORD** è la **CELLA**

Il dominio è l'insieme dei valori presenti in una colonna

Il grado della tabella è il numero delle colonne

La cardinalità della tabella è il numero delle righe

SQL - DDL

IL MODELLO LOGICO

Un DB ha un proprio **MODELLO LOGICO**, cioè:

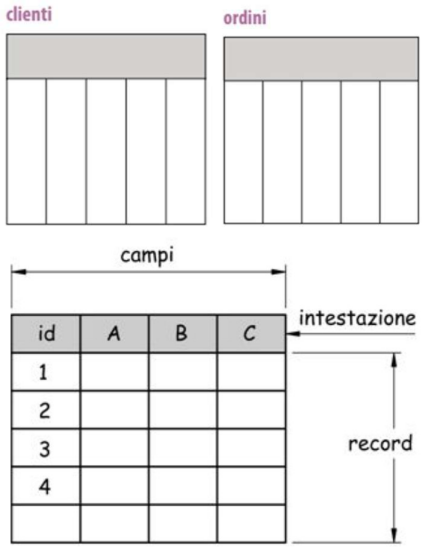
- **un insieme di concetti che utilizza per strutturare/organizzare i dati relativi ad un certo dominio d'interesse** (p.e. un dato deve avere una certa lunghezza fissa)
- **in insieme di regole che descrivono i vincoli sui singoli dati, oppure tra più dati o gruppi di dati** (p.e un dato deve essere sempre > 0 oppure una regola è la presenza contemporanea di alcuni dati come giorno, mese, anno)

Concetti e regole sono generali, e devono essere specifici per il DB, indipendentemente dall' applicazione che lo userà

MODELLO RELAZIONALE

Il modello relazionale è un modello logico che usa come struttura base dei dati la TABELLA o relazione

Il modello relazionale consiste quindi in un insieme di tabelle connesse, e si basa sul concetto matematico delle **relazioni insiemistiche**



Ogni tabella:

- **è composta da righe e colonne:**
- **le colonne sono i diversi campi (o attributi, o proprietà)**
- **ogni riga rappresenta un record**
- **ha una chiave primaria, che identifica in maniera univoca una riga della tabella**

Nel modello relazionale esistono relazioni tra due o più campi della tabella che hanno **vincoli d'integrità (i vincoli d'integrità** sono regole che garantiscono la correttezza e la coerenza dei dati in un database).

Il DB relazionale rispetta l'indipendenza sia logica, sia fisica

SQL - DDL

LINGUAGGIO

Il linguaggio standard del modello relazionale è **SQL (Structured Query Language)**

SQL:

- non è un linguaggio di programmazione;
- è un linguaggio che consente di interagire con i dati presenti nei DB, cioè permette di **creare, gestire, interrogare dei dati** di un DB;
- può essere suddiviso in 3 (+1) sottolinguaggi:
    1. **DDL – Data Definition Language**
    2. **DML – Data Manipulation Language**
    3. **QL – Query Language**
    4. **DCL – Data Control Language**: usato per gestire i permessi di accesso alle risorse del DB

DDL – Data Definition Language

Il DDL è la parte del linguaggio SQL che comprende tutti i comandi per la creazione di tabelle e la definizione dei dati in esse contenuti.
Con il linguaggio DDL si definiscono/modificano le strutture delle relazioni dello schema del DB

DML – Data Manipulation Language

Il DML è la parte del linguaggio SQL che contiene tutti i comandi per la modifica dei dati contenuti nelle tabelle (aggiunta – eliminazione – modifica – aggiornamento dati)

QL – Query Language

Il QL è la parte del linguaggio SQL che contiene i comandi che consentono di interrogare, raggruppare, filtrare, conteggiare, visualizzare (con viste personalizzate) i dati presenti nelle varie tabelle del DB

DCL – Data Control Language

Il DCL è la parte del linguaggio SQL che permette di gestire i permessi di accesso alle risorse del DB

SQL - DDL

OPERAZIONI con linguaggio DDL

CREATE DATABASE nome_database
Questa query **crea un database** chiamato *nome_database*

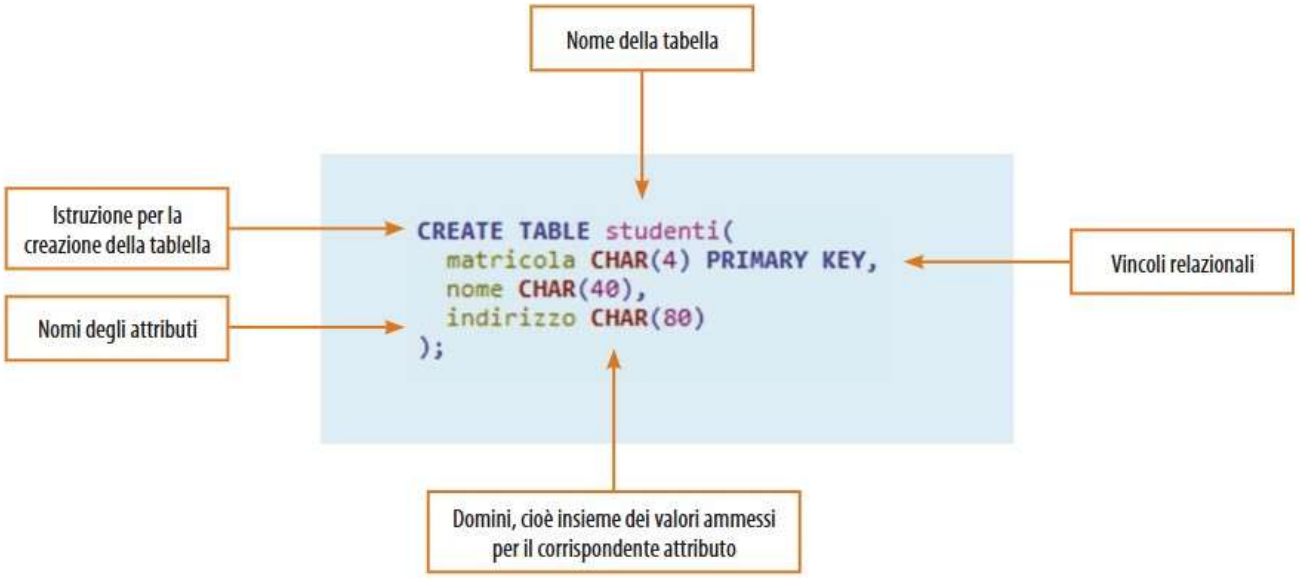
DROP DATABASE nome_database
Questa query **elimina il database** chiamato *nome_database*

CREATE TABLE nomeTabella (
nomeAttributo1 tipo [Default][vincoli],
nomeAttributo2 tipo [Default][vincoli],
. . .
);
Questa query **crea una tabella** chiamata *nomeTabella*.

Per ogni attributo della tabella si indica:

- il nome
- il dominio (tipo, eventuale lunghezza, ...)
- un eventuale valore di Default
- eventuali Vincoli di varia natura sul dato

ESEMPIO DI TABELLA DEL DB Scuola



Nome tabella: studenti
Attributi: matricola - nome - indirizzo

SQL - DDL

DOMINIO E TIPI DI DATI

Il **Dominio di un attributo** è l'insieme dei valori che possono essere presenti in una colonna.

Questi valori sono definiti in base al **tipo di dati:**

- **tipi di dati numerici**
- **tipi di dati stringhe di caratteri**
- **tipi di dati temporali**
- **tipi di dati Booleano**

TIPI DI DATI NUMERICI

I dati numerici sono di 2 tipi:

- **interi**
- **decimali**

Numeri interi

è possibile definire attributi con tipi di dati numerici interi (cioè senza decimali), che possono essere di dimensioni differenti, dipendenti dalla rappresentazione interna del calcolatore:

- **INT**
- **SMALLINT**
- **BIGINT**

La differenza è nello spazio in memoria allocato per il singolo dato intero, che influisce sul range rappresentabile

inoltre è anche possibile **indicare l'assenza di segno** con la clausola **UNSIGNED.**
p.e.

```
CREATE TABLE Prodotti (
...
quantita INT UNSIGNED,
...
);
```

TIPO	BYTE	Range SIGNED (circa)	Range UNSIGNED (circa)
SMALLINT	2	-32 mila → + 32 mila	0 → 65 mila
INT	4	-2 mld → + 2 mld	0 → 4 mld
BIGINT	8	-2^{64} → + 2^{64}	grandissimo

SQL - DDL

Sui **tipi di dato interi** si può applicare il vincolo **AUTO_INCREMENT**, che, quando un attributo intero non è specificato durante l'aggiunta di un record, prende **il valore massimo presente in tabella per quell'attributo e ci aggiunge 1** per riempire il campo del nuovo oggetto

CREATE TABLE Prodotti (
ID INT AUTO_INCREMENT, ⟶ **[PRIMARY KEY]**
quantita INT UNSIGNED,
prezzo DECIMAL(6,2),
...
);

id	nome
1	Anna
2	Luca
3	Giulia

`id` ha `AUTO_INCREMENT`

```
DELETE FROM utenti WHERE id = 2;
```

id	nome
1	Anna
3	Giulia

```
INSERT INTO utenti VALUES (NULL, 'Marco');
```

id	nome
1	Anna
3	Giulia
4	Marco

`AUTO_INCREMENT` **non** *riempie i buchi!*

Numeri decimali

è possibile definire attributi con tipi di dati numerici in cui vengono specificati il numero di cifre totali e le eventuali cifre decimali:
DECIMAL (n,d) → un numero di **n** cifre, di cui **d** sono cifre decimali

```
CREATE TABLE Prodotti (
...
quantita INT UNSIGNED,
prezzo DECIMAL(6,2),
...
);
```

```
- NUMERIC(4)      // numeri interi tra -9999 e +9999
- NUMERIC(4.1)    // numeri reali tra -999.9 e +999.9
- NUMERIC(4.2)    // numeri reali tra -99.99 e +99.99
```

`NUMERIC = DECIMAL`

Ci sono anche i **numeri a virgola mobile**:

- **FLOAT (o REAL)** = sigle precision
- **DOUBLE** = double precision

SQL - DDL

TIPI DI DATI STRINGHE DI CARATTERI

Le **stringhe di caratteri** (o semplicemente **stringhe**) possono essere di lunghezza fissa oppure una lunghezza variabile, con dimensione minima di un carattere:

- **CHAR (n)**: definisce attributi con lunghezza **fissa** di n caratteri
- **VARCHAR (n)**: definisce attributi con lunghezza **variabile** con un numero massimo n di caratteri

Nel tipo VARCHAR l'occupazione totale di memoria richiede uno o due byte aggiuntivi utilizzato dal DBMS come prefisso (serve perché il DBMS sappia quanto è lunga la stringa)

```
CREATE TABLE Prodotti (
ID INT AUTO_INCREMENT,
nome VARCHAR (50),
quantita INT UNSIGNED,
prezzo DECIMAL (6,2),
...
);
```

CHAR vs VARCHAR

ESEMPIO

Definiamo due attributi di tipo stringa:

```
– parola1 CHAR(6)          // fisso 6 caratteri
– parola2 VARCHAR(6)       // massimo 6 caratteri (secondo la lunghezza della parola)
```

assegniamo alcuni valori e osserviamo l'occupazione:

valori	parola1	spazio occupato	parola2	spazio occupato
'x'	'x '	6 byte	'x'	2 byte
'SOS'	'SOS '	6 byte	'SOS'	4 byte
'amico'	'amico '	6 byte	'amico'	6 byte
'fantasia'	'fantas'	6 byte	'fanta'	6 byte

Si può notare, per esempio:

- la parola 'X' (formata da 1 carattere) in CHAR occupa 6 byte, mentre in VARCHAR occupa 2 byte
- la parola 'SOS' (formata da 3 caratteri) in CHAR occupa 6 byte, mentre in VARCHAR occupa 4 byte

SQL - DDL

TIPI DI DATI TEMPORALI

I **tipi di dati temporali** vengono utilizzati per la memorizzazione del tempo, e sono:

- **DATE**: 10 caratteri composta da tre parti, cioè ha componenti '**AAAA-MM-GG**'
- **TIME**: 8 caratteri almeno, strutturato con componenti '**[-] [h] hh:mm:ss**'
- **DATETIME** 19 caratteri, strutturato '**AAAA-MM-GG hh:mm:ss**'
- **TIMESTAMP**: è un timestamp del momento attuale, ha la struttura di **DATETIME**

Il **TIMESTAMP** è molto importante per dare a tutti i record una validazione con data certa

Il DBMS provvede automaticamente alla **validazione** delle date, cioè accetta solo valori coerenti in base alla durata dei singoli mesi, riconoscendo gli anni bisestili in base ad un calendario perpetuo.

```
CREATE TABLE Prodotti (
ID INT AUTO_INCREMENT,
nome VARCHAR(50),
quantita INT UNSIGNED,
prezzo DECIMAL(6,2),
scadenza DATE,
...
);
```

Esistono comunque varie formattazioni accettabili per le stringhe che rappresentano il tempo

Consigli sull'utilizzo

- **DATE**: Una data schietta, come ad esempio compleanni, scadenze
- **TIME**: time rappresenta orari (di apertura, chiusura, ecc...), ma anche durate (percorrenze, ecc...). Può essere negativo
- **DATETIME:** Ideale per eventi precisi (appuntamenti, ecc...)
- **TIMESTAMP**: è un timestamp, di solito si usa per marchiare il momento della creazione o modifica di un record, o altri eventi significativi da tenere nel DB.
  Il suo unico valore legale è **CURRENT_TIMESTAMP**, che rappresenta l'istante attuale.

SQL - DDL

TIPI DI DATI BOOLEANI

I **tipi di dato Booleani** sono dei **tipi di dato binario**, e rappresentano i valori **TRUE** (vero) e **FALSE** (falso)

BOOLEAN: è il tipo booleano, i suoi valori legali sono **TRUE** e **FALSE**

```
CREATE TABLE Prodotti (
ID INT AUTO_INCREMENT,
nome VARCHAR(50),
quantita INT UNSIGNED,
prezzo DECIMAL(6,2),
in_stock BOOLEAN
);
```


*[NOTA: In realtà per il DBMS **BOOLEAN** in memoria è un **TINYINT** cioè un intero su di un byte nel quale **0** rappresenta **FALSE** ogni altro numero rappresenta **TRUE**.*
Di conseguenza anche i valori numeri in range sono legali]

SQL - DDL

VINCOLI DI BASE

I **VINCOLI** sono delle caratteristiche essenziali dei dati, che quando si crea una tabella possono essere aggiunti per completare la definizione degli attributi, e per permettere la corretta memorizzazione di dati

I **VINCOLI** possono essere di tipo:

- **intrarelazionali:** vincoli esistenti su tutti i campi di un'unica relazione, e devono essere rispettati da tutte le istanze di quel dominio o attributo
  - **sui dati** (valori di default, NULL, NOT NULL, CHECK, UNIQUE)
  - **di chiave** (PRIMARY KEY)
- **interrelazionale:** vincoli esistenti tra due differenti relazioni, e sostanzialmente riguardano le chiavi esterne (FOREIGN KEY)

VINCOLI DI TIPO INTRARELAZIONALE – sui dati

DEFAULT: indica il valore di default di un determinato campo

```
CREATE TABLE Prodotti (
ID INT AUTO_INCREMENT,
nome VARCHAR(50),
quantita INT UNSIGNED DEFAULT 0,
prezzo DECIMAL(6,2),
in_stock BOOLEAN
);
```

NOT NULL : indica che il campo non può accettare valori nulli, né per inserimento, né per modifica

Se non è specificato il valore NOT NULL, il valore di default è **NULL**

```
CREATE TABLE Prodotti (
ID INT NOT NULL AUTO_INCREMENT,
nome VARCHAR(50) NOT NULL,
quantita INT UNSIGNED DEFAULT 0,
prezzo DECIMAL(6,2),
in_stock BOOLEAN
);
```

SQL - DDL

CHECK: con questo vincolo si limita il valore per ogni attributo
Quando la condizione **CHECK è vera, MySQL accetta il valore**
p.e. CHECK (prezzo >= 0)
CHECK (VALUE >= 1 AND value <=10)

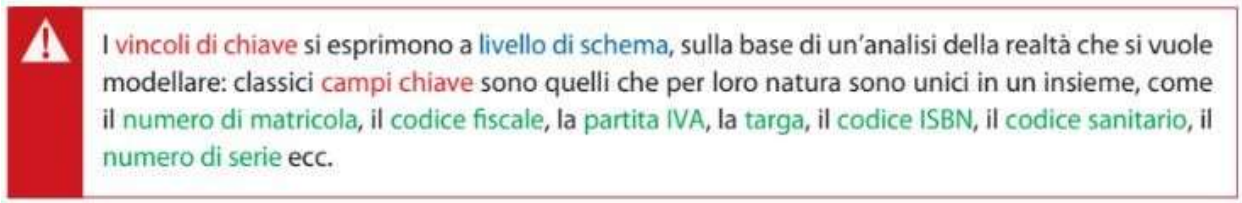
CHECK può essere inserito:

- subito dopo all'attributo a cui si fa riferimento quando riguarda un solo attributo
- alla fine della CREATE TABLE


```
CREATE TABLE Prodotti (
ID INT NOT NULL AUTO_INCREMENT ,
nome VARCHAR(50) NOT NULL,
quantita INT UNSIGNED DEFAULT 0,
prezzo DECIMAL(6,2) CHECK (prezzo >= 0),
in_stock BOOLEAN,
categoria VARCHAR(20) NOT NULL,
CHECK (categoria IN ('alimentare', 'elettronica', 'abbigliamento'))
);
```

UNIQUE: con questo vincolo, che può essere riferito ad un singolo campo oppure a più campi, si pone un vincolo di unicità – questo permette di definire attributi che **identificano la tupla** (la riga, il record) che, quindi, diventano una **chiave candidata** per quella tabella

L'attributo chiave è un attributo che si inserisce in una tabella per fare in modo che che quell'attributo sia UNICO per quella tupla, e quindi diverso in ciascuna tupla



Esempio di UNIQUE su più campi:

```
CREATE TABLE Prodotti (
ID INT NOT NULL UNIQUE AUTO_INCREMENT ,
nome VARCHAR(50) NOT NULL UNIQUE,
quantita INT UNSIGNED DEFAULT 0,
prezzo DECIMAL(6,2) CHECK (prezzo >= 0),
in_stock BOOLEAN,
categoria VARCHAR(20) NOT NULL,
CHECK (categoria IN ('alimentare', 'elettronica', 'abbigliamento'))
);
```

SQL - DDL

Esempio di UNIQUE a fine tabella:

CREATE TABLE Prodotti (
ID INT NOT NULL UNIQUE AUTO_INCREMENT,
nome VARCHAR(50) NOT NULL,
quantita INT UNSIGNED DEFAULT 0,
prezzo DECIMAL(6,2) CHECK (prezzo >= 0),
in_stock BOOLEAN,
categoria VARCHAR(20) NOT NULL,
CHECK (categoria IN ('alimentare', 'elettronica', 'abbigliamento')),
UNIQUE (nome, categoria)
);
In pratica significa: **NON POSSONO ESSERCI TABELLE CON LA STESSA COPPIA "nome, categoria)**

Altro esempio di UNIQUE:

```
1   /* singolo attributo   */
2   CREATE TABLE dipartimenti(
3     nome_dipartimento CHAR(15) UNIQUE
4   )
5
6   /* coppia di attributi */
7   CREATE TABLE docenti(
8     cognome CHAR(30) NOT NULL,
9     nome CHAR (20) NOT NULL,
10    UNIQUE (cognome, nome)
11  )
```

Mettere la condizione di unicità sulla coppia di attributi è differente che porla singolarmente sui due attributi, come possiamo constatare negli esempi riportati nella tabella che segue.

Sui singoli attributi	Esito	Sulla coppia	Esito
'Rossi' 'Mario'	ammesso	'Rossi Mario'	ammesso
'Rossi' 'Gino'	rifiutato	'Rossi Gino'	ammesso
'Verdi' 'Mario'	rifiutato	'Verdi Mario'	ammesso

Se il vincolo UNIQUE è posto singolarmente sui due attributi "ROSSI" e "MARIO", i due attributi sono accettati una sola volta, e mai più ne solo come ROSSI, ne solo come MARIO

Se il vincolo UNIQUE è posto sulla coppia di attributi ROSSI MARIO, ROSSI MARIO è accettato una sola volta, ma la tabella accetta anche solo ROSSI o solo MARIO abbinati ad altri attributi

SQL - DDL

VINCOLI DI TIPO INTRARELAZIONALE – di chiave

La CHIAVE PRIMARIA o **PRIMARY KEY (PK)** è la chiave (cioè l'attributo chiave) prescelta a livello logico per la tabella

IMPORTANTE :

- **può essere presente una sola dichiarazione di PRIMARY KEY per tabella**
- **i vincoli NOT NULL e UNIQUE sono impliciti nella clausola PRIMARY KEY**

La **CHIAVE PRIMARIA** si indica graficamente in grassetto, oppure sottolineata oppure con affianco la sigla PK

CREATE TABLE Prodotti (
ID INT PRIMARY KEY AUTO_INCREMENT ,
nome VARCHAR(50) NOT NULL UNIQUE,
quantita INT UNSIGNED DEFAULT 0,
prezzo DECIMAL(6,2) CHECK (prezzo >= 0),
in_stock BOOLEAN,
categoria VARCHAR(20) NOT NULL,
CHECK (categoria IN ('alimentare', 'elettronica', 'abbigliamento'))
);

CHIAVE PRIMARIA COMPOSTA: la chiave primaria composta è una chiave primaria costituita da più attributi presi insieme

```
1  CREATE TABLE interrogazioni(
2    id_studente CHAR(20),
3    id_materia INT,
4    data_interrogazione DATE,
5    voto INT DEFAULT 1 CHECK (voto >= 1 AND voto <= 10),
6    PRIMARY KEY(id_studente, id_materia, data_interrogazione)
7  )
```

SQL - DDL

VINCOLI DI TIPO INTERRELAZIONALE

I vincoli interrelazionali sono vincoli esistenti tra due differenti relazioni, che si chiamano:

- **relazione esterna**: è la tabella che contiene l'insieme delle chiavi, dove è definito l'attributo
- **relazione interna**: è la tabella contenente la **FOREIGN KEY** di collegamento alla tabella esterna, che prende quindi il nome di CHIAVE ESTERNA

Una **FOREIGN KEY** è un vincolo che collega **una colonna (o più colonne) di una tabella (la tabella interna)** a **uno (o più) campi di una *tabella esterna***

Serve a garantire **integrità referenziale**, cioè che i valori nella tabella esistano nella tabella esterna.

E' necessario che la colonna referenziata sia **PRIMARY KEY** o **UNIQUE** nella tabella esterna, altrimenti non si saprebbe quale riga associare al riferimento

FOREIGN KEY fa uso del keyword **REFERENCES** per specificare a cosa il campo FK faccia riferimento, con la struttura *tabella(campo1, campo2, ...)*

```
CREATE TABLE Categorie (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(50) NOT NULL UNIQUE,
    CHECK (nome IN
      ('alimentare', 'elettronica', 'abbigliamento'))
);
```

```
CREATE TABLE Prodotti (
      ID INT PRIMARY KEY AUTO_INCREMENT ,
      nome VARCHAR(50) NOT NULL UNIQUE,
      quantita INT UNSIGNED DEFAULT 0,
      prezzo DECIMAL(6,2) CHECK (prezzo >= 0),
      in_stock BOOLEAN,
      id_Categoria INT,
      FOREIGN KEY (id_Categoria) REFERENCES Categorie(ID)
      );
```

Esempio 1 – chiave esterna formata da 1 attributo

docenti

ID_docente	cognome	nome	dipartimento
123	Russo	Antonio	Informatica
132	Verdi	Gino	Fisica
321	Rossi	Mario	Matematica

Tabella interna

dipartimenti

nome_dipartimento
Informatica
Fisica
Matematica

Tabella esterna

La codifica è la seguente:

```
1 CREATE TABLE dipartimenti(
2   nome_dipartimento CHAR(15) PRIMARY KEY
3 )
```

```
1   CREATE TABLE docenti(
2     ID_docente INT NOT NULL PRIMARY KEY,
3     nome CHAR(20) NOT NULL,
4     cognome CHAR(20) NOT NULL,
5     dipartimento CHAR(15) REFERENCES dipartimenti(nome_dipartimento)
6   )
7
```

SQL - DDL

Esempio 1 – chiave esterna formata da 2 attributi

studenti

ID_docente	cognome	nome	classe	sezione
R123	Russo	Antonio	5	M1
IT123	Verdi	Gino	5	M1
IT321	Rossi	Mario	5	Info3

Tabella Interna

corsi

classe	sezione	specializzazione
5	M1	Meccanica
3	Info1	Informatica
4	Info1	Informatica
5	Info1	Informatica

Tabella esterna

La codifica è la seguente:

```
1 CREATE TABLE corsi(
2   classe INT,
3   sezione CHAR(4),
4   nome_corso CHAR(25),
5   PRIMARY KEY(classe, sezione)
6 )
```

```
1 CREATE TABLE studenti(
2   matricola CHAR(20) PRIMARY KEY,
3   nome VARCHAR(20),
4   cognome VARCHAR(20),
5   data_nascita DATE,
6   classe INT,
7   sez CHAR(4),
8   FOREIGN KEY(classe, sez) REFERENCES corsi(classe, sez)
9 )
```

Quando si dichiara un campo come **FOREIGN KEY**, si possono specificare due clausole:

- **ON DELETE**: descrive qual è il comportamento da attuare se la riga a cui la FK fa riferimento viene **eliminata** nella tabella esterna
- **ON UPDATE**: descrive qual è il comportamento da attuare se la riga a cui la FK fa riferimento viene **modificata** nella tabella esterna

Se, per esempio, venisse cancellato il dipartimento di fisica, quale conseguenza ne deriverebbe per la chiave esterna presente nella tabella docenti?

docenti

ID_docente	cognome	nome	dipartimento
123	Russo	Antonio	Fisica
132	Verdi	Gino	Informatica
321	Rossi	Mario	Informatica

Tabella Interna

dipartimenti

nome_dipartimento
Informatica
Fisica
Matematica

Tabella esterna

Queste clausole sono seguite da una fra le seguenti parole chiave:
RESTRICT, SET NULL, SET DEFAULT, CASCADE

SQL - DDL

RESTRICT

la parola chiave **RESTRICT impedisce l'eliminazione o l'aggiornamento della riga padre se ci sono dei record figli collegati** – in pratica con RESTRICT non si può eliminare una categoria se esistono dei prodotti associati a quella categoria

```
CREATE TABLE Categorie (
ID INT AUTO_INCREMENT PRIMARY KEY,
nome VARCHAR(50) NOT NULL UNIQUE,
);


CREATE TABLE Prodotti (
ID INT PRIMARY KEY AUTO_INCREMENT ,
nome VARCHAR(50) NOT NULL UNIQUE,
...
id_Categoria INT,
FOREIGN KEY (id_Categoria) REFERENCES Categorie(ID)
ON DELETE RESTRICT
);
```

SET NULL

la parola chiave SET NULL imposta il valore della colonna a NULL quando la riga della tabella referenziata viene eliminata oppure aggiornata

```
CREATE TABLE Categorie (
ID INT AUTO_INCREMENT PRIMARY KEY,
nome VARCHAR(50) NOT NULL UNIQUE,
);

CREATE TABLE Prodotti (
ID INT PRIMARY KEY AUTO_INCREMENT ,
nome VARCHAR(50) NOT NULL UNIQUE,
...
id_Categoria INT,
FOREIGN KEY (id_Categoria) REFERENCES Categorie(ID)
ON DELETE SET NULL
);
```

Se elimini una categoria, il campo id_Categoria dei Prodotti che appartenevano a quella categoria conterrà NULL

SQL - DDL

SET DEFAULT

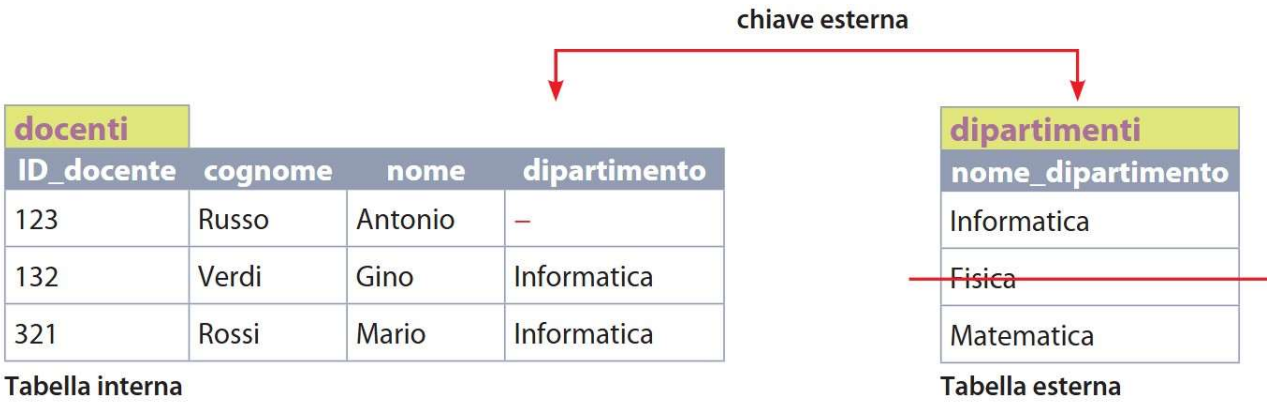
Con la parola chiave **SET DEFAULT** si imposta il valore della colonna al proprio valore di default quando la riga della tabella esterna referenziata viene eliminata oppure aggiornata

La colonna deve avere un valore di DEFAULT definito

```
CREATE TABLE Categorie (
ID INT AUTO_INCREMENT PRIMARY KEY,
nome VARCHAR(50) NOT NULL UNIQUE,
);

CREATE TABLE Prodotti (
ID INT PRIMARY KEY AUTO_INCREMENT ,
nome VARCHAR(50) NOT NULL UNIQUE,
...
id_Categoria INT DEFAULT -1,
FOREIGN KEY (id_Categoria) REFERENCES Categorie(ID)
ON DELETE SET DEFAULT
);
```

Se elimini una categoria, il campo id_Categoria dei Prodotti che appartenevano a quella categoria conterrà -1



CASCADE

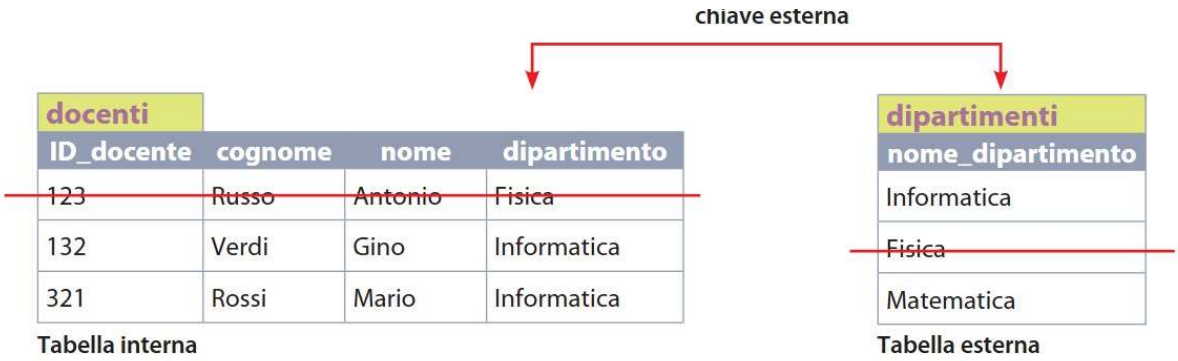
La parola chiave CASCADE propaga l'azione di modifica oppure cancellazione dell'attributo di riferimento alla tabella interna collegata

```
CREATE TABLE Categorie (
ID INT AUTO_INCREMENT PRIMARY KEY,
nome VARCHAR(50) NOT NULL UNIQUE,
);

CREATE TABLE Prodotti (
ID INT PRIMARY KEY AUTO_INCREMENT ,
nome VARCHAR(50) NOT NULL UNIQUE,
...
id_Categoria INT,
FOREIGN KEY (id_Categoria) REFERENCES Categorie(ID)
ON DELETE CASCADE,
ON UPDATE CASCADE,
);
```

Se modifichi una categoria, il campo id_Categoria dei Prodotti che appartenevano a quella categoria verrà aggiornato di conseguenza

Se elimini una categoria, i prodotti che avevano nel campo id_Categoria il riferimento alla categoria eliminata, verranno eliminati a loro volta



REGOLE DI INSERIMENTO DI UNA RIGA IN UNA TABELLA

Cosa succede se voglio inserire una riga in una tabella, con un campo FK, una riga che avrebbe un valore per quel campo, che non esiste nella tabella esterna?

```
CREATE TABLE Prodotti (
ID INT PRIMARY KEY AUTO_INCREMENT ,
nome VARCHAR(50) NOT NULL UNIQUE,
id_Categoria INT,
FOREIGN KEY (id_Categoria) REFERENCES Categorie(ID)
ON DELETE CASCADE,
ON UPDATE CASCADE,
);
```

CATEGORIE	
ID	**nome**
0	elettronica
1	abbigliamento

```
INSERT INTO Prodotti (nome, id_Categoria) VALUES ( "Pane Integrale", 2);
```

Cosa succede?
Banalmente, l' INSERT viene bloccato con un messaggio di errore

SQL - ALTER

Esistono delle operazioni del linguaggio DDL che si possono pensare come intermedie tra il linguaggio DDL è il linguaggio DML: queste operazioni si chiamano **ALTER**

- **Il linguaggio SQL – DDL** comprende tutti i comandi per la creazione di tabelle, la definizione dei dati, attributi, vincoli
- **Il linguaggio SQL – DML** comprende tutti i comandi per la modifica dei dati in tabella (inserimento, eliminazione, modifica, aggiornamento)
- **Le operazioni ALTER** sono operazioni del linguaggio SQL-DDL che agiscono sulla struttura della tabella, non sui contenuti; sono operazioni di modifica, non di creazione delle definizioni.

ALTER TABLE

ALTER TABLE è un'istruzione che serve per modificare la struttura di una tabella già esistente, senza cancellarla.

Consente di:

- aggiungere colonne (ADD)
- modificare colonne (MODIFY)
- eliminare colonne o tabelle (DROP)
- aggiungere o rimuovere vincoli (ADD.....)
- rinominare colonne o tabelle (RENAME)

```
ALTER TABLE nome_tabella
operazione1,
operazione2;
```

ADD – aggiungere una colonna

Serve per aggiungere una colonna alla tabella

Esempio 1: Aggiunge la colonna **descrizione** alla tabella **Prodotti**

```
ALTER TABLE Prodotti
ADD descrizione TEXT;
```

Esempio 2 : Con alcuni DBMS si può specificare la posizione in cui inserire la nuova colonna
Aggiunge la colonna **descrizione** alla tabella **Prodotti,** dopo la tabella **nome**

```
ALTER TABLE Prodotti
ADD descrizione TEXT AFTER nome;
```

SQL - ALTER

MODIFY – modificare una colonna

Serve per cambiare **tipo**, **dimensione** o **vincoli** di una colonna

```
ALTER TABLE Prodotti
MODIFY quantita INT UNSIGNED DEFAULT 0;
ALTER TABLE Prodotti
MODIFY prezzo DECIMAL(8,2) NOT NULL;
```

ATTENZIONE:

SOSTITUISCE COMPLETAMENTE la definizione della colonna: tutti i vincoli devono essere rispecificati.

Esempio :

situazione iniziale:

```
prezzo DECIMAL(8,2) NOT NULL DEFAULT 0
```

eseguo MODIFY:

```
ALTER TABLE Prodotti
MODIFY prezzo DECIMAL(10,2)
```

risultato finale

```
prezzo DECIMAL(10,2)
```

Ho perso **NOT NULL DEFAULT 0**

DROP – elimina una tabella o una colonna di tabella

Serve per **cancellare una tabella** oppure **una colonna di tabella**

ATTENZIONE: il DROP è IRREVERSIBILE

```
DROP TABLE nome_tabella; DROP TABLE Prodotti, Clienti;

ALTER TABLE Prodotti
DROP COLUMN quantità;
```

DROP viene negato se una colonna o tabella è implicata in **un vincolo di chiave esterna**

In tal caso occorre prima fare DROP sul vincolo stesso

```
ALTER TABLE Prodotti
DROP FOREIGN KEY fk_categoria;
```

SQL - ALTER

ADD – aggiungere un vincolo

Con ADD si possono anche aggiungere nuovi vincoli sulla tabella

Esempio 1 - Aggiunge unicità sulle coppie **nome-categoria:**

```
ALTER TABLE Prodotti
ADD UNIQUE (nome, categoria_id);
```

Esempio 2 - Aggiunge chiave primaria su **ID** alla tabella:

```
ALTER TABLE Clienti
ADD PRIMARY KEY (ID);
```


Esempio 3 - Aggiunge un controllo su **prezzo**:

```
ALTER TABLE Prodotti
ADD CHECK (prezzo >= 0);
```

ATTENZIONE: I dati già presenti nella tabella devono essere coerenti con i nuovi vincoli, altrimenti l'ALTER fallisce!
esempio: se nella tabella sono presenti righe con campo **prezzo** *con valore minore di 0, l'ultimo* **ALTER** *fallisce.*

Esempio 4 - Aggiunge un vincolo di chiave esterna FOREIGN KEY:

```
ALTER TABLE Prodotti
ADD CONSTRAINT fk_categoria      ──→ nome del vincolo
FOREIGN KEY (categoria_id)
REFERENCES Categorie(ID)         ──→ espressione del vincolo
ON DELETE SET NULL
ON UPDATE CASCADE;               ──→ eventuali clausole
```

ATTENZIONE : Anche qui, occorrono dati coerenti

SQL - ALTER

RENAME – rinominare una tabella o una colonna di tabella

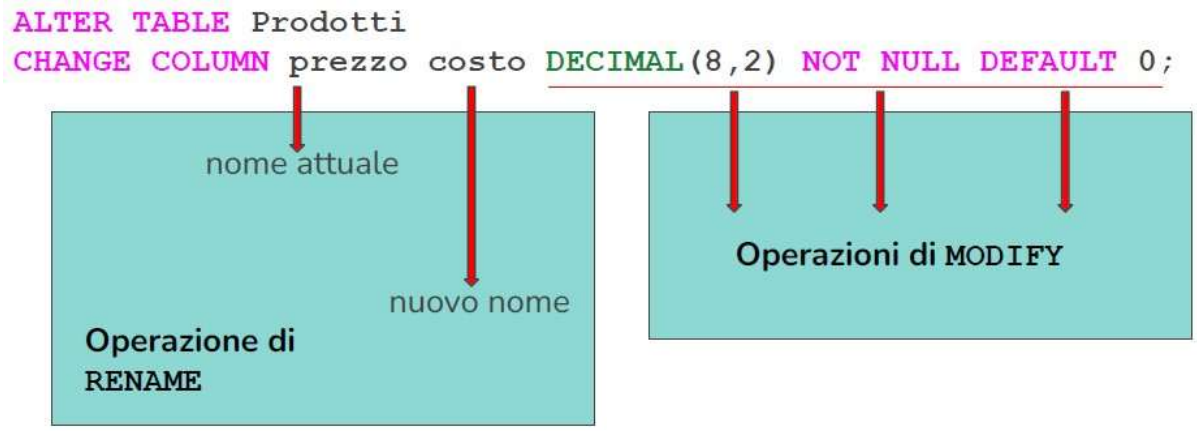
RENAME TO – rinomina tabella:

```
ALTER TABLE nome_vecchio
RENAME TO nome_nuovo;
```

RENAME COLUMN – rinomina colonna di tabella:

```
ALTER TABLE nome_tabella
RENAME COLUMN nome_vecchio_colonna TO
nome_nuovo_colonna;
```


Non tutti i **DBMS** supportano il **RENAME COLUMN** – in tal caso si usa **CHANGE COLUMN,** che effettua contemporaneamente le operazioni codificate d**a MODIFY** e **RENAME COLUMN**



SQL - DML

DML – Data Manipulation Language

Il DML è la parte del linguaggio SQL che contiene tutti i comandi per la modifica dei dati contenuti nelle tabelle (aggiunta – eliminazione – modifica – aggiornamento dati)

Nel linguaggio DML ci sono quindi 3 costrutti:

- **INSERT**
- **DELETE**
- **UPDATE**

In sintesi:

- **il linguaggio DDL** ha come scopo definire e modificare le strutture, lo schema logico delle tabelle e delle colonne
- **il linguaggio DML** ha come scopo la modifica dei dati presenti nelle tabelle, cioè delle righe

---

SQL - DML

INSERT

INSERT è l'operazione che permette l'aggiunta di record (righe) in una tabella

Esempio 1:

Inserimento di una riga

```
INSERT INTO nome_tabella (colonna1, colonna2, ...)
VALUES (valore1, valore2, ...);
```

valori da dare ai campi inseriti

campi da riempire

Esempio 2:

Inserimento di più righe insieme, separate da virgole

```
INSERT INTO Prodotti (nome, prezzo)
VALUES
('Bicicletta', 150.00),
('Casco', 30.00),
('Guanti', 15.00);
```

NOTA: ovviamente INSERT fallisce se viene violato un vincolo presente sulla tabella.

```
Per esempio:
ID INT PRIMARY KEY ,
nome VARCHAR(50) NOT NULL UNIQUE,
prezzo DECIMAL(6,2) CHECK (prezzo >= 0),
);
```

```
INSERT INTO Prodotti (nome, prezzo)
VALUES ('Sellino', -10.00)
```

→ **INSERT fallito**

Esempio 3:

Si possono totalmente omettere i campi da riempire se si passano tutti i valori per tutti i campi della tabella

```
INSERT INTO nome_tabella VALUES (valore1, valore2, ...);
```

Esempio 4:

Se si omettono campi con valori di DEFAULT o AUTO_INCREMENT, questi vengono riempiti automaticamente

```
CREATE TABLE Clienti (
ID INT AUTO_INCREMENT PRIMARY KEY,
nome VARCHAR(50) NOT NULL,
email VARCHAR(100) UNIQUE,
data_registrazione DATE DEFAULT CURRENT_DATE
);

INSERT INTO Clienti (nome, email)
VALUES ('Mario Rossi', 'mario@example.com');
```

SQL - DML

DELETE

DELETE è l'operazione che permette di eliminare dati da una tabella
Si usa insieme al Keyword **WHERE**

```
DELETE FROM nome_tabella
WHERE condizione;
```

```
DELETE FROM Clienti
WHERE ID = 2;
```

```
DELETE FROM Clienti
WHERE nome = 'Mario' AND cognome = 'Rossi';
```

A `WHERE` segue una condizione, che **può essere anche composta**, come abbiamo visto in algebra relazionale!

Omettere il `WHERE` significa eliminare ogni riga della tabella

```
DELETE FROM Clienti;
```

Cancella TUTTI i clienti

Ricordate che quando fate `DELETE` (o `UPDATE`) si applicano le clausole sulle `FOREIGN KEY` (`CASCADE`, `SET NULL`, ecc...)

UPDATE

UPDATE è l'operazione che **modifica i valori di una o più colonne per uno o più record di una tabella**

UPDATE **modifica** i valori di **una o più colonne** per **uno più record** di **una tabella**

```
UPDATE nome_tabella
SET colonna1 = valore1, colonna2 = valore2
WHERE condizione;
```

Vale quanto ricordato prima sulle condizioni composte

```
UPDATE Clienti
SET email = 'mario.rossi@example.com', paese = 'Italy'
WHERE nome = 'Mario Rossi';
```

Anche qui, se `WHERE` viene omesso, vengono modificate tutte le righe della tabella

```
UPDATE Prodotti
SET prezzo = prezzo * 1.10
WHERE categoria = 'elettronica';
```

SQL - DML

ESEMPIO RIASSUNTIVO

Vediamo un esempio completo dove utilizziamo tutte le istruzioni viste finora; dopo aver creato il database **libreria**, iniziamo con la creazione di una tabella chiamata **libri**, con chiave primaria rappresentata dal codice **isbn**, e memorizziamo i dati relativi al titolo, all'autore, all'editore, al costo, all'anno di pubblicazione.
La query di creazione è la seguente:

```
CREATE TABLE libri(
  isbn CHAR(13) PRIMARY KEY,
  titolo VARCHAR(50),
  autore VARCHAR(50),
  editore VARCHAR(50),
  anno_pub CHAR(4),
  categoria VARCHAR(50),
  prezzo NUMERIC(5) DEFAULT 0
)
```

1 Inseriamo un nuovo record nella tabella **libri** con codice **isbn** uguale a '9788836003365':

```
INSERT INTO libri VALUES
  ('9788836003365', 'TPSIT 3', 'Camagni Paolo', 'Hoepli', '2021', 'Informatica', 22.90)
```

Aggiungiamo anche un secondo libro e visualizziamo la nostra tabella:

libri | Enter a SQL expression to filter results (use Ctrl+Space)

Grid

	isbn	titolo	autore	editore	anno_pub	categoria	prezzo
1	9788836003365	TPSIT 3	Camagni Paolo	Hoepli	2021	Informatica	23,95
2	9788836003366	TPSIT 2	Camagni Paolo	Hoepli	2020	Informatica	24,55

2 Aggiungiamo alla struttura della tabella il campo **citta** di tipo **VARCHAR** di dimensione **30** caratteri:

```
ALTER TABLE libri
ADD citta VARCHAR(30)
```

Come verifica, andiamo a visualizzare la struttura della tabella **libri**:

Colonne: Aggiungi Rimuovi Su Giù

#	Nome	Tipo di dati	Lunghezza/set	Senza segno	Permetti NULL	Riempi
1	isbn	CHAR	13	☐	☐	
2	titolo	VARCHAR	50	☐	☑	
3	autore	VARCHAR	50	☐	☑	
4	editore	VARCHAR	50	☐	☑	
5	anno_pub	CHAR	4	☐	☑	
6	categoria	VARCHAR	50	☐	☑	
7	prezzo	DECIMAL	6,2	☐	☑	
8	citta	VARCHAR	30	☐	☑	

Aiuto Scarta Salva

3 Aggiorniamo tutti i record riducendo il prezzo del 20%:

```
UPDATE libri
SET prezzo = prezzo * 0.8;
```

Riportiamo di seguito l'effetto dell'operazione ai dati presenti nell'archivio:

prima	dopo
prezzo	prezzo
14,00	11,20
29,00	23,20
17,00	13,60
21,00	16,80
11,00	8,80
19,00	15,20
12,00	9,60
17,90	14,32
18,00	14,40

4 Cancelliamo tutti i **libri** che hanno un costo inferiore a € 10:

```
1   DELETE FROM libri
2   WHERE prezzo < 10;
```

e verifichiamo, infine, che non siano più presenti nella nostra tabella.

Host: 127.0.0.1 | Database: provesql | Tabella: libri | Dati | Query*

provesql.libri: 10 righe totali (circa) — Successivo | Mostra tutto | Ordinamento | Colonne

isbn	titolo	autore	editore	anno_pub	categoria	prezzo	citta
9255583600340	Harry Potter e il calice di fuoco	K.Rowling	Salani	1993	Avventura	11,20	Milano
9268844332121	Disegnare una casa	Lanterio	Zanichelli	1979	Architettura	23,20	Bologna
9288836003001	La Divina Commedia	Dante Alighieri	Signorelli	2010	Letteratura	13,60	Milano
9288836003111	Algoritmi	Camagni Paolo	Hoepli	2005	Informatica	16,80	Milano
9288868522070	Io Uccido	Giorgio Faletti	Dalai	2006	Thriller	15,20	Milano
9788836003111	PHP	Nikolassy Riccardo	Hoepli	2017	Informatica	14,32	Milano
9788836003121	Java	Nikolassy Riccardo	Hoepli	2016	Informatica	14,40	Milano
9788836003361	TPSIT 1	Camagni Paolo	Hoepli	2019	Informatica	16,80	Milano
9788836003362	TPSIT 1	Camagni Paolo	Hoepli	2019	Informatica	17,52	Milano
9788836003365	TPSIT 3	Camagni Paolo	Hoepli	2021	Informatica	19,16	Milano