

La Programmazione ad Oggetti in Python

Docente: Ambra Demontis

Anno Accademico: 2024 - 2025

Corso di Laurea in Ingegneria Elettronica, Informatica e delle
Telecomunicazioni



University of Cagliari,
Italy


Department of Electrical and
Electronic Engineering



L'utilizzo dei Database

- Installazione MySQL
- Creazione di tabelle
- Query

Installare MySQL



The world's most popular open source database

[Contact MySQL](#) | [Login](#) | [Register](#)

[MYSQL.COM](#) [DOWNLOADS](#) [DOCUMENTATION](#) [DEVELOPER ZONE](#)

[f](#) [X](#) [in](#) [You Tube](#)

HeatWave

- Integrated and automated generative AI with HeatWave GenAI
- Accelerate query performance with HeatWave MySQL
- Query data in object storage and MySQL with HeatWave Lakehouse
- Automate the machine learning pipeline with HeatWave AutoML

[Try Free](#) [Technical Guides](#)

MySQL Enterprise Edition for Developers

Free! Access the full range of MySQL Enterprise Edition features while learning, developing, and prototyping.

[Download Now »](#)

MySQL Newsletter

[Subscribe »](#)

[Archive »](#)

MySQL Enterprise Edition

MySQL Enterprise Edition includes the most comprehensive set of advanced features, management tools and technical support for MySQL.

MySQL Enterprise Edition for Developers

Installare MySQL

ORACLE Products Industries Resources Customers Partners Developers Company

MySQL > Enterprise >

MySQL Enterprise Edition Downloads

Access the full range of MySQL Enterprise Edition features for free while learning, developing, and prototyping.

Download includes MySQL Enterprise Server, Backup, Router, Shell and Connectors.

E' necessario registrarsi

Discover HeatWave MySQL, our fully managed cloud database service [Learn more](#)

MySQL Enterprise Edition

Linux **MacOS** Windows

Operating System	Architecture	File size	Download (zip file of dmg/zip packages)
Apple MacOS 15, 14	arm64	1.0 GB	mysql-enterprise-9.3.0_macos15_arm64_bundle.zip



Creare un Utente

MySQL server ha un utente già disponibile dal momento dell'installazione per effettuare il login, l'utente root@localhost.

Questo utente è un amministratore e ha tutti i privilegi (accedere/cancellare tutte tabelle, gestire utenti).

Meglio quindi crearne un altro il quale avrà meno privilegi.

Per fare questo dobbiamo prima di tutto effettuare il login con l'utente root:
`mysql -u root -p`

Creare un Utente

Creiamo il nuovo utente:

```
mysql > CREATE USER '<pyuser>' @ 'localhost'  
IDENTIFIED BY '<Py@pp4Demo>';
```

Nome utente

Password

Creazione di un Database

Definire un database:

```
CREATE DATABASE (<nome database>);
```

Creiamo un database per il nostro esempio:

```
CREATE DATABASE DB_libreria;
```

Concessione Privilegi

Concedere ad un utente tutti i permessi (creazione, modifica, selezione, cancellazione, tabelle ad un utente) su un database ad un utente:

```
GRANT ALL PRIVILEGES  
ON <database>.*  
TO '<utente>'@'localhost';
```

```
GRANT ALL PRIVILEGES  
ON DB_libreria.*  
TO 'pyuser'@'localhost';
```


Cambiare Utente

Effettuiamo il login con il nuovo utente:

```
mysql > exit;
```

```
mysql -u pyuser -p
```

Utilizziamo il database creto prima:

```
USE DB_libreria;
```

Creazione di Tabelle

I **tipi di dato** che utilizzeremo negli esempi seguenti.

Tipi **numerici esatti**:

- Interi:
INTEGER
- Con una parte decimale di lunghezza prefissata:
NUMERIC (<PRECISIONE>,<SCALA>)
PRECISIONE: totale cifre
SCALA: totale cifre dopo la virgola

Creazione di Tabelle

I **tipi di dato** che utilizzeremo negli esempi seguenti.

Stringhe:

- Di lunghezza variabile:
VARCHAR (<lunghezza massima>)
- Di lunghezza predefinita:
CHARACTER(<numero caratteri>)

Creazione di Tabelle

I **tipi di dato** che utilizzeremo negli esempi seguenti.

Tipi temporali:

- DATA il cui formato è anno-mese-giorno'
- TIME il cui formato è ore:minuti:secondi'

Creazione di Tabelle

I **tipi di dato** che utilizzeremo negli esempi seguenti.

Valori booleani:

- TRUE
- FALSE

Creazione di Tabelle

CREATE TABLE <nome>

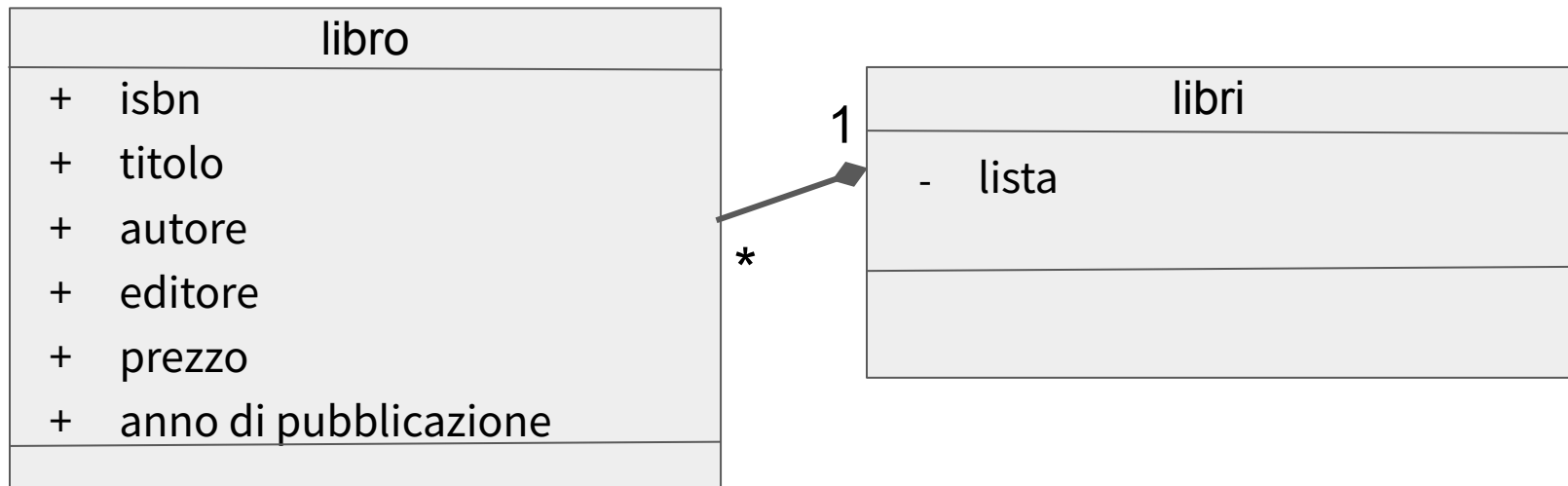
```
(  <nome colonna 1> <dominio 1> [valore default], [vincoli colonna],  
    <nome colonna 2> <dominio 2> [valore default], [vincoli colonna],  
    ...  
    [altri vincoli]  
)
```

Creazione di Tabelle

Supponiamo di voler creare le tabelle per memorizzare le informazioni riguardo ad una lista di libri.

Supponiamo di voler accertarci che il prezzo sia un valore positivo.

Il diagramma delle classi sarebbe:



Creazione di Tabelle

Ci basterà un'unica tabella.

isbn	titolo	autore	editore	prezzo	anno Pubbl
9374287969	Python 3 Object-Oriented Programming	Dusty Phillips	Packt	34,99	2018
...
1484236939	MySQL Connector	Jesper Wisborg Krogh	Apress	64,19	2018

Creazione di Tabelle

Es con **vincoli di colonna** in blu.

```
CREATE TABLE libri
(
  ISBN CHARACTER(14) PRIMARY KEY,
  titolo VARCHAR(40) NOT NULL UNIQUE,
  autore VARCHAR(20) NOT NULL,
  editore VARCHAR(20) NOT NULL,
  prezzo NUMERIC(4,2) NOT NULL CHECK (prezzo > 0),
  anno_publicazione INTEGER NOT NULL
);
```

Chiave primaria (non nulla e unica) → **PRIMARY KEY**

Unico (due righe non possono avere lo stesso valore per questo attributo) → **UNIQUE**

Deve avere un valore (non nullo) → **NOT NULL**

Per controllare che il valore rispetti i requisiti utilizziamo il vincolo CHECK che esprime una condizione che deve essere vera per tutte le righe

Inserimento Righe in Tabella

INSERT INTO <nome tabella> VALUES (<valore 1>, <valore 2>, ...)

ES:

```
INSERT INTO libri VALUES ('9374287969', 'Python 2 Object-Oriented  
Programming', 'Dusty Phillips', 'Packt', 34.99, 2018);
```

 Vanno indicati tra apici le stringhe, le date e time

Nel caso di input che violano uno dei vincoli riceveremo un errore e l'inserimento non verrà effettuato.

Eliminare una Tabella

DROP TABLE <tabella>

Creazione di Tabelle

Supponiamo di venire chiamati da un negozio di libri per aiutarli a gestire le ricevute emesse.

Supponiamo il negozio tenga già traccia:

- Di tutti i libri venduti (per ogni libro memorizza il codice ISBN, il titolo, l'autore, il prezzo;
- Dei clienti del negozio (per ogni cliente memorizza il nome, il cognome, il codice fiscale, il numero di telefono).

Creazione di Tabelle

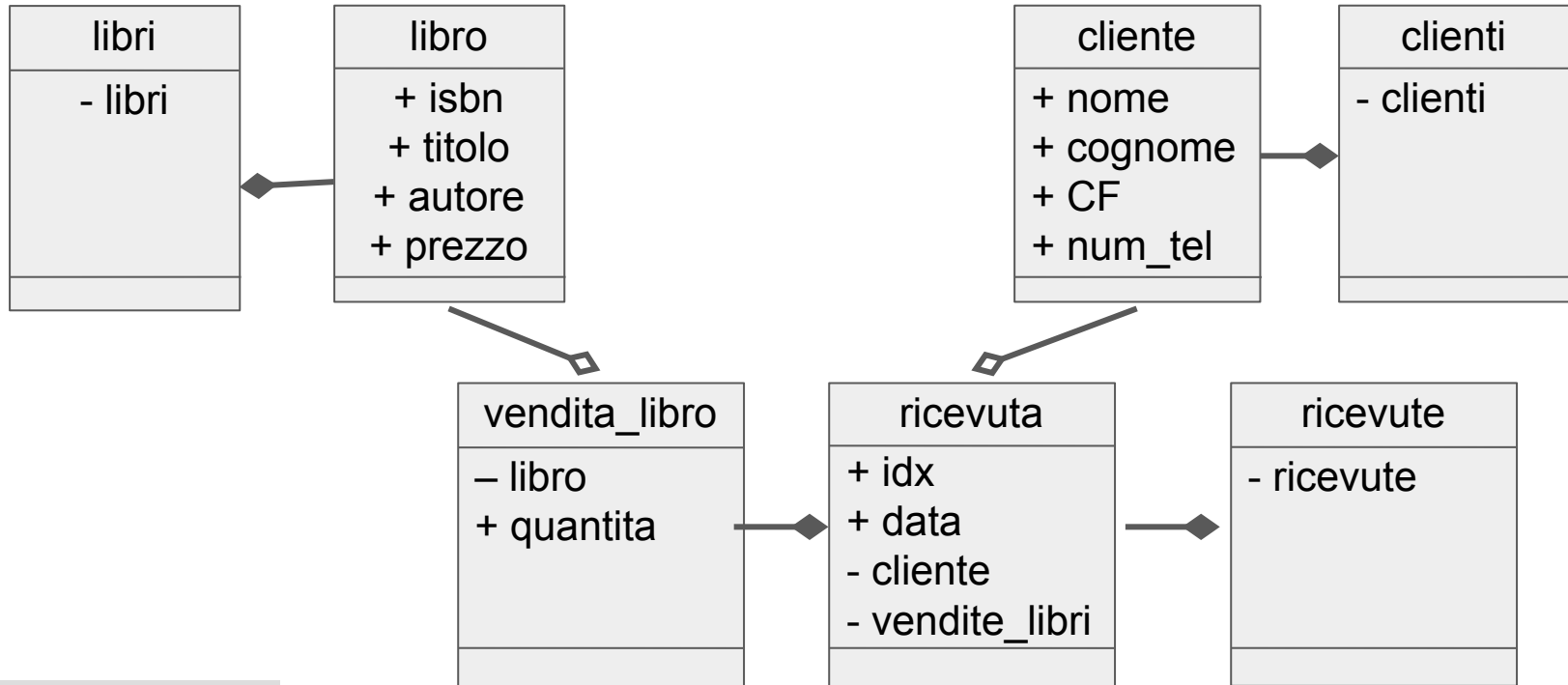
Supponiamo di voler tenere traccia delle ricevute emesse. Per ogni ricevuta, vogliamo tenere conto:

- Del suo indice;
- Della data di emissione;
- Del nome e cognome del cliente al quale l'abbiamo emessa;
- Per ogni tipologia di libro venduto vogliamo tenere conto del suo codice isbn, di quante copie di quel libro sono state vendute in una singola vendita e del prezzo del libro.

Supponiamo che i prezzi dei libri siano tutti fissati dalla casa editrice e non possano venire modificati dal negozio di libri.

Creazione di Tabelle

Il diagramma delle classi sarebbe:



Creazione di Tabelle

```
class CLibro:
```

```
    def __init__(self, isbn, titolo, autore, prezzo):  
        self.ISBN = isbn  
        self.titolo = titolo  
        self.autore = autore  
        self.prezzo = prezzo
```

```
@classmethod
```

```
    def crea_libro(cls):  
        ISBN = input("inserisci il codice ISBN del libro ")  
        titolo = input("inserisci il titolo del libro ")  
        autore = input("inserisci l'autore del libro ")  
        prezzo = float(input("inserisci il prezzo del libro "))  
        return cls(ISBN, titolo, autore, prezzo)
```

Creazione di Tabelle

```
from c_libro import CLibro

class CLibri:

    def __init__(self):
        self._libri = []

    def add_libro(self):
        nuovo_libro = CLibro.crea_libro()
        self._libri.append(nuovo_libro)

    def cerca_libro(self, titolo):
        for libro in self._libri:
            if libro.titolo == titolo:
                return libro

        raise ValueError("Libro non presente")
```


Creazione di Tabelle

```
class CVenditaLibro:

    def __init__(self, libro, num_copie):
        self._libro = libro
        self.num_copie = num_copie

    @property
    def ISBN(self):
        return self._libro.ISBN

    @property
    def prezzo(self):
        return self._libro.prezzo

    @classmethod
    def crea_vendita_libro(cls, libri):
        titolo = input("inserisci il titolo del libro per il quale vuoi inserire una vendita ")
        libro = libri.cerca_libro(titolo)
        num_copie = input("inserisci il numero di copie vendute del libro ")
        return cls(libro, num_copie)
```

Creazione di Tabelle

```
class CCliente:
```

```
    def __init__(self, nome, cognome, CF, num_tel):  
        self.nome = nome  
        self.cognome = cognome  
        self.CF = CF  
        self.num_tel = num_tel
```

```
@classmethod
```

```
def crea_cliente(cls):  
    nome = input("inserisci il nome del cliente ")  
    cognome = input("inserisci il cognome del cliente ")  
    CF = input("inserisci il codice fiscale del cliente ")  
    num_tel = input("inserisci il numero di telefono del cliente ")  
    nuovo_cliente = cls(nome, cognome, CF, num_tel)  
    return nuovo_cliente
```

Creazione di Tabelle

```
class CClienti:

    def __init__(self):
        self._clienti = []

    def add_cliente(self):
        nuovo_cliente = CCliente.crea_cliente()
        self._clienti.append(nuovo_cliente)

    def cerca_cliente(self, CF):
        for cliente in self._clienti:
            if cliente.CF == CF:
                return cliente

        raise ValueError("Libro non presente")
```

Creazione di Tabelle

```
from c_vendita_libro import CVenditaLibro

class CRicevuta:

    def __init__(self, idx, data, cliente, vendite_libri):
        self.idx = idx
        self.data = data
        self._cliente = cliente
        self._vendite_libri = vendite_libri

    @property
    def nome(self):
        return self._cliente.nome

    @property
    def cognome(self):
        return self._cliente.cognome
```

Creazione di Tabelle

```
@classmethod
def crea_ricevuta(cls, libri, clienti):
    idx = input("inserisci l'idx della ricevuta ")
    data = input("inserisci la data della ricevuta ")
    CF = input("inserisci il codice fiscale del cliente per il quale vuoi inserire
una vendita ")
    cliente = clienti.cerca_cliente(CF)
    num_dif_libri_venduti = int(input("inserisci il numero di libri differenti
venduti "))
    vendite_libri = []
    for idx_tipologia_libro_venduto in range(num_dif_libri_venduti):
        vendite_libri.append(CVenditaLibro.crea_vendita_libro(libri))
    return cls(idx, data, cliente, vendite_libri)
```

Creazione di Tabelle

```
from c_ricevuta import CRicevuta

class CRicevute:

    def __init__(self):
        self._ricevute = []

    def add_ricevuta(self, libri, clienti):
        nuova_ricevuta = CRicevuta.crea_ricevuta(libri, clienti)
        self._ricevute.append(nuova_ricevuta)
```

Creazione di Tabelle

```
from c_clienti import CClienti
from c_libri import CLibri
from c_ricevute import CRicevute

clienti = CClienti()
clienti.add_cliente()
libri = CLibri()
libri.add_libro()
libri.add_libro()

ricevute = CRicevute()
ricevute.add_ricevuta(libri, clienti)
```

Creazione di Tabelle

NB: bisogna evitare di duplicare i dati che sono già memorizzati in altre tabelle

Tabelline che mi servono nel db con le varie colonne

libri

isbn	titolo	autore	prezzo

clienti

CF	nome	cognome	numero telefono

vendite libri

idx_ricevuta	isbn	quantità

ricevute

idx	CF	data

Creazione di Tabelle

Ci riferiamo alle altre tabelle usando le chiavi esterne (in blu)

Tabelline che mi servono nel db con le varie colonne

libri

isbn	titolo	autore	prezzo

clienti

CF	nome	cognome	numero telefono

vendite libri

idx_ricevuta	isbn	quantità

ricevute

idx	CF	data

Creazione di Tabelle

```
CREATE TABLE libri
```

```
(
```

```
    ISBN CHARACTER(14) PRIMARY KEY,  
    titolo VARCHAR(40) NOT NULL UNIQUE,  
    autore VARCHAR(20) NOT NULL,  
    prezzo NUMERIC(3,2) NOT NULL
```

```
);
```

Creazione di Tabelle

```
CREATE TABLE clienti  
(  
    CF CHARACTER(16) PRIMARY KEY,  
    nome VARCHAR(20) NOT NULL,  
    cognome VARCHAR(20) NOT NULL,  
    n_telefono VARCHAR(15) NOT NULL  
);
```

Creazione di Tabelle

```
CREATE TABLE ricevute
```

```
(
```


```
    IDX CHARACTER(8) PRIMARY KEY,
```

```
    CF CHARACTER(16) NOT NULL REFERENCES clienti(CF)
```

```
    ON DELETE CASCADE ON UPDATE CASCADE,
```

```
    data DATE NOT NULL
```

```
);
```



In caso di cancellazione o aggiornamento della tabella riferita le righe coinvolte in questa tabella vengono a loro volta cancellate/aggiornate

Creazione di Tabelle

```
CREATE TABLE vendite_libri  
(  
    IDX_ricevuta CHARACTER(8) NOT NULL REFERENCES  
ricevute(idx) ON DELETE CASCADE ON UPDATE CASCADE,  
    ISBN CHARACTER(14) NOT NULL REFERENCES libri(ISBN) ON  
DELETE CASCADE ON UPDATE CASCADE,  
    quantita INTEGER NOT NULL  
);
```

Esercizio: Creazione di Tabelle

Supponiamo di voler creare un programma che permette di memorizzare alcune informazioni relative al calcio.

In particolare deve permettere di memorizzare:

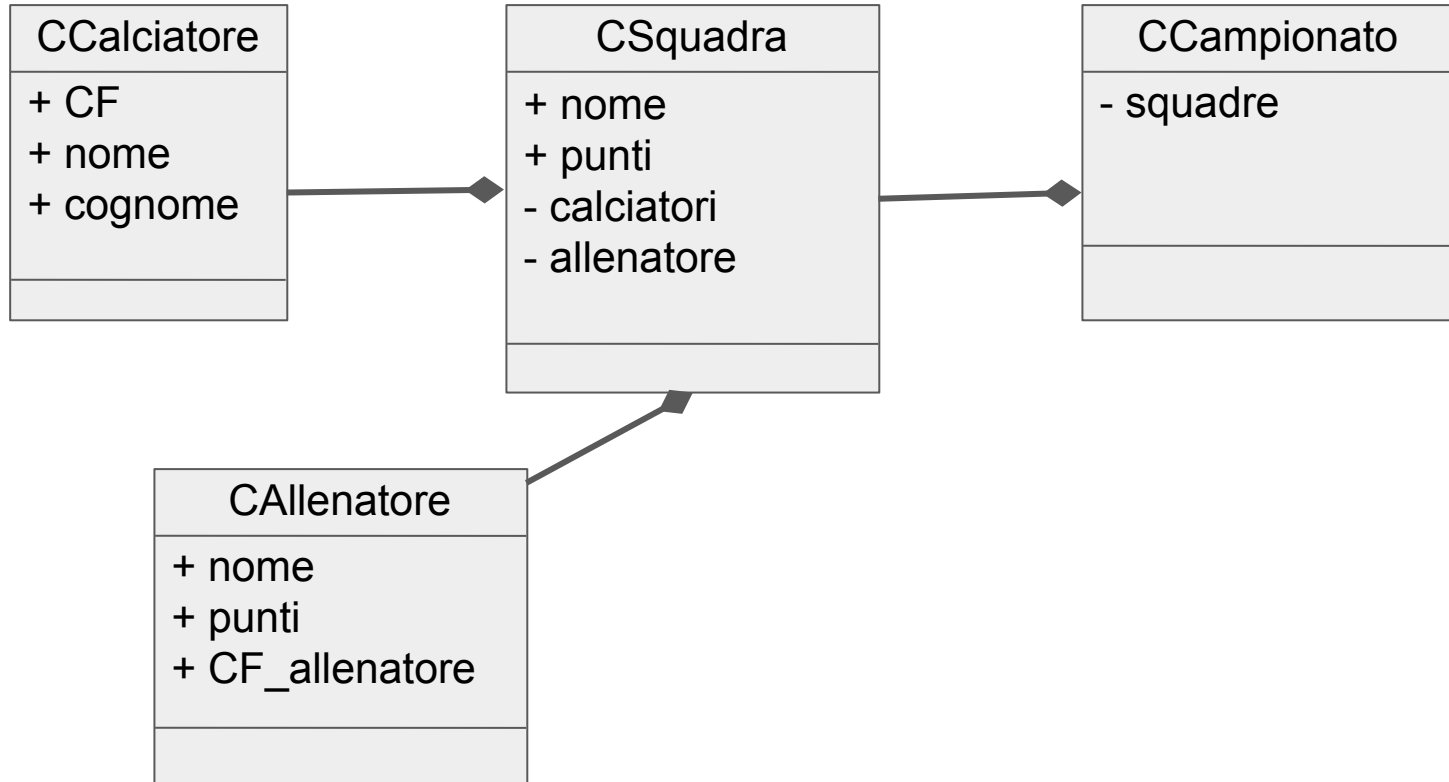
- Per ogni calciatore il suo codice fiscale, nome, cognome e la squadra alla quale appartiene;
- Per ogni squadra il nome, i punti ottenuti nel campionato e il codice fiscale dell'allenatore;
- Per ogni allenatore il codice fiscale, il suo nome, il suo cognome

Esercizio: Creazione di Tabelle

Parte 1:

Creare diagrammi di classe che useremmo se dovessimo implementarlo con classi e oggetti.

Esercizio: Creazione di Tabelle



Esercizio: Creazione di Tabelle

Parte 2:

Creare le tabelle che ci servono nel caso in cui si voglia realizzare il programma utilizzando i database.

Ricordatevi di creare un nuovo database per questo esercizio.

Ricordatevi che il dataset va creato come utente root.

Esercizio: Creazione di Tabelle

calciatori

CF	nome	cognome	squadra

squadre

nome	punti	CF_allenatore

allenatori

CF	nome	cognome

Esercizio: Creazione di Tabelle

calciatori

CF	nome	cognome	squadra

squadre

nome	punti	CF_allenatore

allenatori

CF	nome	cognome

Ci riferiamo alle altre
tabelle usando le chiavi
esterne (in blu)

Esercizio: Creazione di Tabelle

```
CREATE TABLE allenatori  
(  
    CF CHARACTER(16) PRIMARY KEY,  
    nome VARCHAR(20) NOT NULL,  
    cognome VARCHAR(20) NOT NULL  
);
```

Esercizio: Creazione di Tabelle

```
CREATE TABLE squadre  
(  
    nome VARCHAR(20) PRIMARY KEY,  
    punti INTEGER NOT NULL,  
    CFAllenatore CHARACTER(16) NOT NULL REFERENCES  
    allenatori(CF) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

Esercizio: Creazione di Tabelle

```
CREATE TABLE calciatori  
(  
    CF CHARACTER(16) PRIMARY KEY,  
    nome VARCHAR(20) NOT NULL,  
    cognome VARCHAR(20) NOT NULL,  
    squadra VARCHAR(20) NOT NULL REFERENCES squadre(nome)  
    ON DELETE CASCADE ON UPDATE CASCADE  
);
```

Selezione di Tuple da Tabelle

Per estrarre delle informazioni da delle tabelle possiamo utilizzare la SELECT.

```
SELECT <tabella.attr1>, <tabella.attr2>, ...  
FROM tabella  
[WHERE <condizione>]
```

Alcuni operatori che possiamo utilizzare per esprimere la condizione:

<, >, <=, >=, = (uguale a), <> (diverso da)

OR AND

Selezione di Tuple da Tabelle

Se volessimo ad esempio ottenere titolo e prezzo del libro con ISBN '9374287969' ci basterebbe scrivere:

```
SELECT libri.titolo, libri.prezzo  
FROM libri  
WHERE libri.ISBN = '9374287969';
```

Stamperebbe:

```
+-----+-----+  
| titolo                | prezzo |  
+-----+-----+  
| Python 2 Object-Oriented Programming | 34.99 |  
+-----+-----+  
1 row in set (0,001 sec)
```


Selezione di Tuple da Tabelle

Possiamo specificare se, in caso di righe di output duplicate vogliamo vengano visualizzate tutte (ALL) o no (DISTINCT):

```
SELECT [ALL | DISTINCT] <attr1>, <attr2>, ... (all è il default)
```

```
FROM <tabella>
```

```
WHERE <condizione>
```

Selezione di Tuple da Tabelle

```
SELECT ALL libri.anno_pubblicazione  
FROM libri  
WHERE libri.prezzo > 30;
```

+-----+	
anno_pubblicazione	
+-----+	
	2018
	2018
+-----+	

Selezione di Tuple da Tabelle

```
SELECT DISTINCT libri.anno_pubblicazione  
FROM libri  
WHERE libri.prezzo > 30;
```

```
+-----+  
| anno_pubblicazione |  
+-----+  
|          2018      |  
+-----+
```

Selezione di Tuple da Tabelle

Può capitare che i dati che vogliamo andare a recuperare coinvolgano più tabelle.

Possiamo ottenere il prodotto cartesiano delle righe nella tabella 1 e nella tabella 2 scrivendo:

```
SELECT <attr1>, <attr2>, ...  
FROM <tabella1>, <tabella2>  
WHERE <condizione>;
```

Selezione di Tuple da Tabelle

Supponiamo di avere le seguenti tabelle:

clienti

CF	nome	cognome	numero telefono
STFBNC78R31 E432T	Stefania	Bianchi	3334499558
MRCRSS66R3 3E542T	Marcello	Rossi	3883367453

ricevute

idx	CF	data
123	STFBNC78R3 1E432T	20/05/25
456	MRCRSS66R 33E542T	25/05/25

Selezione di Tuple da Tabelle

Il prodotto cartesiano sarebbe:

CF	nome	cognome	numero telefono	idx	CF	data
STFBNC78R31E432T	Stefania	Bianchi	3334499558	123	STFBNC78R31E432T	20/05/25
MRCRSS66R33E542T	Marcello	Rossi	3883367453	123	STFBNC78R31E432T	20/05/25
STFBNC78R31E432T	Stefania	Bianchi	3334499558	456	MRCRSS66R33E542T	25/05/25
MRCRSS66R33E542T	Marcello	Rossi	3883367453	456	MRCRSS66R33E542T	25/05/25

Selezione di Tuple da Tabelle

Se volessimo andare a recuperare l'idx delle ricevute emesse per il cliente con cognome Bianchi potremmo scrivere:

```
SELECT ricevute.idx  
FROM clienti, ricevute  
WHERE clienti.CF = ricevute.CF AND clienti.cognome='Bianchi';
```

```
+-----+
```

```
| idx |
```

```
+-----+
```

```
| 123 |
```

```
+-----+
```

```
1 row in set (0,001 sec)
```

Selezione di Tuple da Tabelle

Per mettere insieme le informazioni ottenute da diverse tabelle si può utilizzare anche l'inner join (scelta migliore).

```
SELECT <attr1>, <attr2>  
FROM <tab1> JOIN <tab2>  
ON <tab1.attr> = <tab2.attr>
```


Selezione di Tuple da Tabelle

Ad esempio, la query di prima diventerebbe

```
SELECT ricevute.idx  
FROM clienti JOIN ricevute  
ON clienti.CF = ricevute.CF  
WHERE clienti.cognome = 'Bianchi';
```

```
+-----+  
| idx |  
+-----+  
| 123 |  
+-----+  
1 row in set (0,001 sec)
```

Selezione di Tuple da Tabelle

Scriviamo la query per andare a recuperare dal database gli isbn di tutti i libri venduti al cliente con cognome Bianchi.

```
SELECT vendite_libri.isbn
FROM vendite_libri JOIN (SELECT ricevute.idx
                        FROM ricevute JOIN clienti
                        ON ricevute.CF = clienti.CF
                        WHERE clienti.cognome = 'Bianchi')
AS ricevute_bianchi
ON vendite_libri.idx_ricevuta = ricevute_bianchi.idx;
```

Diamo un nome alla
tabella risultante
dalla query

Selezione di Tuple da Tabelle

Se i dati nella tabella “vendite_libri” fossero i seguenti:

idx_ricevuta	isbn	quantità
123	9374287969	4
123	1484236939	4
456	9374287969	1

Selezione di Tuple da Tabelle

Il risultato sarebbe:

```
+-----+  
| isbn  |  
+-----+  
| 9374287969 |  
| 1484236939 |  
+-----+  
2 rows in set (0,001 sec)
```

Esercizio: Creazione di Tabelle

calciatori

CF	nome	cognome	squadra
ALS	Alessandro	Del Piero	Juventus
RCC	Riccardo	Kaka	Milan
GGB	Gigi	Buffon	Juventus

squadre

nome	punti	CF_allenatore
Juventus	15	FBC
Milan	13	RBT

allenatori

CF	nome	cognome
FBC	Fabio	Capello
RBT	Roberto	Mancini

Ci riferiamo alle altre
tabelle usando le chiavi
esterne (in blu)

Esercizio: Selezione di Tuple da Tabelle

Considerando le tabelle scritte per risolvere l'esercizio precedente, scrivere la query per trovare i cognomi di tutti gli allenatori di una squadra che ha totalizzato almeno 15 punti nel campionato.

Esercizio: Selezione di Tuple da Tabelle

```
SELECT allenatori.cognome  
FROM allenatori join squadre  
ON squadre.CFAllenatore = allenatori.CF  
WHERE squadre.punti >= 15;
```

```
+-----+  
| cognome |  
+-----+  
| Capello |  
+-----+
```

Esercizio: Selezione di Tuple da Tabelle

Considerando le tabelle scritte per risolvere l'esercizio precedente, scrivere la query per trovare i cognomi di tutti i giocatori che hanno come allenatore "Allegri".

Esercizio: Selezione di Tuple da Tabelle

```
SELECT calciatori.cognome
FROM calciatori join(SELECT squadre.nome
                      FROM squadre join allenatori
                      ON squadre.CF_allenatore = allenatori.CF
                      WHERE allenatori.cognome = 'Capello')
                      as squadra_capello
ON calciatori.squadra = squadra_capello.nome;
```

```
+-----+
| cognome |
+-----+
| Del Piero |
| Buffon   |
+-----+
```

Eliminare Righe

```
DELETE FROM <tabella>  
WHERE <condizione>
```

Es:

```
DELETE FROM allenatori  
WHERE CF = 'FBC';
```

(Elimina la prima riga della tabella allenatori)

Gli Oggetti e i Database

Poichè sfruttando i database i dati vengono memorizzati nelle tabelle non ci serve più creare delle classi?

No.

Come vedremo ci servono comunque delle classi per gestire il database.