

# 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 MySQLConnector
- Connettersi a MySQL
- Effettuare query ad un database
- Gestire gli input degli utenti
- Le transazioni

# Testo di Riferimento

MySQL Connector/Python Revealed

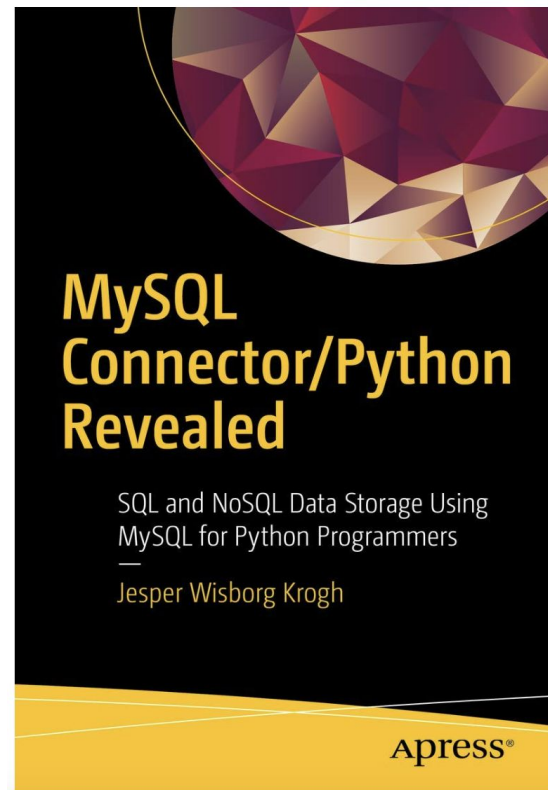
**Autore:** Jesper Wisborg Krogh

**Editore:** Apress

**Anno pubblicazione:** 2018

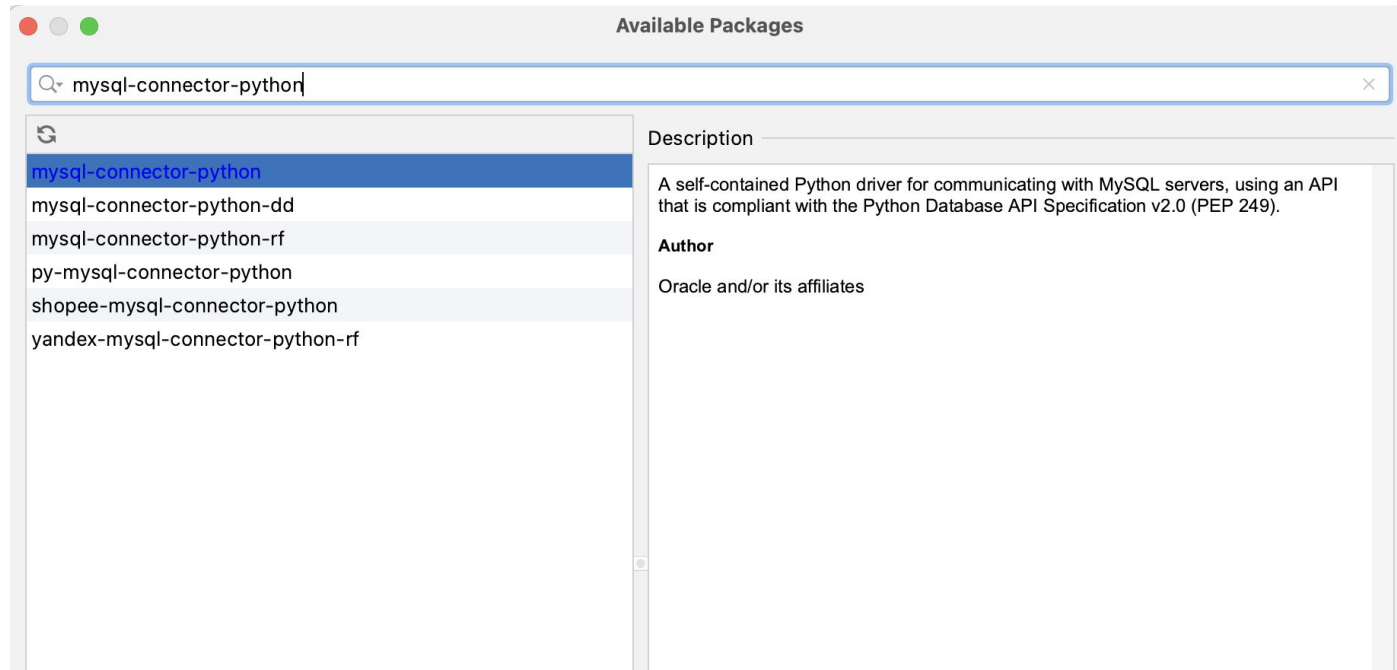
Da pagina 1 a pagina 132.

Da pagina 175 a pagina 190.



# Installare MySQLConnector

Si può installare tramite pycharm.



# Connettersi a MySql

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(host="127.0.0.1",
                    port=3306,
                    user="pyuser",
                    password="Py@pp4Demo")

print("MySQL connection ID for mysqlconn :
{}".format(mysqlcon_obj.connection_id))

mysqlcon_obj.close()
```

La connessione va  
sempre chiusa  
quando non è  
essenziale stia aperta.

# Connettersi a MySQL

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(host="127.0.0.1",
                    port=3306,
                    user="pyuser",
                    password="Py@pp4Demo")

print("MySQL connection ID for mysqlconn :
{}".format(mysqlcon_obj.connection_id))

mysqlcon_obj.close()
```

Questo codice funziona ma per ragioni di sicurezza l'utente e soprattutto la password non andrebbero mai scritte nel codice!

(Non è detto che tutti gli sviluppatori debbano avere accesso alle credenziali, inoltre il codice andrebbe condiviso con metodi sicuri).

# Connettersi a MySql

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(host="127.0.0.1",
                    port=3306,
                    user="pyuser",
                    password="Py@pp4Demo")

print("MySQL connection ID for mysqlconn :
{}".format(mysqlcon_obj.connection_id))

mysqlcon_obj.close()
```

Anche riceverle come argomenti da linea di comando non è una buona soluzione perchè potrebbero essere recuperate da altri utenti es con il comando 'ps'.

# Connettersi a MySQL

Nel caso in cui si possa interagire con l'utente.

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect( host="127.0.0.1",
                      port=3306,
                      user=input("inserisci l'utente "),
                      password=input("inserisci la password "))

print("MySQL connection ID for mysqlconn :
{}".format(mysqlcon_obj.connection_id))

mysqlcon_obj.close()
```



# Connettersi a MySql

Quando il processo viene avviato in automatico è necessario memorizzarle. Possiamo memorizzarle utilizzando un file di configurazione di MySQL.

```
[connector_python]
user = pyuser
host = 127.0.0.1
port = 3306
password = Py@pp4Demo
```

Di solito questo file viene chiamato:

- my.ini su windows
- my.cnf negli altri sistemi operativi

Viene di solito posizionato nella stessa directory del progetto

# Connettersi a MySql

Creiamo la connessione utilizzando il file di configurazione di MySQL.

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

print("MySQL connection ID for mysqlconn :  
{0}".format(mysqlcon_obj.connection_id))

mysqlcon_obj.close()
```

# Eseguire delle Query ad un Database

```
import mysql.connector as mysqlcon
import pprint

printer = pprint.PrettyPrinter(indent = 1)

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

results = mysqlcon_obj.cmd_query(
    """SELECT *
FROM DB_libreria.libri
""")

result_set = mysqlcon_obj.get_rows()
printer.pprint(result_set)


mysqlcon_obj.close()
```

# Eseguire delle Query ad un Database

Il risultato sarà:

```
((['1484236939',  
  'MySQL Connector',  
  'Jesper Wisborg Krogh',  
  'Apress',  
  Decimal('64.19'),  
  2018),  
 ('9374287969',  
  'Python 2 Object-Oriented Programming',  
  'Dusty Phillips',  
  'Packt',  
  Decimal('34.99'),  
  2018)),  
 {'status_flag': 33, 'warning_count': 0})
```

Il primo elemento della tupla restituita è una lista che contiene una tupla per ogni riga presente nella tabella.



# Eseguire delle Query ad un Database

Se ad esempio volessimo mostrarne n (es 1) per volta:

```
import mysql.connector as mysqlcon
import pprint

printer = pprint.PrettyPrinter(indent = 1)

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

results = mysqlcon_obj.cmd_query(
    """SELECT *
FROM DB_libreria.libri
""")
```

```
results = mysqlcon_obj.get_rows(1)
printer.pprint(results)
```

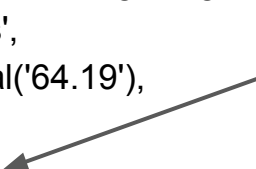
```
results = mysqlcon_obj.get_rows(1)
printer.pprint(results)
```

```
mysqlcon_obj.close()
```

# Eseguire delle Query ad un Database

Il risultato sarebbe:

```
((('1484236939',  
  'MySQL Connector',  
  'Jesper Wisborg Krogh',  
  'Apress',  
  Decimal('64.19'),  
  2018)),  
None)
```




Poichè c'è una riga da leggere (il libro con isbn '1484236939'), il secondo elemento della tupla è uguale a None.

```
((('9374287969',  
  'Python 2 Object-Oriented Programming',  
  'Dusty Phillips',  
  'Packt',  
  Decimal('34.99'),  
  2018)),  
None)
```

# Eseguire delle Query ad un Database

Se invocassimo ancora la `get_rows` non essendoci più righe l'output sarebbe:

```
([], {'status_flag': 33, 'warning_count': 0})
```



Il fatto che ci venga restituito un oggetto come secondo elemento della tupla ci segnala il fatto che non ci sono più righe da leggere.

# Eseguire delle Query ad un Database

Quando si esegue una query, tutti i risultati devono essere consumati prima di poterne eseguire un'altra.

```
import mysql.connector as mysqlcon
import pprint

printer = pprint.PrettyPrinter(indent = 1)

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

results = mysqlcon_obj.cmd_query(
    """SELECT *
FROM DB_libreria.libri
""")
```

```
results = mysqlcon_obj.get_rows(1)
printer.pprint(results)
```

```
results = mysqlcon_obj.get_rows(1)
printer.pprint(results)
```

```
mysqlcon_obj.close()
```



# Eseguire delle Query ad un Database

Tutti i risultati devono essere consumati prima di poter eseguire delle altre query.

```
import mysql.connector as mysqlcon
import pprint

printer = pprint.PrettyPrinter(indent = 1)

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

query="""SELECT *
FROM DB_libreria.libri
"""

results = mysqlcon_obj.cmd_query(
query)
```

```
results = mysqlcon_obj.get_rows(1)
printer.pprint(results)

#eseguo una query prima di leggere la
seconda riga

results = mysqlcon_obj.cmd_query(
query)

mysqlcon_obj.close()
```

**mysql.connector.errors.InternalError:**  
**Unread result found**

# Eseguire delle Query ad un Database

Generalmente si utilizzano dei cursori per ottenere i risultati delle query. Questi permettono anche di ottenere le righe come dizionari.

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

cursore = mysqlcon_obj.cursor(dictionary=True)

query="""SELECT * FROM DB_libreria.libri
"""
cursore.execute(query)
riga = cursore.fetchall()
print(riga)
cursore.close()
mysqlcon_obj.close()
```

# Eseguire delle Query ad un Database

```
[  
{'ISBN': '1484236939', 'titolo': 'MySQL Connector', 'autore': 'Jesper Wisborg Krogh',  
'editore': 'Apress', 'prezzo': Decimal('64.19'), 'anno_pubblicazione': 2018},  
  
{'ISBN': '9374287969', 'titolo': 'Python 2 Object-Oriented Programming', 'autore':  
'Dusty Phillips', 'editore': 'Packt', 'prezzo': Decimal('34.99'), 'anno_pubblicazione':  
2018}  
]
```

# Eseguire delle Query ad un Database

Per leggerli uno alla volta

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

cursore = mysqlcon_obj.cursor(dictionary=True)

query="""SELECT * FROM DB_libreria.libri
"""
cursore.execute(query)
riga = cursore.fetchone()
while riga:
    print(riga)
    riga = cursore.fetchone()
cursore.close()
mysqlcon_obj.close()
```

# Eseguire delle Query ad un Database

```
{'ISBN': '1484236939', 'titolo': 'MySQL Connector', 'autore': 'Jesper Wisborg Krogh', 'editore': 'Apress',  
'prezzo': Decimal('64.19'), 'anno_pubblicazione': 2018}  
{'ISBN': '9374287969', 'titolo': 'Python 2 Object-Oriented Programming', 'autore': 'Dusty Phillips',  
'editore': 'Packt', 'prezzo': Decimal('34.99'), 'anno_pubblicazione': 2018}
```

# Gestire gli Input degli Utenti

Spesso le query vengono generate sulla base di input degli utenti o di altre sorgenti esterne.

**Attenzione:** Mai utilizzare le informazioni in input senza assicurarsi che verranno gestite in modo che non possano cambiare il significato della query!

# Gestire gli Input degli Utenti

Strategie per gestire i dati in input:

- Validarli (controllare che siano del tipo e che abbiano i valori attesi)
- Parametrizzare le query

# Gestire gli Input degli Utenti

Parametrizzare le query permette di non doversi preoccupare di inserire i caratteri di escape nei dati.

Supponete di voler creare una query fatta così:

```
SELECT *  
FROM DB_libreria.libri
```

```
WHERE Name = ?
```

Dove il ? rappresenta un nome fornito in input dall'utente



# Gestire gli Input degli Utenti

L'utente potrebbe inserire un input come quello mostrato per farsi restituire tutte le colonne.

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

cursore = mysqlcon_obj.cursor(dictionary=True)

input = "1 OR True"
#input = "9374287969"
query="""SELECT *
FROM DB_libreria.libri WHERE ISBN = {}
""".format(input)
params = {"isbn":input}
```

```
cursore.execute(query)

print(cursore.statement)

riga = cursore.fetchall()
print(riga)

cursore.close()
mysqlcon_obj.close()
```

# Gestire gli Input degli Utenti

Ciò che verrebbe stampato sarebbe:

```
SELECT *  
FROM DB_libreria.libri WHERE ISBN = 1 OR True
```

```
[{'ISBN': '1484236939', 'titolo': 'MySQL Connector', 'autore': 'Jesper  
Wisborg Krogh', 'editore': 'Apress', 'prezzo': Decimal('64.19'),  
'anno_publicazione': 2018},  
{'ISBN': '9374287969', 'titolo': 'Python 2 Object-Oriented Programming',  
'autore': 'Dusty Phillips', 'editore': 'Packt', 'prezzo': Decimal('34.99'),  
'anno_publicazione': 2018}]
```

# Gestire gli Input degli Utenti

Parametrizzando la query il cursore saprebbe che l'input dovrebbe essere una stringa.

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

cursore = mysqlcon_obj.cursor(dictionary=True)

input = "1 OR True"
#input = "9374287969"
query="""SELECT *
FROM DB_libreria.libri WHERE ISBN = %(isbn)s
"""
```

```
params = {"isbn":input}

cursore.execute(query,
params=params)

print(cursore.statement)

cursore.close()
mysqlcon_obj.close()
```

# Gestire gli Input degli Utenti

Ciò che verrebbe stampato sarebbe:

```
SELECT *  
FROM DB_libreria.libri WHERE ISBN = '1 OR True'  
[]
```

# Gestire gli Input degli Utenti

Se dobbiamo ripetere la stessa query per più valori conviene settare il parametro “prepared” della funzione cursor uguale a True

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

cursore = mysqlcon_obj.cursor(dictionary=True,
prepared=True)

inputs = ("9374287969", "1484236939")
query="""SELECT *
FROM DB_libreria.libri WHERE ISBN = %(isbn)s
"""
```

```
for input in inputs:
    params = {"isbn": input}
    cursore.execute(query,
params=params)

    riga = cursore.fetchall()
    print(riga)

cursore.close()
mysqlcon_obj.close()
```

# Gestire gli Input degli Utenti

```
[{'ISBN': '9374287969', 'titolo': 'Python 2 Object-Oriented Programming',  
'autore': 'Dusty Phillips', 'editore': 'Packt', 'prezzo': Decimal('34.99'),  
'anno_pubblicazione': 2018}]
```

```
[{'ISBN': '1484236939', 'titolo': 'MySQL Connector', 'autore': 'Jesper  
Wisborg Krogh', 'editore': 'Apress', 'prezzo': Decimal('64.19'),  
'anno_pubblicazione': 2018}]
```

# Gestire gli Input degli Utenti

Utilizzando `prepared = True` lo statement viene preparato prima.

Per le esecuzioni verranno inviati tramite la rete solo i parametri da sostituire.

E' più efficiente nel caso in cui la stessa query vada eseguita con diversi input.

# Inserimento Righe

Per inserire una nuova riga prima di chiudere la connessione bisogna effettuare un commit.

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

cursore = mysqlcon_obj.cursor(dictionary=True, prepared=True)

query="""INSERT INTO DB_libreria.libri VALUES ('3711227654', 'Artificial Intelligence',
'Stuart Russel', 'Global Edition', 84.99, 2022);
"""
cursore.execute(query, params=None)

mysqlcon_obj.commit()
cursore.close()
mysqlcon_obj.close()
```



# Default Database

Per evitare di dover ripetere il nome del database in tutte le query possiamo settarlo come proprietà della connessione.

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect(option_files="my.cnf")

mysqlcon_obj.database = "DB_libreria"
```

# Transazioni

Raggruppano insieme diverse query per assicurare che vengano eseguite tutte oppure nessuna (quelle già eseguite vengono annullate se quelle dopo non vanno a buon fine).

I cambiamenti effettuati da una transazione non saranno visibili fino a che tutte non sono state eseguite.

# Transazioni

**ACID** (**A**tomicità, **C**onsistenza, **I**solamento, **D**urabilità) definisce il comportamento che il database deve avere per essere considerato abbastanza affidabile per processare transazioni.

**Atomicità.** *Per le transazioni che contengono più comandi deve effettuarli tutti o nessuno* (se uno fallisce quelli precedenti vengono annullati).

ES: un bonifico fatto tra due clienti di una stessa banca deve diminuire il conto di un cliente e aumentare quello di un'altro.

**Consistenza.** *Solo i dati validi devono essere memorizzati nel database.* Se violano delle regole devono essere scartati e il database deve avere lo stato in cui era prima della transazione.

# Transazioni

**Isolamento.** *Transazioni multiple che vengono eseguite nello stesso momento non interferiscono l'una con l'altra.* Per far sì che questo accada molti sistemi usano un meccanismo chiamato *locking* che evita che i dati vengano usati da un'altra transazione fino a che la prima non è terminata.

**Durabilità.** *Nessuna transazione porterà alla perdita di dati esistenti nel database o di quelli creati/alterati durante la transazione.*

# Transazioni

Supponiamo di voler aggiungere una riga nella tabella “vendite\_libri” e una nella tabella “ricevute” e di voler annullare la transazione se una delle operazioni non va a buon fine.

NB: la vendita e la ricevuta potremo aggiungerle solo se i dati relativi ai libri e al cliente sono già presenti. Inseriamo la vendita in modo che sia valida mentre facciamo sì che la ricevuta non sia valida.

L’inserimento della vendita dovrà quindi essere annullato.

# Transazioni

```
import mysql.connector as mysqlcon

mysqlcon_obj = mysqlcon.MySQLConnection()
mysqlcon_obj.connect( option_files="my.cnf")
mysqlcon_obj.database = "DB_libreria"

cursore = mysqlcon_obj.cursor( dictionary=True, prepared=True)

try:
    mysqlcon_obj.start_transaction()

    query_inserimento_vendita= """
INSERT INTO vendite_libri VALUES
( %(idx_ricevuta)s, %(isbn)s, %(quantita)s );
"""
    params = { "idx_ricevuta": "888",
               "isbn": "1484236939",
               "quantita": 1}

    results = cursore.execute(query_inserimento_vendita, params=params)

    query_inserimento_ricevuta= """
INSERT INTO ricevute VALUES ('888', 'ZZZZZZZZZZZZZZZZ', '28-05-25');
"""
    results = cursore.execute(query_inserimento_ricevuta)
    mysqlcon_obj.commit()
```

```
except:
    mysqlcon_obj.rollback()
    print("transazioni annullate")

query_inserimento_cliente= """
INSERT INTO clienti VALUES
('ELNRSS78R31E432T', 'Eleonora', 'Rossi',
'3330011222');
"""
cursore.execute(query_inserimento_cliente)

mysqlcon_obj.commit()

cursore.close()
mysqlcon_obj.close()
```

Andando a controllare le tabelle vedreste che il primo inserimento non è presente (è stato annullato).

# L'utilizzo dei Database

Utilizziamo i database nell'esercizio che chiedeva di creare un programma per un paesino turistico.

# L'utilizzo dei Database

incassi\_annui

anno	nome_albergo	incasso

sagre

anno	costo_cibo

concerti

anno	n_musicisti	ore_durata



# L'utilizzo dei Database

```
CREATE TABLE incassi_annui  
(  
    anno INTEGER NOT NULL,  
    nome_albergo VARCHAR(40) NOT NULL,  
    incasso NUMERIC(9,2) NOT NULL  
);
```

# L'utilizzo dei Database

```
CREATE TABLE sagre  
(  
    anno INTEGER NOT NULL,  
    costo_cibo NUMERIC(7,2) NOT NULL  
);
```

# L'utilizzo dei Database

```
CREATE TABLE concerti  
(  
    anno INTEGER NOT NULL,  
    n_musicisti INTEGER NOT NULL,  
    ore_durata INTEGER NOT NULL  
);
```

# L'utilizzo dei Database

Modificate il codice sostituendo la lettura dei dati relativi agli incassi dal database invece che dal file.

Modificate il codice sostituendo la serializzazione della lista degli eventi e la sua lettura con scrittura e lettura di dati nel database.

NB: questo possiamo farlo supponendo che i dati non siano tantissimi, visto che sono dati relativi a degli anni e di un singolo comune. Altrimenti converrebbe evitare di tenere in memoria la lista di oggetti ed effettuare per ogni occorrenza letture dal database.

# L'utilizzo dei Database

Create il diagramma delle classi.