



**FRANCESCO
SPALLUZZI**

SQLAlchemy e Python

Profilo



Francesco Spalluzzi

SESSION OWNER

Developer.net in aesys tech srl

DotNet Developer con esperienza pluriennale nello sviluppo di soluzioni software utilizzando tecnologie Microsoft, con focus su .NET Framework, .NET Core, e strumenti moderni come Blazor e Azure. Forte capacità di lavorare in team medio-grandi e competenze consolidate nella progettazione, sviluppo e ottimizzazione di applicazioni web e servizi backend e Training in .NET E Python per Jobformazione.it

✉ francescospalluzzi@gmail.com

AGENDA

- ❖ Ecosistema Database e DBMS; Perché utilizzare SQLAlchemy con SQL Server Setup Ambiente
- ❖ Codice Python per gestire una connessione ad una base dati in SQL Server e comprensione del codice per l'uso pratico della libreria SQLAlchemy
- ❖ CRUD – CREATE READ UPDATE DELETE con del codice Python da commentare e considerazioni sull'esecuzione delle query semplici e avanzate e calling di stored-procedure con l'uso della libreria SQLAlchemy
- ❖ Q&A

Ecosistema DATABASE E DBMS in Python – Libreria SQLAlchemy – Setup Ambiente (5 minuti)

Introduzione all'Ecosistema DATABASE e DBMS

Database: Struttura organizzata per immagazzinare dati. Esistono database relazionali (SQL) e non relazionali (NoSQL).

DBMS: Sistema di gestione del database che permette di interagire con i dati. Esempi di DBMS relazionali includono SQL Server, MySQL, PostgreSQL.

Python è una scelta popolare per lavorare con i database grazie alle sue librerie, tra cui spicca **SQLAlchemy** per la gestione relazionale.

Libreria SQLAlchemy

- **Cos'è SQLAlchemy?**
 - È una libreria Python che funge da ORM (*Object Relational Mapper*) e motore SQL.
 - Permette di interagire con i database usando codice Python senza scrivere query SQL direttamente.
 - Supporta diversi DBMS, tra cui SQL Server, grazie ai driver.
- **Caratteristiche principali:**
 - ORM: Mappa classi Python alle tabelle del database.
 - Core SQL: Consente di scrivere query SQL dettagliate.
 - Portabilità: Cambiare database è semplice.
- **Vantaggi:**
 - Codice leggibile e manutenibile.
 - Riduzione degli errori legati all'uso diretto di SQL.

Ecosistema DATABASE E DBMS in Python – Libreria SQLAlchemy – Setup Ambiente (5 minuti)

3. Setup Ambiente con SQL Server (utilizzo del modulo venv di Python)

```
C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio>dir
Il volume nell'unità C è Windows
Numero di serie del volume: 9CB4-DB92

Directory di C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio

0/12/2024 10:54 <DIR>      .
1/12/2024 14:56 <DIR>      ..
0/12/2024 10:17          1.466 abstract CALL FOR PAPER_Python_SQLALCHEMY.txt
6/12/2024 11:04 <DIR>      labSQLAlchemy
                1 File          1.466 byte
                3 Directory 360.085.778.432 byte disponibili

C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio>python -m venv demoSQLAlchemy
```

Prerequisiti

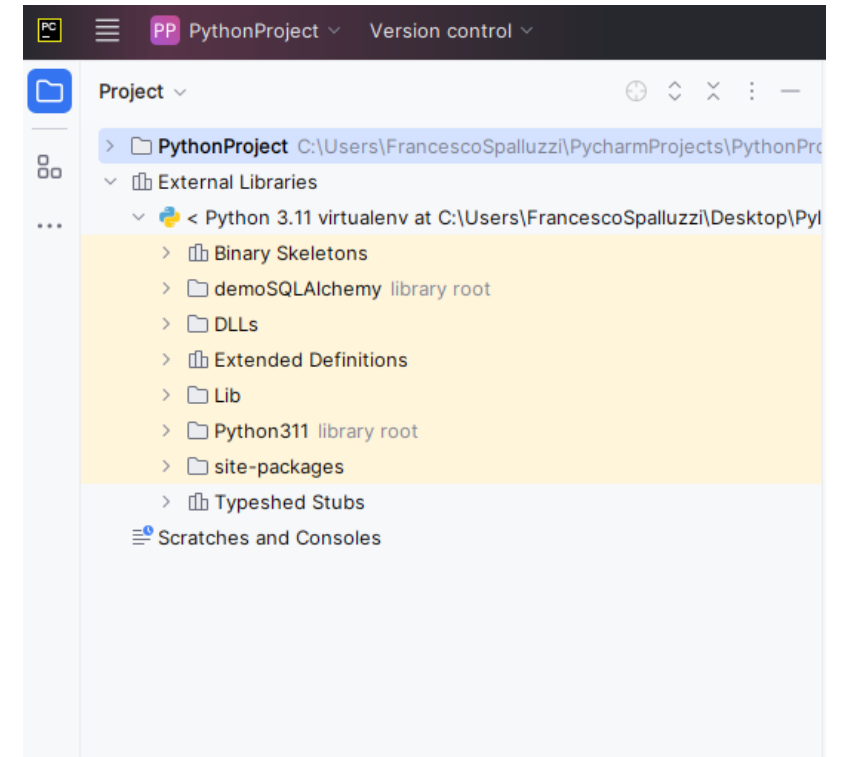
- SQL Server installato e configurato.
- Driver ODBC o **pyodbc** installato per la connessione.
- `pip install sqlalchemy` `pip install pyodbc`

```
Administratore: Prompt dei comandi

(demoSQLAlchemy) C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scripts>pip install SQLAlchemy
Collecting SQLAlchemy
  Obtaining dependency information for SQLAlchemy from https://files.pythonhosted.org/packages/b1/03/d12b7c1d36fd0150c1d52e121614cf9377dac99e5497af8d8f5b2a8db64/SQLAlchemy-2.0.36-cp311-cp311-win_amd64.whl.metadata
  Using cached SQLAlchemy-2.0.36-cp311-cp311-win_amd64.whl.metadata (9.9 kB)
Collecting typing-extensions>=4.6.0 (from SQLAlchemy)
  Obtaining dependency information for typing-extensions>=4.6.0 from https://files.pythonhosted.org/packages/26/9f/ad03fc0248c5379346306f866cdabe2e2ec95e01216d2b8ff09ff037d0/typing_extensions-4.12.2-py3-none-any.whl.metadata
  Using cached typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Collecting greenlet<0.4.17 (from SQLAlchemy)
  Obtaining dependency information for greenlet<0.4.17 from https://files.pythonhosted.org/packages/12/da/b9ed5e310bb8b09661b0c6cd4d5a6b7903bbcd7fc854923f5ebb4144f0/greenlet-3.1.1-cp311-cp311-win_amd64.whl.metadata
  Using cached greenlet-3.1.1-cp311-cp311-win_amd64.whl.metadata (3.9 kB)
Using cached SQLAlchemy-2.0.36-cp311-cp311-win_amd64.whl (2.1 MB)
Using cached greenlet-3.1.1-cp311-cp311-win_amd64.whl (298 kB)
Using cached typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: typing-extensions, greenlet, SQLAlchemy
Successfully installed SQLAlchemy-2.0.36 greenlet-3.1.1 typing-extensions-4.12.2

[notice] A new release of pip is available: 23.2.1 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

(demoSQLAlchemy) C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scripts>
```



```
(demoSQLAlchemy) C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scripts>pip install pyodbc
Collecting pyodbc
  Obtaining dependency information for pyodbc from https://files.pythonhosted.org/packages/7c/6b/f8ad7d8a53d58f35f375ffbf367c68d0ec54452a431d23b0ebee4cd44c6/pyodbc-5.2.0-cp311-cp311-win_amd64.whl.metadata
  Using cached pyodbc-5.2.0-cp311-cp311-win_amd64.whl.metadata (2.8 kB)
Using cached pyodbc-5.2.0-cp311-cp311-win_amd64.whl (68 kB)
Installing collected packages: pyodbc
Successfully installed pyodbc-5.2.0

[notice] A new release of pip is available: 23.2.1 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

(demoSQLAlchemy) C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scripts>
```

Ecosistema DATABASE E DBMS in Python – Libreria SQLAlchemy – Setup Ambiente (5 minuti)

Configurazione DSN di Microsoft SQL Server

☒ Usa il seguente database predefinito:
DemoSQLAlchemyPython

Server mirror:
SPN per server mirror (facoltativo):

☐ Associa file di database:

☒ Usa identificatori delimitati ANSI
☒ Usa avvisi, riempimenti e caratteri Null ANSI

Finalità dell'applicazione:
READWRITE

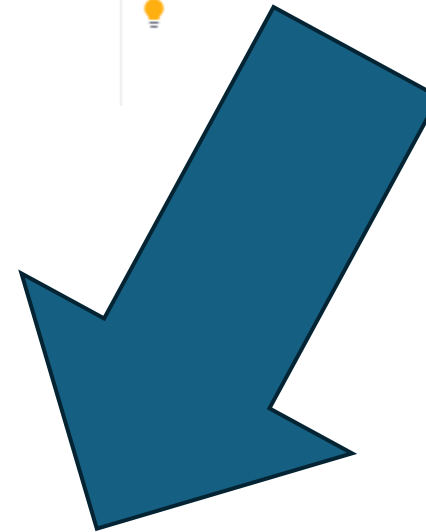
☐ Failover su più subnet.

< Indietro **Avanti >** Annulla ?

testInstallSqlAlchemy.py ×

sqlalchemy is supporte... Try PyCharm Professional Dismiss

```
1 import sqlalchemy ✓
2 print(sqlalchemy.__version__)
```



Run testInstallSqlAlchemy ×

```
C:\Users\FrancescoSpalluzzi\Desktop\PyLaboratorio\demoSQLAlchemy\Scripts\python.exe C:\Users\FrancescoSpalluzzi\PycharmProjects\PythonProject\testInstallSqlAlchemy.py
2.0.36

Process finished with exit code 0
```

Ecosistema DATABASE E DBMS in Python – Libreria SQLAlchemy – Setup Ambiente (5 minuti)

FILE JSON DI CONFIGURAZIONE

```
1 {
2     "sql_server": {
3         "username": "sa",
4         "password": "dbsvil",
5         "server": "(localdb)\\MSSQLLocalDB",
6         "database": "DemoSQLAlchemyPython",
7         "driver": "DemoSqlAlchemy"
8     }
9 }
```

Configurazione della Connessione

```
CreateTableAndCrud.py x connessioneSQL.json testInstallSqlAlchemy.py SQLAlchemyCore_Example.py
sqlalchemy is supported by PyCharm Professional

1 from sqlalchemy import create_engine
2 import pyodbc
3 import json
4 with open("connessioneSQL.json", "r") as file:
5     config = json.load(file)
6
7 # Estrarre i dati dal file JSON
8 db_config = config["sql_server"]
9 username = db_config["username"]
10 password = db_config["password"]
11 server = db_config["server"]
12 database = db_config["database"]
13 driver = db_config["driver"]
14
15 # Creare la stringa di connessione
16 connection_string_1 = f"mssql+pyodbc://{username}:{password}@{driver}"
17
18
19 #Test connessione con SQL Alchemy library
20
21 engine = create_engine(connection_string_1)
22 try:
23     with engine.connect() as connection:
24         print(f"Connessione riuscita! per {connection_string_1}")
25 except Exception as e:
26     print(f"Errore di connessione: {e}")
27
28
```

```
Run TestConnessioneSQLSrv x
C:\Users\FrancescoSpalluzzi\Desktop\PyLaboratorio\demoSQLAlchemy\Scripts\python.exe C:\Users\FrancescoSpalluzzi\PycharmProjects\PythonProject\TestConnessioneSQLSrv.py
Connessione riuscita! per mssql+pyodbc://sa:dbsvil@DemoSqlAlchemy
Process finished with exit code 0
```

Ecosistema DATABASE E DBMS in Python – Libreria SQLAlchemy – Setup Ambiente (5 minuti)

Utilizzando SQLAlchemy Core: Accesso diretto a una tabella – Una prima query di esempio

```
from sqlalchemy import create_engine, MetaData, Table, text
import json
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
```

```
def connessione(): 1 usage
    engine=None
    with open("connessioneSQL.json", "r") as file:
        config = json.load(file)
```

```
# Estrarre i dati dal file JSON
db_config = config["sql_server"]
username = db_config["username"]
password = db_config["password"]
server = db_config["server"]
database = db_config["database"]
driver = db_config["driver"]
```

```
# Creare la stringa di connessione
connection_string_1 = f"mssql+pyodbc://{username}:{password}@{driver}"
```

```
# Creazione engine
```

```
engine = create_engine(connection_string_1)
```

```
return engine, connection_string_1
```

```
def EseguiQuery(connessioneEngineSQL, stringaconnessione, sqlQuery): 1 usage
    try:
        with connessioneEngineSQL.connect() as connection:
            print(f"Connessione riuscita! per {stringaconnessione}")
            result = connection.execute(text(sqlQuery))
            for row in result:
                print(row)
            connection.close()
    except Exception as e:
        print(f"Errore: {e}")
```

```
#configurazione engine leggendo la stringa di connessione da un file JSON
engine, stringaconnessione=connessione();
```

```
#ESECUZIONE QUERY DI ESEMPIO
```

```
sql_query="Select * from [dbo].[listaPersone]"
EseguiQuery(engine, stringaconnessione, sql_query)
```


Ecosistema DATABASE E DBMS in Python – Libreria SQLAlchemy – Setup Ambiente (5 minuti)

Output a run-time di questo snippet code

```
Run SQLAlchemyCore_Example x
C:\Users\FrancescoSpalluzzi\Desktop\PyLaboratorio\demoSQLAlchemy\Scripts\python.exe C:\Users\FrancescoSpalluzzi\PycharmProjects\PythonProject\SQLAlchemyCore_Example.py
Connessione riuscita! per mssql+pyodbc://sa:dbsvil@DemoSQLAlchemy
['id', 'name']
(1, 'Banco frigorifero')
(2, 'HP I7 16 GB')
(3, 'HP I7 32 GB')
Process finished with exit code 0
```

(5 minuti) Codice Python per gestire una connessione ad una base dati in SQL Server e comprensione del codice per l'uso pratico della libreria SQLAlchemy

```
from sqlalchemy import create_engine, MetaData, Table, text
import json
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

def connessione(): 1 usage
    engine=None
    with open("connessioneSQL.json", "r") as file:
        config = json.load(file)

    # Estrarre i dati dal file JSON
    db_config = config["sql_server"]
    username = db_config["username"]
    password = db_config["password"]
    server = db_config["server"]
    database = db_config["database"]
    driver = db_config["driver"]

    # Creare la stringa di connessione
    connection_string_1 = f"mssql+pyodbc://{username}:{password}@{driver}"

    # Creazione engine

    engine = create_engine(connection_string_1)

    return engine, connection_string_1
```

engine=None

engine, stringaconnessione=connessione()

EsecuzioneQueryWithCreatingModel(engine, stringaconnessione)

```
def EsecuzioneQueryWithCreatingModel(engine, stringaconnessione): 1 usage
    try:
        with engine.connect() as connection:
            print(f"Connessione riuscita! per {stringaconnessione}")
            # Creare un oggetto MetaData
            metadata = MetaData()

            # Riflettere una tabella specifica dal database
            table_name = "Descrizioni" # Sostituisci con il nome della tua tabella
            my_table = Table(table_name, metadata, autoload_with=engine)
            # Stampare informazioni sulla tabella
            print(my_table.columns.keys()) # Elenco delle colonne della tabella

            # Creare una sessione
            Session = sessionmaker(bind=engine)
            session = Session()

            # Esempio di query: ottenere tutti i record
            records = session.query(my_table).all()
            for record in records:
                print(record)

    except Exception as e:
        print(f"Errore: {e}")
```

(5 minuti) Codice Python per gestire una connessione ad una base dati in SQL Server e comprensione del codice per l'uso pratico della libreria SQLAlchemy

```
C:\Users\FrancescoSpalluzzi\Desktop\PyLaboratorio\demoSQLAlchemy\Scripts\python.exe C:\Users\FrancescoSpalluzzi\PycharmProjects\PythonProject\SQLAlchemyCore_Example.py
Connessione riuscita! per mssql+pyodbc://sa:dbsvil@DemoSQLAlchemy
['id', 'name']
(1, 'Banco frigorifero')
(2, 'HP I7 16 GB')
(3, 'HP I7 32 GB')
Process finished with exit code 0
```

(5 minuti) Codice Python per gestire una connessione ad una base dati in SQL Server e comprensione del codice per l'uso pratico della libreria SQLAlchemy

```
from sqlalchemy import create_engine, MetaData, Table, text
import json
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

def connessione(): 1 usage
    engine=None
    with open("connessioneSQL.json", "r") as file:
        config = json.load(file)

    # Estrarre i dati dal file JSON
    db_config = config["sql_server"]
    username = db_config["username"]
    password = db_config["password"]
    server = db_config["server"]
    database = db_config["database"]
    driver = db_config["driver"]

    # Creare la stringa di connessione
    connection_string_1 = f"mssql+pyodbc://{username}:{password}@{driver}"

    # Creazione engine
    engine = create_engine(connection_string_1)

    return engine,connection_string_1
```

```
engine=None
engine,stringaconnessione=connessione();
```

```
class Descrizioni(): 1 usage
    def __init__(self,nometabella,engine):
        self.tableName=nometabella
        self.__tablename__='Descrizioni'
        self.id = Column( __name_pos: 'id',Integer, primary_key=True)
        self.name = Column( __name_pos: 'name',String)
        meta=MetaData()
        Table(self.tableName,meta, *args: self.id,self.name)
        meta.create_all(engine)

def CreateTable(nometabella,engine): 1 usage
    Descrizioni(nometabella,engine)

def InsertIntoTable(nometabella,engine): 1 usage
    metadata=MetaData()
    _table_=Table(nometabella, metadata, autoload_with=engine)
    return _table_
```

(5 minuti) Codice Python per gestire una connessione ad una base dati in SQL Server e comprensione del codice per l'uso pratico della libreria SQLAlchemy

```
# Base per la definizione delle classi ORM
CreateTable( nome_tabella: "Descrizioni", engine)

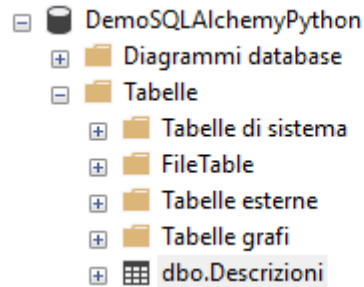
tabellaDescrizioni=InsertIntoTable( nome_tabella: "Descrizioni", engine)

# crea sql statement insert
sqlInsert=Insert(tabellaDescrizioni).values(name='Banco frigorifero')

sqlInsert1=Insert(tabellaDescrizioni).values(name='HP I7 16 GB')

sqlInsert2=Insert(tabellaDescrizioni).values(name='HP I7 32 GB')

try:
    with engine.connect() as connection:
        print(f"Connessione riuscita! per {stringaconnessione}")
        connection.execute(sqlInsert)
        connection.execute(sqlInsert1)
        connection.execute(sqlInsert2)
        connection.commit()
        connection.close()
except Exception as e:
    print(f"Errore: {e}")
```



SQLQuery7.sql - (L...emyPython (sa (64)) - X X ~vs8CD5.sql - non connesso Esplora oggetti

```
1 SELECT TOP (1000) [id]
2     , [name]
3 FROM [DemoSQLAlchemyPython].[dbo].[Descrizioni]
4
```

90 %

	id	name
1	1	Banco frigorifero
2	2	HP I7 16 GB
3	3	HP I7 32 GB

Risultati Messaggi

(10 minuti) CRUD – CREATE READ UPDATE DELETE con del codice Python da commentare e considerazioni sull'esecuzione delle query semplici e avanzate e calling di stored-procedures con l'uso della libreria SQLAlchemy

```
from sqlalchemy import create_engine, MetaData, Table, text, Column, Integer, String, Insert
import json
from sqlalchemy.orm import DeclarativeBase, Session

class Base(DeclarativeBase):
    metadata = MetaData()

def connessione():
    engine=None
    with open("connessioneSQL.json", "r") as file:
        config = json.load(file)

    # Estrarre i dati dal file JSON
    db_config = config["sql_server"]
    username = db_config["username"]
    password = db_config["password"]
    server = db_config["server"]
    database = db_config["database"]
    driver = db_config["driver"]

    # Creare la stringa di connessione
    connection_string_1 = f"mssql+pyodbc://{username}:{password}@{driver}"

    # Creazione engine

    engine = create_engine(connection_string_1)

    return engine, connection_string_1
```

(10 minuti) CRUD – CREATE READ UPDATE DELETE con del codice Python da commentare e considerazioni sull'esecuzione delle query semplici e avanzate e calling di stored-procedure con l'uso della libreria SQLAlchemy

```
class User(Base): 2 usages
    __tablename__ = 'users'
    id = Column( __name_pos: 'id', Integer, primary_key=True)
    name = Column( __name_pos: 'name', String(50))
    email = Column( __name_pos: 'email', String(100))

def CreateTable(): 1 usage
    engine, stringaconnessione = connessione()
    Base.metadata.create_all(engine)

def insert_data(data): 1 usage
    engine, stringaconnessione = connessione()
```

```
def insert_data(data): 1 usage
    engine, stringaconnessione = connessione()

    with Session(engine) as session:
        try:
            if isinstance(data, list):
                users = [User(**record) for record in data]
                session.add_all(users)
            elif isinstance(data, dict):
                user = User(**data)
                session.add(user)
            else:
                raise ValueError("I dati devono essere un dizionario o una lista di dizionari.")
            session.commit()
            print("Dati inseriti con successo!")
        except Exception as e:
            session.rollback()
            print(f"Errore durante l'inserimento: {e}")
```

(10 minuti) CRUD – CREATE READ UPDATE DELETE con del codice Python da commentare e considerazioni sull’esecuzione delle query semplici e avanzate e calling di stored-procedure con l’uso della libreria SQLAlchemy

```
# Creare le tabelle nel database
CreateTable()

# Esempio di inserimento dati
data_to_insert = [
    {"name": "Alice", "email": "alice@example.com"},
    {"name": "Bob", "email": "bob@example.com"}
]
insert_data(data_to_insert)
```

A screenshot of a Python IDE's 'Run' console. The title bar shows 'CreateEntityWithSqlAlchemy'. The console output displays the file path 'C:\Users\FrancescoSpalluzzi\Desktop\PyLaboratorio\demoSQLAlchemy\Scripts\python.exe C:\Users\FrancescoSpalluzzi\PycharmProjects\PythonProject\CreateEntityWithSqlAlchemy.py', followed by the message 'Dati inseriti con successo!' and 'Process finished with exit code 0'.

Run CreateEntityWithSqlAlchemy x

C:\Users\FrancescoSpalluzzi\Desktop\PyLaboratorio\demoSQLAlchemy\Scripts\python.exe C:\Users\FrancescoSpalluzzi\PycharmProjects\PythonProject\CreateEntityWithSqlAlchemy.py

Dati inseriti con successo!

Process finished with exit code 0

(10 minuti) **CRUD – CREATE READ UPDATE DELETE** con del codice Python da commentare e considerazioni sull’esecuzione delle query semplici e avanzate e calling di stored-procedures con l’uso della libreria **SQLAlchemy**

SQLQuery14.sql - (...emyPython (sa (55))) ~vs8CD5.sql - non connesso Esplora oggetti

```
1 SELECT TOP (1000) [id]
2     , [name]
3     , [email]
4 FROM [DemoSQLAlchemyPython].[dbo].[users]
5
```

90 %

Risultati Messaggi

	id	name	email
1	1	Alice	alice@example.com
2	2	Bob	bob@example.com

(10 minuti) CRUD – CREATE READ UPDATE DELETE con del codice Python da commentare e considerazioni sull’esecuzione delle query semplici e avanzate e calling di stored-procedures con l’uso della libreria SQLAlchemy

Creazione di un modello per rappresentare il join tra clienti e ordini

```
1 from sqlalchemy import create_engine, MetaData, Table, text, Column, Integer, String, Insert, ForeignKey, Boolean
2 import json
3 from sqlalchemy.orm import DeclarativeBase, Session, relationship
4
5 class Base(DeclarativeBase):
6     metadata = MetaData()
```

```
class Cliente(Base):
    __tablename__ = 'clienti'

    id = Column(Integer, primary_key=True, autoincrement=True)
    nome = Column(String, nullable=False)
    email = Column(String, nullable=False)

    # Relazione con Ordine
    ordini = relationship(argument="Ordine", back_populates="cliente", cascade="all, delete-orphan")
```

```
# Entità Ordine
class Ordine(Base):
    __tablename__ = 'ordini'

    id = Column(Integer, primary_key=True, autoincrement=True)
    descrizione = Column(String, nullable=False)
    evaso = Column(Boolean, default=False)
    cliente_id = Column(Integer, ForeignKey('clienti.id'), nullable=False)

    # Relazione inversa con Cliente
    cliente = relationship(argument="Cliente", back_populates="ordini")
```

Crea_struttura

n.b Si può raffinare questa procedura

```
#crea struttura tabelle
def crea_struttura(engine):
    try:
        Base.metadata.create_all(engine)
        print("Struttura del database creata con successo.")
    except Exception as e:
        print(f"Errore durante la creazione della struttura: {e}")
```

```
def inserisci_dati(session):
    # Inserimento di 5 clienti
    clienti = [
        Cliente(nome="Mario Rossi", email="mario.rossi@example.com"),
        Cliente(nome="Luigi Verdi", email="luigi.verdi@example.com"),
        Cliente(nome="Anna Bianchi", email="anna.bianchi@example.com"),
        Cliente(nome="Paola Neri", email="paola.neri@example.com"),
        Cliente(nome="Giovanni Blu", email="giovanni.blug@example.com")
    ]
    session.add_all(clienti)
    session.commit()

    # Recupero degli ID dei clienti
    clienti = session.query(Cliente).all()

    # Inserimento di 6 ordini
    ordini = [
        Ordine(descrizione="Ordine 1", evaso=False, cliente_id=clienti[0].id),
        Ordine(descrizione="Ordine 2", evaso=False, cliente_id=clienti[1].id),
        Ordine(descrizione="Ordine 3", evaso=False, cliente_id=clienti[2].id),
        Ordine(descrizione="Ordine 4", evaso=False, cliente_id=clienti[3].id),
        Ordine(descrizione="Ordine 5", evaso=False, cliente_id=clienti[4].id),
        Ordine(descrizione="Ordine 6", evaso=False, cliente_id=clienti[0].id)
    ]
    session.add_all(ordini)
    session.commit()
```

(10 minuti) CRUD – CREATE READ UPDATE DELETE con del codice Python da commentare e considerazioni sull'esecuzione delle query semplici e avanzate e calling di stored-procedure con l'uso della libreria SQLAlchemy

Visualizzazione_ordini_evasi

```
def visualizza_ordini_evasi(session):  
    from sqlalchemy.orm import joinedload  
  
    ordini_evasi = (  
        session.query(Ordine)  
        .options(joinedload(Ordine.cliente))  
        .filter(Ordine.evaso == False)  
        .all()  
    )  
    for ordine in ordini_evasi:  
        print(f"Cliente: {ordine.cliente.nome}, Email: {ordine.cliente.email}, Ordine: {ordine.descrizione}")
```

Visualizzazione_clienti_senza_ordini

```
def visualizza_clienti_senza_ordini(session):  
    # Query per selezionare i clienti che non hanno ordini  
    clienti_senza_ordini = (  
        session.query(Cliente)  
        .outerjoin(Ordine)  
        .filter(Ordine.id.is_(None))  
        .all()  
    )  
    if clienti_senza_ordini:  
        for cliente in clienti_senza_ordini:  
            print(f"Cliente: {cliente.nome}, Email: {cliente.email}")  
    else:  
        print("Non ci sono clienti senza ordini.")
```

(10 minuti) CRUD – CREATE READ UPDATE DELETE con del codice Python da commentare e considerazioni sull'esecuzione delle query semplici e avanzate e calling di stored-procedure con l'uso della libreria SQLAlchemy

Implementazione della funzione di aggiornamento ordine

```
def aggiorna_ordine(session, ordine_id, nuova_descrizione=None, nuovo_stato=None): 1 usage
# ordine = session.query(Ordine).get(ordine_id) #deprecated
# attuale per la versione 2.x di SQLAlchemy
ordine = session.get(Ordine, ordine_id)
if ordine:
    if nuova_descrizione:
        ordine.descrizione = nuova_descrizione
    if nuovo_stato is not None:
        ordine.evaseo = nuovo_stato
    session.commit()
    print(f"Ordine {ordine_id} aggiornato con successo.")
else:
    print(f"Ordine {ordine_id} non trovato.")
```

Main da eseguire

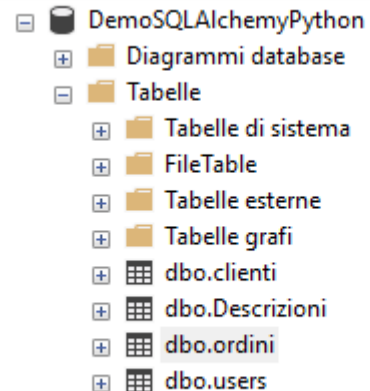
```
engine=None
engine, stringaconnessione=connessione()
#crea_struttura(engine)
Session=Session(engine)
#inserisci_dati(Session)

#visualizza_ordini_evasei(Session)

#visualizza_clienti_senza_ordini(Session)

aggiorna_ordine(Session, ordine_id: 6, nuova_descrizione: 'Ordine 6 Update', nuovo_stato: True)
```

La creiamo lato server con SSMS



```
1 SET ANSI_NULLS ON
2 GO
3 SET QUOTED_IDENTIFIER ON
4 GO
5 CREATE PROCEDURE sp_GetOrdineById(
6     -- Add the parameters for the stored procedure here
7     @idOrdine int)
8 AS
9 BEGIN
10     -- SET NOCOUNT ON added to prevent extra result sets from
11     -- interfering with SELECT statements.
12     SET NOCOUNT ON;
13
14     -- Insert statements for procedure here
15     SELECT * from Clienti c
16     join Ordini o
17     on c.Id=o.Cliente_id
18     where o.Id=@idOrdine
19 END
20 GO
21
```

(10 minuti) CRUD – CREATE READ UPDATE DELETE con del codice Python da commentare e considerazioni sull'esecuzione delle query semplici e avanzate e calling di stored-procedure con l'uso della libreria SQLAlchemy

dbo.ordini
Colonne
id (PK, int, Non Null)
descrizione (varchar(max), Non Null)
evaso (bit, Null)
cliente_id (FK, int, Non Null)
Chiavi
Vincoli
Trigger
Indici
Statistiche
dbo.users
Viste
Risorse esterne
Sinonimi
Programmabilità
Stored procedure
Stored procedure di sistema
dbo.sp_GetOrdineById

```
def esegui_spOnServer(Session,idOrdine): 1 usage
# Esecuzione della stored procedure
isql=f"EXEC sp_GetOrdineById {idOrdine}"
result = Session.execute(
    text(isql))

# Mappare i risultati
mapped_results = [
    ProcedureResult(row.Evaso, row.nome) for row in result.fetchall()
]
for item in mapped_results:
    print(item.Evaso, item.nome)
```

```
engine=None
engine,stringaconnessione=connessione()
#crea_struttura(engine)
Session=Session(engine)
#inserisci_dati(Session)

#visualizza_ordini_evasi(Session)

#visualizza_clienti_senza_ordini(Session)

#aggiorna_ordine(Session, 6, 'Ordine 6 Update',True)

esegui_spOnServer(Session, idOrdine: 1)
```

```
SQLQuery31.sql - (...emyPython (sa (59))
SQLQuery29.sql - (...emyPython (sa (53))
~vs8CD5.sql - non connesso
Esplora oggetti

1 USE [DemoSQLAlchemyPython]
2 GO
3
4 DECLARE @return_value int
5
6 EXEC @return_value = [dbo].[sp_GetOrdineById]
7 @idOrdine = 1
8
9 SELECT 'Return Value' = @return_value
10
11 GO
12
```

nome	Evaso
Mario Rossi	0

```
class ProcedureResult: 1 usage
def __init__(self, Evaso, nome):
    self.nome=nome
    self.Evaso=Evaso

def esegui_spOnServer(Session,idOrdine): 1 usage
# Esecuzione della stored procedure
isql=f"EXEC sp_GetOrdineById {idOrdine}"
result = Session.execute(
    text(isql))

# Mappare i risultati
mapped_results = [
    ProcedureResult(row.Evaso, row.nome) for row in result.fetchall()
]
for item in mapped_results:
    print(item.Evaso, item.nome)
```

```
C:\Users\FrancescoSpalluzzi\Desktop\PyLaboratorio\demoSQLAlchemy\Scripts\python.exe C:\Users\FrancescoSpalluzzi\PycharmProjects\PythonProject\demo_completa.py
False Mario Rossi

Process finished with exit code 0
```

GRAZIE 😊

Q&A

