

**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

10/12/2024  10:54    <DIR>      .
11/12/2024  14:56    <DIR>      ..
10/12/2024  10:17              1.466 abstract CALL FOR PAPER_Python_SQLALCHEMY.txt
16/12/2024  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
```

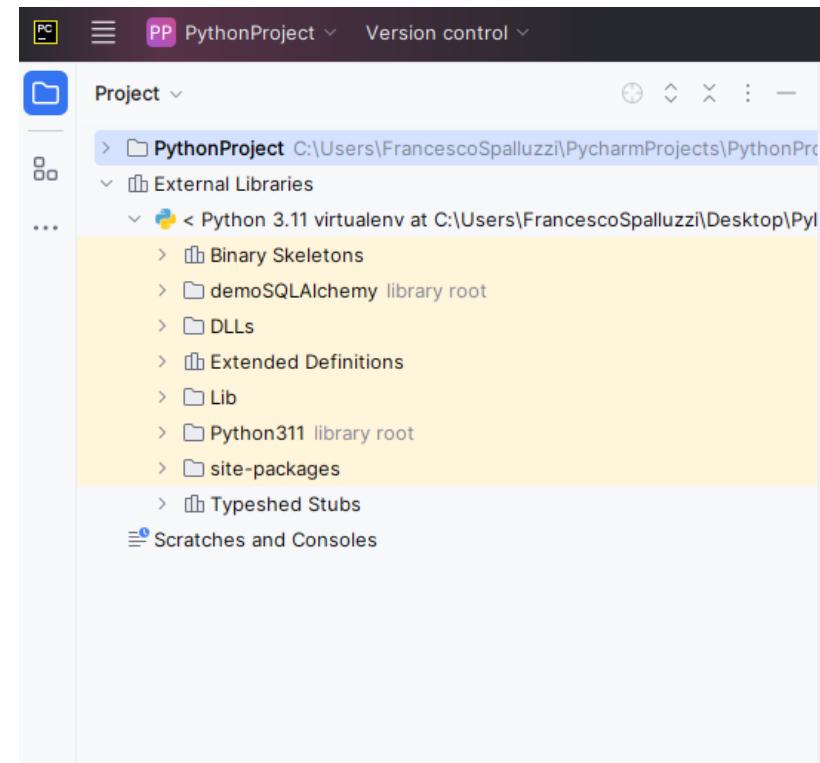
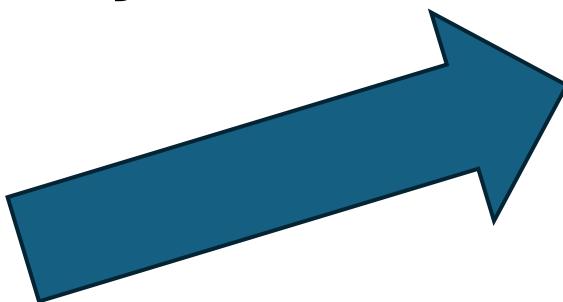
python -m venv demoSQLAlchemy

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

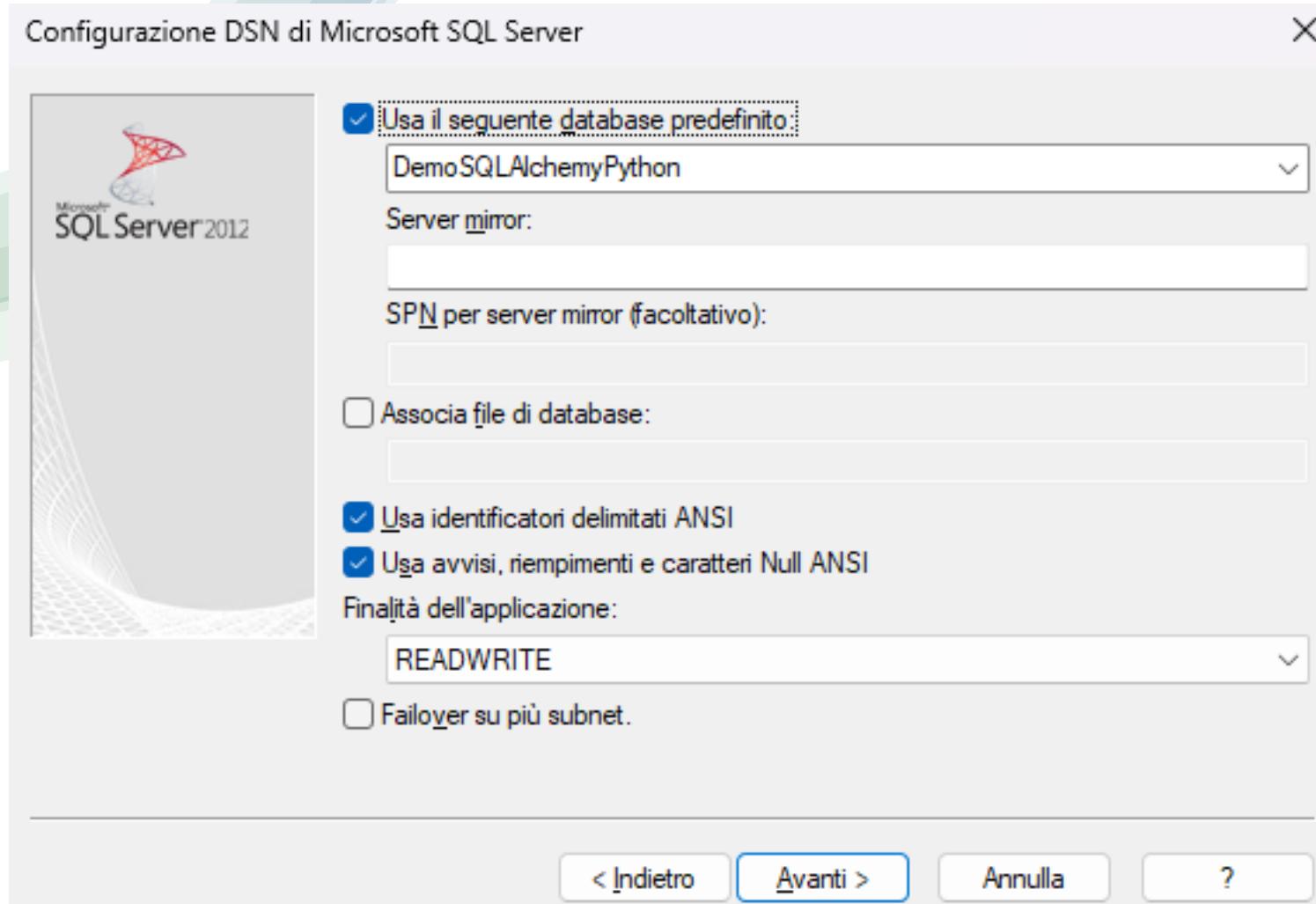
Prerequisiti (dal prompt di DOS lanciando prima il comando Activate)

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

Setup environment con Pycharm



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



Configurazione ODBC Driver per SQL Server

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

test\installSqlAlchemy.py

sqlalchemy is supported Try PyCharm Professional Dismiss

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



Run TestConnessioneSQLSrv

...

```
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)

File di configurazione json della connessione

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

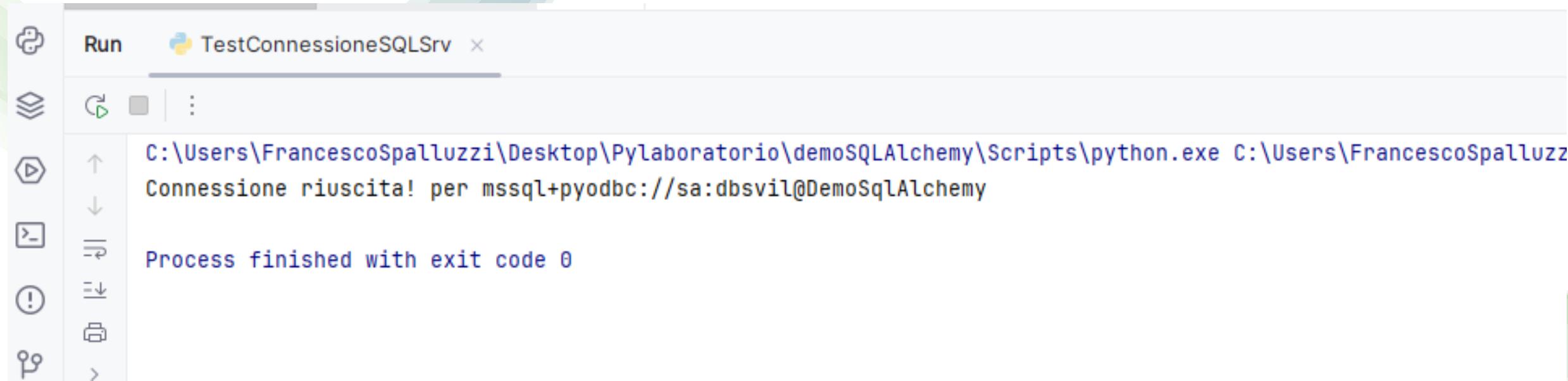
Ecosistema DATABASE E DBMS in Python – Libreria SQLAlchemy – Setup Ambiente (5 minuti)

```
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
```

**CODICE PER
TESTARE LA
CONNESSIONE A
SQL SERVER**

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



The screenshot shows the PyCharm IDE interface with a terminal window. The terminal title is "Run" and the tab name is "TestConnessioneSQLSrv". The terminal output displays the following text:

```
C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scripts\python.exe C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scripts\test_conn.py  
Connessione riuscita! per mssql+pyodbc://sa:dbsvil@DemoSqlAlchemy  
Process finished with exit code 0
```

The terminal window has a vertical toolbar on the left with icons for Run, Stop, Refresh, and other navigation functions.

OUTPUT NEL TERMINALE DI PYCHARM

Ecosistema DATABASE E DBMS in Python – Libreria SQLAlchemy –

Setup Ambiente (5 minuti)

sqlalchemy is supported by PyCharm Professional

```
1  from sqlalchemy import create_engine, MetaData, Table, text
2  import json
3  from sqlalchemy.ext.declarative import declarative_base
4  from sqlalchemy.orm import sessionmaker
5
```

IMPORTAZIONE LIBRERIE NECESSARIE

Ecosistema DATABASE E DBMS in Python – Libreria SQLAlchemy –

Setup Ambiente (5 minuti)

```
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
```

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

```
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}")
```

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

INVOCAZIONE CODICE «MAIN»

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

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

OUTPUT ESECUZIONE QUERY NEL TERMINALE

```
C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scripts\python.
Conessione 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
```

Codice Python per gestire una connessione ad una base dati in SQL Server e comprendere del codice per l'uso pratico della libreria SQLAlchemy

```
from sqlalchemy import create_engine, MetaData, Table, text
import json
```

Importazione librerie per codice sorgente QueryExample.py

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

```
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}")
```

Codice per implementare l'esecuzione della query

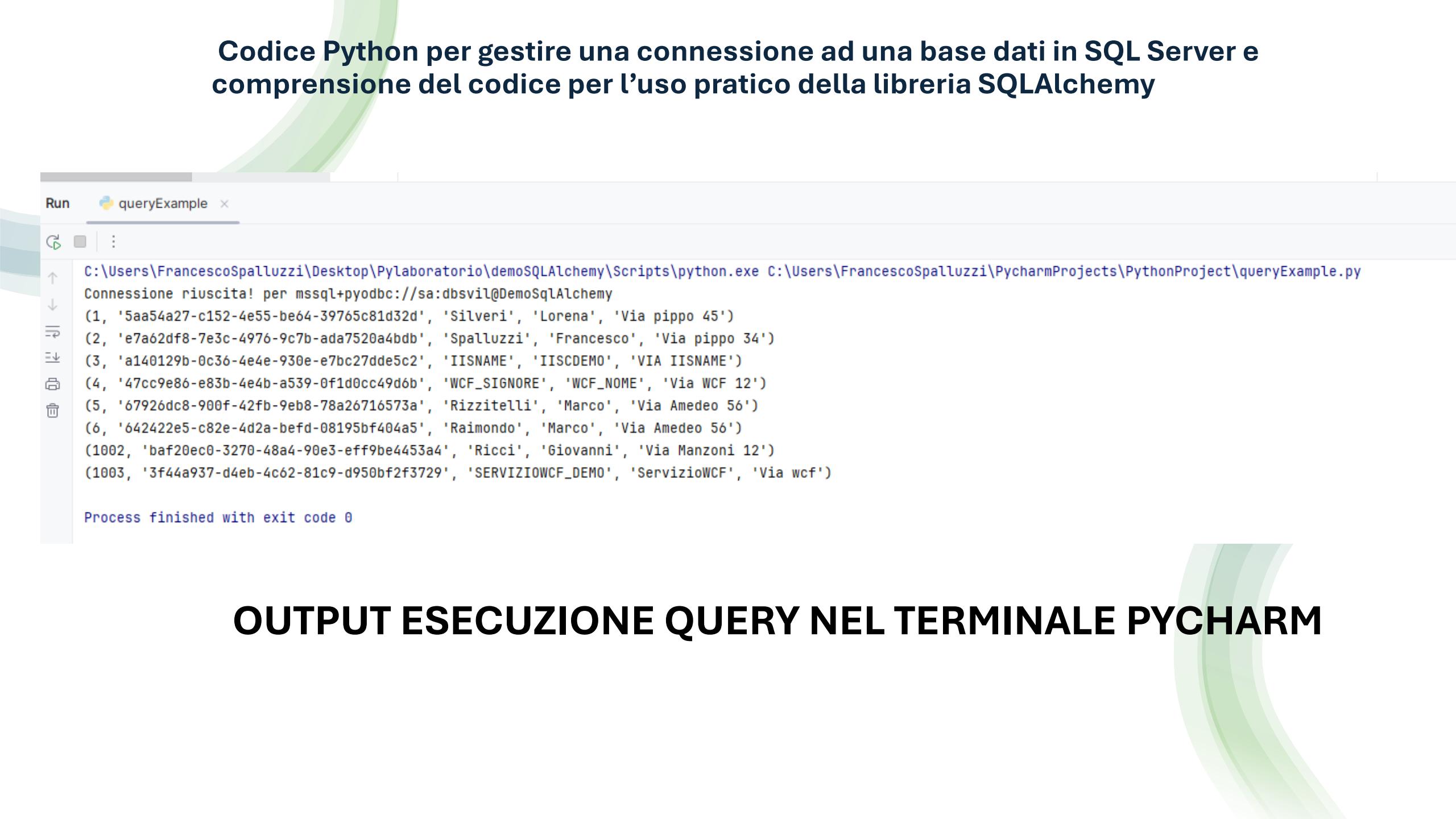
Codice Python per gestire una connessione ad una base dati in SQL Server e comprendere del codice per l'uso pratico della libreria SQLAlchemy

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

#ESECUZIONE QUERY DI ESEMPIO
sql_query="Select * from [dbo].[listaPersone]"
EseguìQuery(engine, stringaconnessione, sql_query)
```

CODICE «MAIN DA INVOCARE»

Codice Python per gestire una connessione ad una base dati in SQL Server e comprendere del codice per l'uso pratico della libreria SQLAlchemy



```
Run  queryExample x
C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scripts\python.exe C:\Users\FrancescoSpalluzzi\PycharmProjects\PythonProject\queryExample.py
Connessione riuscita! per mssql+pyodbc://sa:dbsvil@DemoSqlAlchemy
(1, '5aa54a27-c152-4e55-be64-39765c81d32d', 'Silveri', 'Lorena', 'Via pippo 45')
(2, 'e7a62df8-7e3c-4976-9c7b-ada7520a4bdb', 'Spalluzzi', 'Francesco', 'Via pippo 34')
(3, 'a140129b-0c36-4e4e-930e-e7bc27dde5c2', 'IISNAME', 'IISCDemo', 'VIA IISNAME')
(4, '47cc9e86-e83b-4e4b-a539-0f1d0cc49d6b', 'WCF_SIGNORE', 'WCF_NOME', 'Via WCF 12')
(5, '67926dc8-900f-42fb-9eb8-78a26716573a', 'Rizzitelli', 'Marco', 'Via Amedeo 56')
(6, '642422e5-c82e-4d2a-befd-08195bf404a5', 'Raimondo', 'Marco', 'Via Amedeo 56')
(1002, 'baf20ec0-3270-48a4-90e3-eff9be4453a4', 'Ricci', 'Giovanni', 'Via Manzoni 12')
(1003, '3f44a937-d4eb-4c62-81c9-d950bf2f3729', 'SERVIZIOWCF_DEMO', 'ServizioWCF', 'Via wcf')

Process finished with exit code 0
```

OUTPUT ESECUZIONE QUERY NEL TERMINALE PYCHARM

Codice Python per gestire una connessione ad una base dati in SQL Server e comprendere del codice per l'uso pratico della libreria SQLAlchemy

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

LIBRERIE NECESSARIE PER L'ESECUZIONE DEL FILE CreateTable.py

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

```
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
```

CONNESIONE AL DATABASE CON IL FILE DI CONFIGURAZIONE

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

```
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)
```

Definizione del modello Descrizione come Classe in Python

Codice Python per gestire una connessione ad una base dati in SQL Server e comprendere del codice per l'uso pratico della libreria SQLAlchemy

```
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_
```

Due metodi che eseguono rispettivamente:

- ❖ La creazione dell'oggetto come tabella nel db di SQL Server
- ❖ Funzione che ritorna lo schema della tabella creata

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

```
#CREAZIONE ENGINE
engine=None

engine, stringaconnessione=connessione();

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

tabellaDescrizioni=InsertIntoTable( nometabella: "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}")
```

CODICE MAIN DA INVOCARE ALL'AVVIO DELLO SCRIPT

Codice Python per gestire una connessione ad una base dati in SQL Server e
comprendere del codice per l'uso pratico della libreria SQLAlchemy

OUTPUT ESECUZIONE SCRIPT SLIDES PRECEDENTE CREAZIONE TABELLA CON INSERIMENTO DATI

The screenshot shows a SQL Server Management Studio (SSMS) interface. At the top, there's a toolbar with a zoom level of 90%. Below the toolbar, there are two tabs: "Risultati" (Results) and "Messaggi" (Messages). The "Risultati" tab is selected and displays a table with three rows of data:

	<code>id</code>	<code>name</code>
1	1	Banco frigorifero
2	2	HP I7 16 GB
3	3	HP I7 32 GB

Below the results, the SQL query used to retrieve the data is visible in the query editor:

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

(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

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

LIBRERIE NECESSARIE PER L'ESECUZIONE DELLO SCRIPT CreateEntityWithSqlAlchemy.py

(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 Base(DeclarativeBase): 2 usages
    metadata = MetaData()
```



```
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))
```

DEFINIZIONE DELLA CLASSE BASE DI SQLAlchemy e Classe del modello Users (derivata da Base)

(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

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

Procedura che verrà invocata dal main per creare l'oggetto (la tabella) sul db

(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

```
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}")
```

Implementazione metodo per l'inserimento multiplo di record tramite oggetto Session

(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

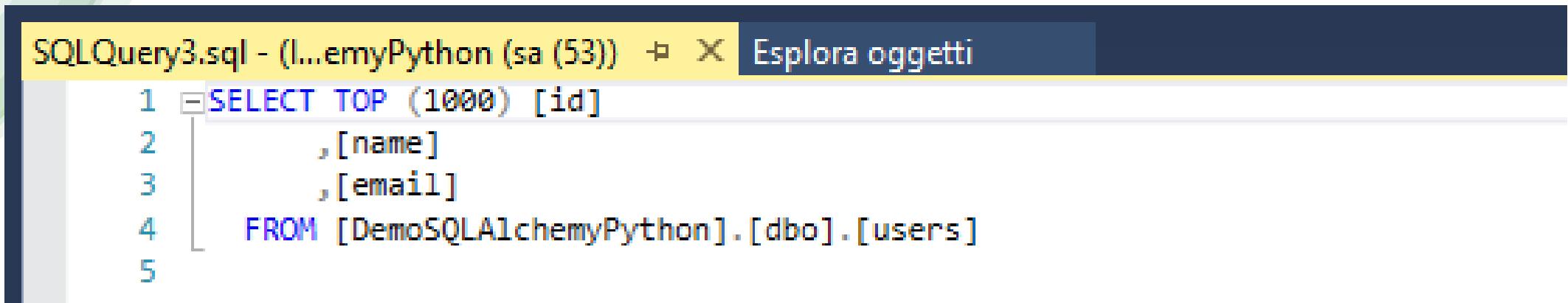
```
# Creare le tabelle nel database
CreateTable()

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

insert_data(data_to_insert)
```

CODICE MAIN DA INVOCARE DURANTE L'ESECUZIONE DELLO SCRIPT

(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



The screenshot shows a SQL Server Management Studio interface. The top bar is dark blue with the title "SQLQuery3.sql - (l...emyPython (sa (53))" and a close button. Below the title is a tab labeled "Esplora oggetti". The main area contains a code editor with the following T-SQL query:

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



The screenshot shows the results pane of SQL Server Management Studio. The title bar indicates a zoom level of "90 %". Below the title bar are two tabs: "Risultati" (Results) and "Messaggi" (Messages). The "Risultati" tab is selected and displays a table with the following data:

	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-procedure con l'uso della libreria SQLAlchemy

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

**Librerie necessarie per l'esecuzione dello script
demo_completa.py**

(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 Cliente(Base): 7 usages
    __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")
```

Definizione nel modello dell'entità Cliente

(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

```
# Entità Ordine
class Ordine(Base): 12 usages
    __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")
```

Definizione dell'entità Ordine nel modello

(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

```
#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}")
```

Procedura per la creazione del modello come entità sul database

(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

```
#Inserimento dati
# Metodo per inserire dati

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.blu@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

```
#ordini evasi

def visualizza_ordini_evasi(session):
    from sqlalchemy.orm import joinedload

    ordini_evasi = (
        session.query(Ordine)
        .options(joinedload(Ordine.cliente))
        .filter(Ordine.evaso == True)
        .all()
    )
    for ordine in ordini_evasi:
        print(f"Cliente: {ordine.cliente.nome}, Email: {ordine.cliente.email}, Ordine: {ordine.descrizione}")
```

Procedura per eseguire la query e visualizzazione degli ordini evasi

(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

```
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.")
```

Procedura per eseguire la query e visualizzazione dei 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

```
# Metodo per aggiornare le informazioni di un ordine

def aggiorna_ordine(session, ordine_id, nuova_descrizione=None, nuovo_stato=None):
    # 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.evaso = nuovo_stato
        session.commit()
        print(f"Ordine {ordine_id} aggiornato con successo.")
    else:
        print(f"Ordine {ordine_id} non trovato.")
```

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

**Classe per mappare il risultato della chiamata ad una
StoredProcedure definita sul database**

(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

```
def esegui_spOnServer(Session, idOrdine):
    # 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)
```

Script per l'esecuzione della chiamata alla stored procedure e visualizzazione dell'output

(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

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

visualizza_ordini_evasi(Session)

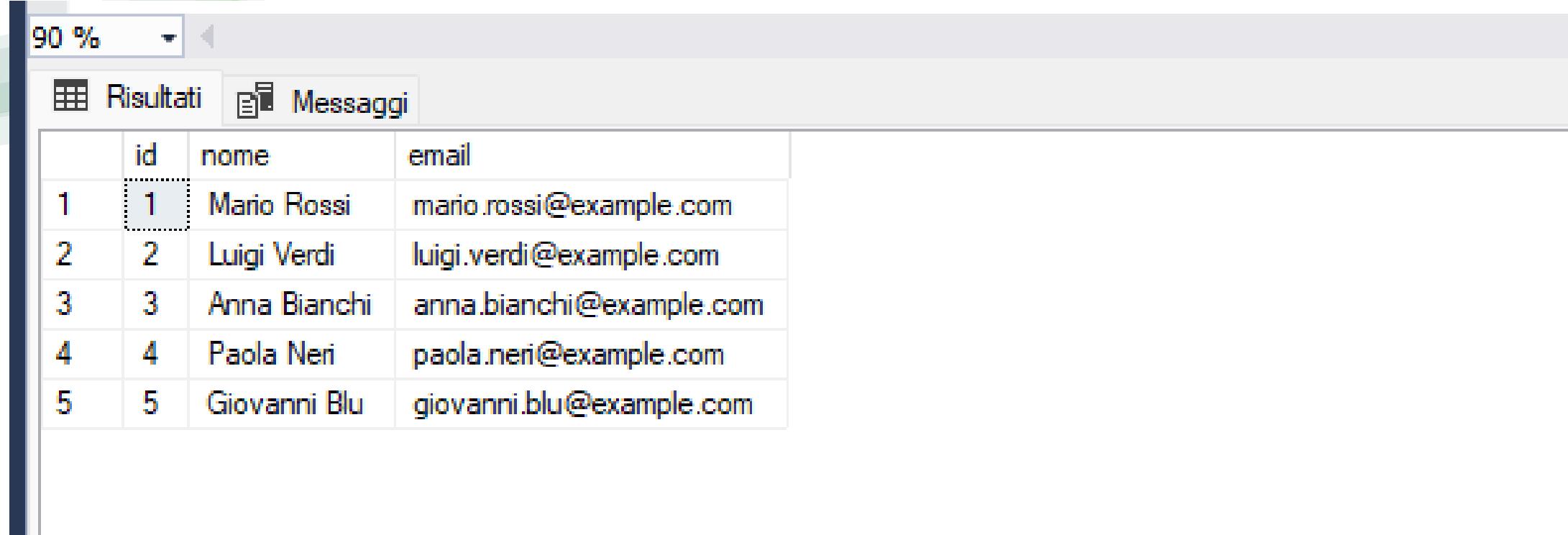
#visualizza_clienti_senza_ordini(Session)

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

#esegui_spOnServer(Session, 1)
```

**Codice per l'elenco degli script da invocare nel Main del sorgente
Demo_completa.py**

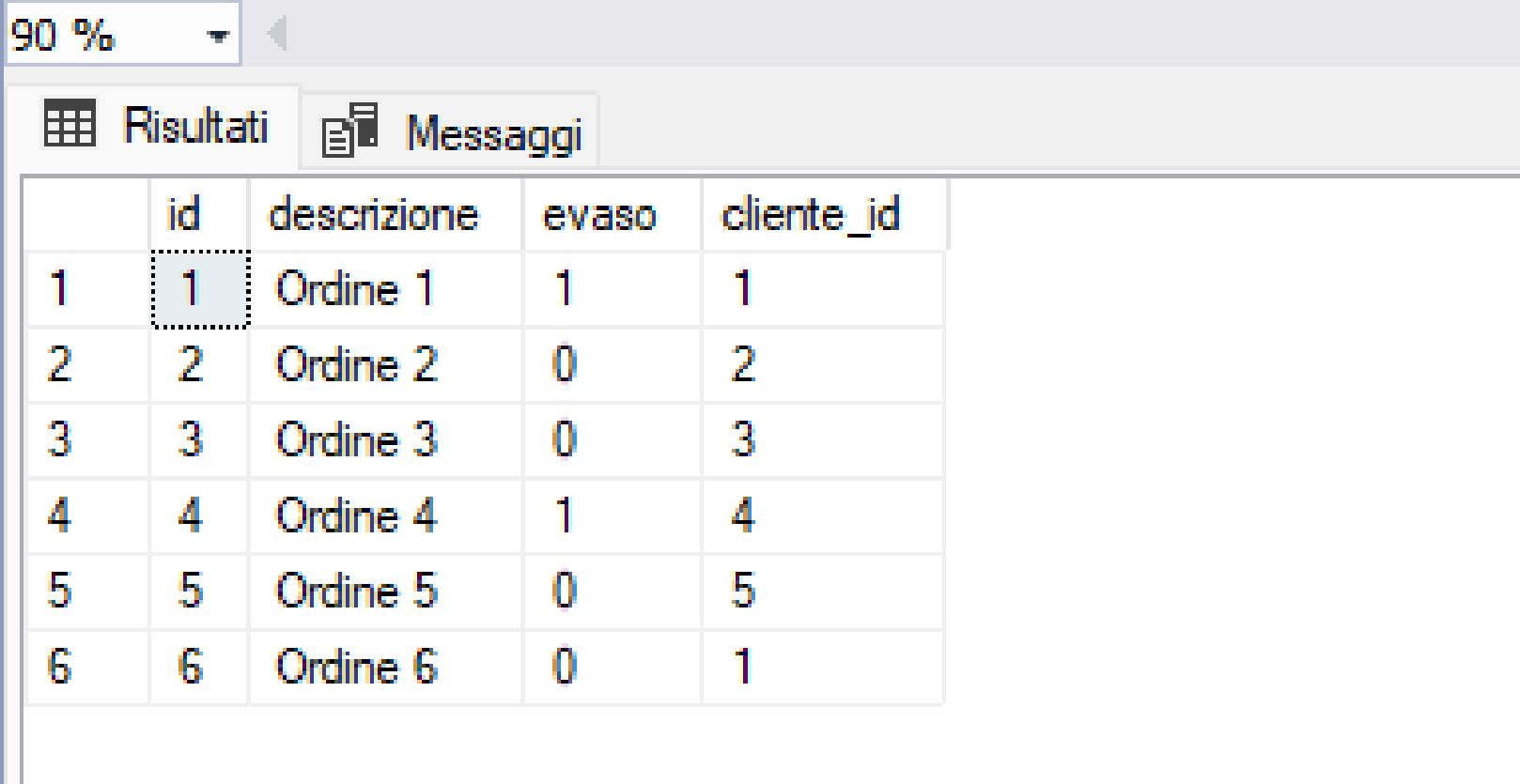
(10 minuti) CRUD – CREATE READ UPDATE DELETE con del codice Python da commentare e considerazioni sull'esecuzione delle query semplici e avanzate e calling di stored-procedure con l'uso della libreria SQLAlchemy



	id	nome	email
1	1	Mario Rossi	mario.rossi@example.com
2	2	Luigi Verdi	luigi.verdi@example.com
3	3	Anna Bianchi	anna.bianchi@example.com
4	4	Paola Neri	paola.neri@example.com
5	5	Giovanni Blu	giovanni.blu@example.com

Records per la tabella Clienti

(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



The screenshot shows a database interface with a results tab selected. The results are displayed in a grid table with columns: id, descrizione, evaso, and cliente_id. The data consists of six rows, each representing an order. The first row, where id=1 and descrizione=Ordine 1, has its id cell highlighted with a dashed border.

	id	descrizione	evaso	cliente_id
1	1	Ordine 1	1	1
2	2	Ordine 2	0	2
3	3	Ordine 3	0	3
4	4	Ordine 4	1	4
5	5	Ordine 5	0	5
6	6	Ordine 6	0	1

Records tabella 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

```
3  ALTER PROCEDURE [dbo].[sp_GetOrdineById](
4      -- Add the parameters for the stored procedure here
5      @idOrdine int)
6  AS
7  BEGIN
8      -- SET NOCOUNT ON added to prevent extra result sets from
9      -- interfering with SELECT statements.
10     SET NOCOUNT ON;
11
12     -- Insert statements for procedure here
13     SELECT c.nome, o.Evaso from Clienti c
14     join Ordini o
15     on c.Id=o.Cliente_id
16     where o.Id=@idOrdine
17 END
```

Stored procedure che poi verrà gestita in esecuzione a livello applicativo

(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

```
C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scripts\python.exe C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scripts\__init__.py
Cliente: Mario Rossi, Email: mario.rossi@example.com, Ordine: Ordine 1
Cliente: Paola Neri, Email: paola.neri@example.com, Ordine: Ordine 4

Process finished with exit code 0
```

OUTPUT TERMINALE PYCHARM ORDINI EVASI AI CLIENTI

(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

```
C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scripts\python.exe C:\Users'  
Non ci sono clienti senza ordini.  
  
Process finished with exit code 0
```

OUTPUT TERMINALE PYCHARM ESECUZIONE VISUALIZZAZIONE 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

```
C:\Users\FrancescoSpalluzzi\Desktop\Pylaboratorio\demoSQLAlchemy\Scr  
True Mario Rossi
```

```
Process finished with exit code 0
```

OUTPUT TERMINALE PER L'ESECUZIONE DELLA STORED PROCEDURE PER IDCLIENTE 1

GRAZIE 😊
Q&A

