# Mastering Python
# الدرس #13
# قواعد البيانات Database

By:

Hussam Hourani

V1.0 - DEC 2019

# Agenda

- What is Database
- SQLite Overview
- Traditional DB manipulation
- New  Object Relational Mapper

# What is SQLAlchemy & SQLite

***SQLite***3 is an in-process library that implements a transactional SQL database.

***SQLAlchemy*** is the Python SQL toolkit and **Object Relational Mapper** that gives application developers the full power and flexibility of SQL.

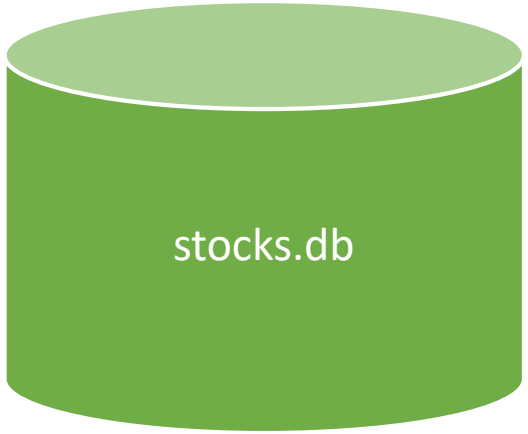
https://www.sqlalchemy.org/

https://www.sqlite.org/



**pip install pysqlite3**

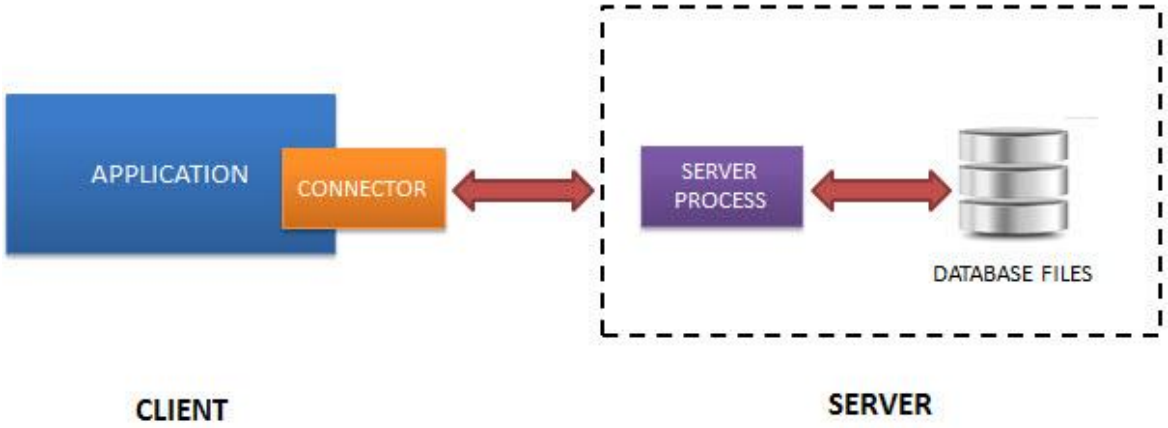
# What is Database




**A database is an organized collection of structured information, or data, typically stored electronically in a computer system.**

**A database is usually controlled by a database management system (DBMS).**

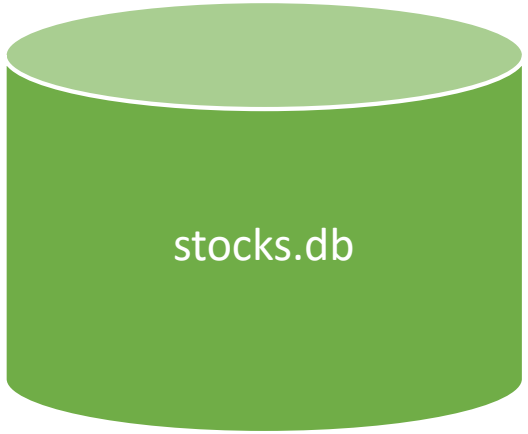
**Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.**




https://www.sqlitetutorial.net/what-is-sqlite/

https://www.oracle.com/database/what-is-database.html

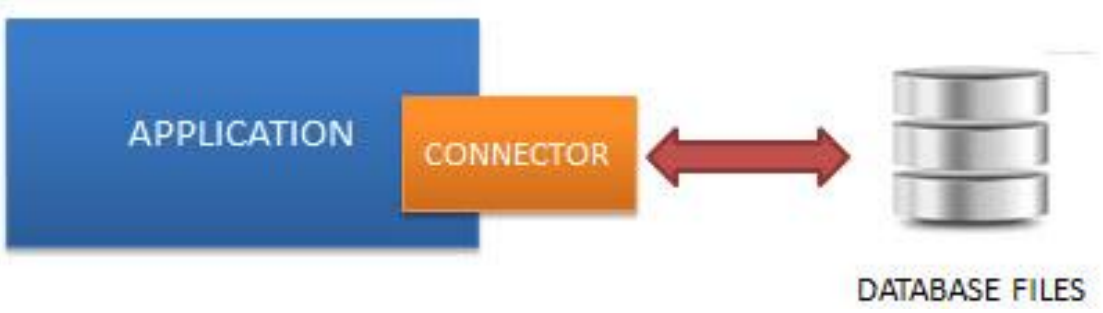
# SQLite Database




**SQLite is a software library that provides a relational database management system.**

**The lite in SQLite means light weight in terms of setup, database administration, and required resource.**

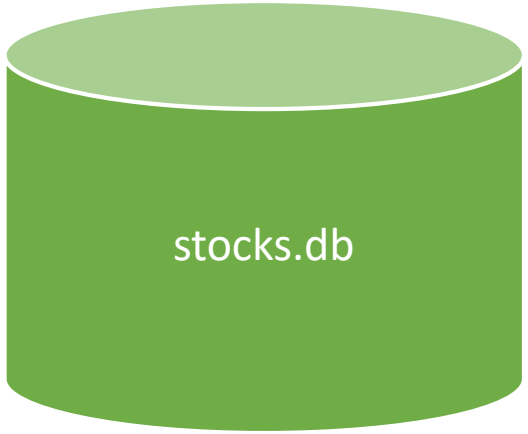
**SQLite has the following noticeable features: self-contained, serverless, zero-configuration, transactional.**




https://www.sqlitetutorial.net/what-is-sqlite/

https://www.tutorialspoint.com/sqlite/sqlite_data_types.htm

# SQLite Storage Classes, Affinity and Type Names




| Sr.No. | Storage Class & Description |
|---|---|
| 1 | **NULL**<br>The value is a NULL value. |
| 2 | **INTEGER**<br>The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value. |
| 3 | **REAL**<br>The value is a floating point value, stored as an 8-byte IEEE floating point number. |
| 4 | **TEXT**<br>The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE) |
| 5 | **BLOB**<br>The value is a blob of data, stored exactly as it was input. |

| Data Type | Affinity |
|---|---|
| •INT<br>•INTEGER<br>•TINYINT<br>•SMALLINT<br>•MEDIUMINT<br>•BIGINT<br>•UNSIGNED BIG INT<br>•INT2<br>•INT8 | **INTEGER** |
| •CHARACTER(20)<br>•VARCHAR(255)<br>•VARYING CHARACTER(255)<br>•NCHAR(55)<br>•NATIVE CHARACTER(70)<br>•NVARCHAR(100)<br>•TEXT<br>•CLOB | **TEXT** |
| •BLOB<br>•no datatype specified | **NONE** |
| •REAL<br>•DOUBLE<br>•DOUBLE PRECISION<br>•FLOAT | **REAL** |
| •NUMERIC<br>•DECIMAL(10,5)<br>•BOOLEAN<br>•DATE<br>•DATETIME | **NUMERIC** |



https://www.tutorialspoint.com/sqlite/sqlite_data_types.htm

# Traditional DB Creation : sqlite3

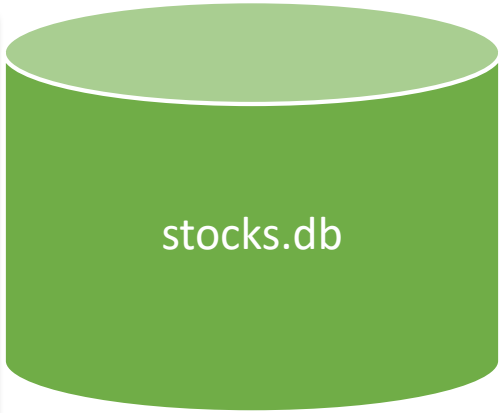
```python
import sqlite3
conn = sqlite3.connect('stocks.db')

c = conn.cursor()

# Create table
c.execute('''CREATE TABLE stocks
             (date text, trans text, symbol text, qty real, price real)''')

# Insert a row of data
c.execute("INSERT INTO stocks VALUES ('2006-01-05','BUY','RHAT',100,35.14)")
c.execute("INSERT INTO stocks VALUES ('2006-01-05','SEL','RHAT',50,35.25)")

# Save (commit) the changes
conn.commit()

# We can also close the connection if we are done with it.
# Just be sure any changes have been committed or they will be lost.
conn.close()
```

a connection is established at first

The SQLite3 cursor is a method of the connection object

The *commit()* method saves all the changes we make

output



# Showing database rescoreds

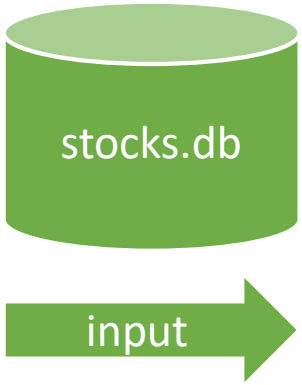

input

```
1 import sqlite3
2 conn = sqlite3.connect('stocks.db')
3
4 c = conn.cursor()
5
6 symbol = 'RHAT'
7 c.execute("SELECT * FROM stocks WHERE symbol = '%s'" % symbol)
8 for row in c:
9         print (row)
10 conn.close()
```

output

```
('2006-01-05', 'BUY', 'RHAT', 100.0, 35.14)
('2006-01-05', 'SEL', 'RHAT', 50.0, 35.25)
```

# Inserting data to the database




```python
import sqlite3
conn = sqlite3.connect('stocks.db')

c = conn.cursor()

c.execute("INSERT INTO stocks VALUES ('2006-01-05','BUY','APPLE',100,4.49)")
c.execute("INSERT INTO stocks VALUES ('2006-01-05','SEL','APPLE',50,3.23)")
c.execute("INSERT INTO stocks VALUES ('2006-01-05','BUY','APPLE',100,4.49)")
c.execute("INSERT INTO stocks VALUES ('2006-01-05','SEL','APPLE',50,3.23)")

conn.commit()

conn.close()
```

# Parsing each record


input

```python
import sqlite3
conn = sqlite3.connect('stocks.db')

c = conn.cursor()

for row in c.execute('SELECT symbol, qty , price FROM stocks ORDER BY price'):
        print (row)
conn.close()
```

output

```
('APPLE', 50.0, 3.23)
('APPLE', 50.0, 3.23)
('APPLE', 100.0, 4.49)
('APPLE', 100.0, 4.49)
('RHAT', 100.0, 35.14)
('RHAT', 50.0, 35.25)
```

# fetchall


input

```python
import sqlite3
conn = sqlite3.connect('stocks.db')

c = conn.cursor()
c.execute('SELECT symbol, qty , price FROM stocks ORDER BY price')

print (c.fetchall())
conn.close()
```

output

```
[('APPLE', 50.0, 3.23), ('APPLE', 50.0, 3.23), ('APPLE', 100.0, 4.49), ('APPLE', 100.0,
4.49), ('RHAT', 100.0, 35.14), ('RHAT', 50.0, 35.25)]
```

# Update Records


input

```python
import sqlite3
conn = sqlite3.connect('stocks.db')

c = conn.cursor()
c.execute('UPDATE stocks SET qty=10000 where symbol = "RHAT"')
conn.commit()
for row in c.execute('SELECT symbol, qty , price FROM stocks ORDER BY price'):
        print (row)
conn.close()
```

output

```
('APPLE', 50.0, 3.23)
('APPLE', 50.0, 3.23)
('APPLE', 100.0, 4.49)
('APPLE', 100.0, 4.49)
('RHAT', 10000.0, 35.14)
('RHAT', 10000.0, 35.25)
```

# Delete Records

```
('APPLE', 50.0, 3.23)
('APPLE', 50.0, 3.23)
('APPLE', 100.0, 4.49)
('APPLE', 100.0, 4.49)
('RHAT', 10000.0, 35.14)
('RHAT', 10000.0, 35.25)
```

input

```python
import sqlite3
conn = sqlite3.connect('stocks.db')

c = conn.cursor()
c.execute('DELETE from stocks where symbol = "APPLE" and price=4.49')
conn.commit()
for row in c.execute('SELECT symbol, qty , price FROM stocks ORDER BY price'):
        print (row)
conn.close()
```

output

```
('APPLE', 50.0, 3.23)
('APPLE', 50.0, 3.23)
('RHAT', 10000.0, 35.14)
('RHAT', 10000.0, 35.25)
```

# Row counts


input

```python
import sqlite3
conn = sqlite3.connect('stocks.db')

c = conn.cursor()
print(c.execute('SELECT * FROM stocks').rowcount)
rows = c.fetchall()

print (len (rows))
conn.close()
```

output

```
-1
4
```

# Select tables names

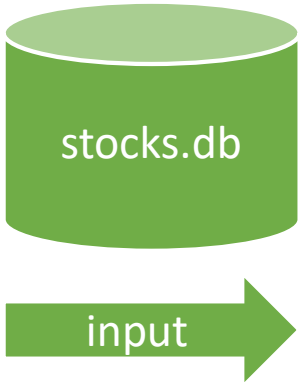

input

```python
import sqlite3
conn = sqlite3.connect('stocks.db')

c = conn.cursor()
c.execute('SELECT name from sqlite_master where type= "table"')
print(c.fetchall())
conn.close()
```

output

```
[('stocks',)]
```

# Others




```
drop table table_name

drop table if exists table_name

create table if not exists table_name (column1, column2, …, columnN)

print(row) for row in cursorObj.fetchall()
```

# Creating New DB

```python
import sqlite3

conn = sqlite3.connect('test.db')
print ("Opened database successfully")

conn.execute('''CREATE TABLE COMPANY
         (ID INT PRIMARY KEY     NOT NULL,
         NAME           TEXT    NOT NULL,
         AGE            INT     NOT NULL,
         ADDRESS        CHAR(50),
         SALARY         REAL);''')
print ("Table created successfully")

conn.close()
```

output


output

```
Opened database successfully
Table created successfully
```

# Traditional DB Creation : sqlite3


input

```python
import sqlite3

conn = sqlite3.connect('test.db')
print ("Opened database successfully")

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (1, 'Paul', 32, 'California', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

conn.commit()
print ("Records created successfully")
conn.close()
```

output

```
Opened database successfully
Records created successfully
```

# Traditional DB Creation : sqlite3


input

```python
import sqlite3

conn = sqlite3.connect('test.db')
print ("Opened database successfully")

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print ("ID = ", row[0])
   print ("NAME = ", row[1])
   print ("ADDRESS = ", row[2])
   print ("SALARY = ", row[3], "\n")

print ("Operation done successfully")
conn.close()
```

output

```
Opened database successfully
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  20000.0

ID =  2
NAME =  Allen
ADDRESS =  Texas
SALARY =  15000.0

ID =  3
NAME =  Teddy
ADDRESS =  Norway
SALARY =  20000.0

ID =  4
NAME =  Mark
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

# Update


input

```python
import sqlite3

conn = sqlite3.connect('test.db')
print ("Opened database successfully")

conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID = 1")
conn.commit()
print ("Total number of rows updated :", conn.total_changes)

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print ("ID = ", row[0])
   print ("NAME = ", row[1])
   print ("ADDRESS = ", row[2])
   print ("SALARY = ", row[3], "\n")

print ("Operation done successfully")
conn.close()
```

output

```
Total number of rows updated : 1
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  25000.0

ID =  2
NAME =  Allen
ADDRESS =  Texas
SALARY =  15000.0

ID =  3
NAME =  Teddy
ADDRESS =  Norway
SALARY =  20000.0

ID =  4
NAME =  Mark
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

# Delete


input

```python
import sqlite3

conn = sqlite3.connect('test.db')
print ("Opened database successfully")

conn.execute("DELETE from COMPANY where ID = 2;")
conn.commit()
print ("Total number of rows deleted :", conn.total_changes)

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print ("ID = ", row[0])
   print ("NAME = ", row[1])
   print ("ADDRESS = ", row[2])
   print ("SALARY = ", row[3], "\n")

print ("Operation done successfully")
conn.close()
```

output

```
Total number of rows updated : 1
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  25000.0

ID =  2
NAME =  Allen
ADDRESS =  Texas
SALARY =  15000.0

ID =  3
NAME =  Teddy
ADDRESS =  Norway
SALARY =  20000.0

ID =  4
NAME =  Mark
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

# Sample Tables

**person**

| | Name | Type | Nullable |
|---|---|---|---|
| | id | Int | No |
| | name | String(250) | No |

**address**

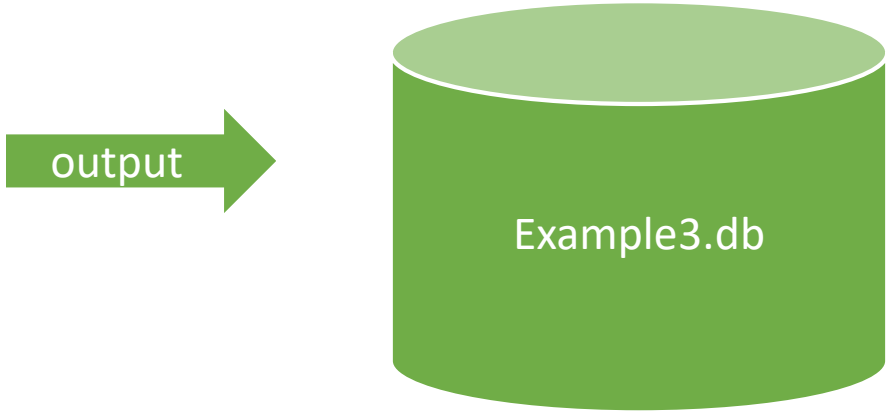
| | Name | Type | Nullable |
|---|---|---|---|
| | id | Int | No |
| | street_name | String(250) | Yes |
| | street_number | String(250) | Yes |
| | post_code | String(250) | No |
| | person_id | Int | No |



https://www.pythoncentral.io/introductory-tutorial-python-sqlalchemy/

# Traditional DB Creation : sqlite3

```python
import sqlite3
conn = sqlite3.connect('example3.db')

c = conn.cursor()
c.execute('''
          CREATE TABLE person
          (id INTEGER PRIMARY KEY ASC, name varchar(250) NOT NULL)
          ''')
c.execute('''
          CREATE TABLE address
          (id INTEGER PRIMARY KEY ASC, street_name varchar(250),
           street_number varchar(250),
           post_code varchar(250) NOT NULL, person_id INTEGER NOT NULL,
           FOREIGN KEY(person_id) REFERENCES person(id))
          ''')

c.execute('''
          INSERT INTO person VALUES(1, 'pythoncentral')
          ''')
c.execute('''
          INSERT INTO address VALUES(1, 'python road', '1', '00000', 1)
          ''')
c.execute('''
          INSERT INTO address VALUES(2, 'Hussam', '2', '10000', 1)
          ''')

conn.commit()
conn.close()
```




**person**

| Name | Type | Nullable |
|---|---|---|
| id | Int | No |
| name | String(250) | No |

**address**

| Name | Type | Nullable |
|---|---|---|
| id | Int | No |
| street_name | String(250) | Yes |
| street_number | String(250) | Yes |
| post_code | String(250) | No |
| person_id | Int | No |



https://www.pythoncentral.io/introductory-tutorial-python-sqlalchemy/

# Traditional DB Creation : sqlite3


Read DB

**person**

| Name | Type | Nullable |
|---|---|---|
| id | Int | No |
| name | String(250) | No |

**address**

| Name | Type | Nullable |
|---|---|---|
| id | Int | No |
| street_name | String(250) | Yes |
| street_number | String(250) | Yes |
| post_code | String(250) | No |
| person_id | Int | No |

```python
import sqlite3
conn = sqlite3.connect('example3.db')

c = conn.cursor()
c.execute('SELECT * FROM person')
print (c.fetchall())

c.execute('SELECT * FROM address')
print (c.fetchall())
conn.close()
```

Output

```
[(1, 'pythoncentral')]
[(1, 'python road', '1', '00000', 1), (2, 'Hussam', '2', '10000', 1)]
```



https://www.pythoncentral.io/introductory-tutorial-python-sqlalchemy/

# New Way & SQLAlchemy's declarative

## Mapping Tables to Classes

**person**

| | Name | Type | Nullable |
|---|---|---|---|
| | id | Int | No |
| | name | String(250) | No |

**address**

| | Name | Type | Nullable |
|---|---|---|---|
| | id | Int | No |
| | street_name | String(250) | Yes |
| | street_number | String(250) | Yes |
| | post_code | String(250) | No |
| | person_id | Int | No |

Map

**map Person to "Person class."**

Map

**map Address to Address class.**

output

sqlalchemy_example.db

# SQLAlchemy's declarative

```python
import os
import sys
from sqlalchemy import Column, ForeignKey, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy import create_engine

Base = declarative_base()

class Person(Base):
    __tablename__ = 'person'
    id = Column(Integer, primary_key=True)
    name = Column(String(250), nullable=False)

class Address(Base):
    __tablename__ = 'address'
    id = Column(Integer, primary_key=True)
    street_name = Column(String(250))
    street_number = Column(String(250))
    post_code = Column(String(250), nullable=False)
    person_id = Column(Integer, ForeignKey('person.id'))
    person = relationship(Person)

engine = create_engine('sqlite:///sqlalchemy_example.db')

Base.metadata.create all(engine)
```

output


**erson**

| Name | Type | Nullable |
|---|---|---|
| id | Int | No |
| name | String(250) | No |

**address**

| Name | Type | Nullable |
|---|---|---|
| id | Int | No |
| street_name | String(250) | Yes |
| street_number | String(250) | Yes |
| post_code | String(250) | No |
| person_id | Int | No |



https://www.pythoncentral.io/introductory-tutorial-python-sqlalchemy/

# SQLAlchemy's : Inserting Data

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

from sqlalchemy3 import Address, Base, Person

engine = create_engine('sqlite:///sqlalchemy_example.db')
Base.metadata.bind = engine

DBSession = sessionmaker(bind=engine)

session = DBSession()

# Insert a Person in the person table
new_person = Person(name='hussam Hourani')
session.add(new_person)
session.commit()

# Insert an Address in the address table
new_address = Address(post_code='00000', person=new_person)
session.add(new_address)
session.commit()
```

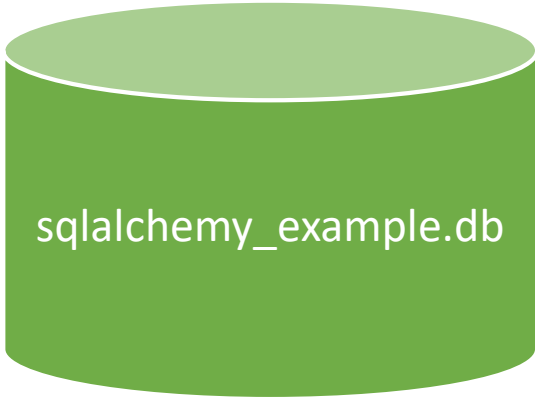

**person**

| Name | Type | Nullable |
|---|---|---|
| id | Int | No |
| name | String(250) | No |

**address**

| Name | Type | Nullable |
|---|---|---|
| id | Int | No |
| street_name | String(250) | Yes |
| street_number | String(250) | Yes |
| post_code | String(250) | No |
| person_id | Int | No |

# SQLAlchemy's : Reading Data




Read

```python
from sqlalchemy3 import Person, Base, Address
from sqlalchemy import create_engine

engine = create_engine('sqlite:///sqlalchemy_example.db')
Base.metadata.bind = engine

from sqlalchemy.orm import sessionmaker
DBSession = sessionmaker()
DBSession.bind = engine
session = DBSession()

person = session.query(Person).first()
print( "Name :", person.name)

address = session.query(Address).filter(Address.person == person).one()
address.post_code
print ("address.id: ", address.id )
print ("address.post_code:", address.post_code )
```

output

```
Name : hussam Hourani
address.id:  1
address.post_code: 00000
```


Master in Software Engineering

Hussam Hourani has over 25 years of Organizations Transformation, VROs, PMO, Large Scale and Enterprise Programs Global Delivery, Leadership, Business Development and Management Consulting. His client experience is wide ranging across many sectors but focuses on Performance Enhancement, Transformation, Enterprise Program Management, Artificial Intelligence and Data Science.