Introducción

Durante su formación académica ha tenido la oportunidad de interactuar con base de datos relacionales y base de datos no relacionales. En este apartado se retoman conceptos de base de datos relacionales y se aborda una nueva temática que tiene relación con el manejo de datos a través de Object Relational Mapper o conocido por sus iniciales ORM.

Introducción

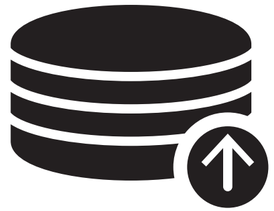
En referencia a Object Relational Mapper (ORM), Cabedo et al (2010), indica que es una técnica que permite a desarrolladores pasar los datos de lenguajes de programación bajo el paradigma de Orientación a Objetos a datos persistentes para su posterior almacenamiento en bases de datos relacionales.

```python
class Docente(Base):
    __tablename__ = 'docentes'
    id = Column(Integer, primary_key=True)
    nombre = Column(String)
    apellido = Column(String)
    ciudad = Column(String, nullable=False)

    def __repr__(self):
        return "Docente: nombre=%s apellido=%s ciudad:%s" % (
                            self.nombre,
                            self.apellido,
                            self.ciudad)
```

Table: docentes

	id	nombre	apellido	ciudad
	Filter	Filter	Filter	Filter
1	1	Tony	García	Loja
2	2	Luis	Borrero	Loja
3	3	Ana	Salcedo	Zamora



Ventajas del uso de ORM

- Proceso de desarrollo más rápido.
- Separación de las bases de datos; cuando se utiliza un ORM se puede cambiar el motor de base de datos en cualquier momento.
- Seguridad, en los ORM existen procedimientos que impiden ataques como SQL injections

```python
class Docente(Base):
    __tablename__ = 'docentes'
    id = Column(Integer, primary_key=True)
    nombre = Column(String)
    apellido = Column(String)
    ciudad = Column(String, nullable=False)

    def __repr__(self):
        return "Docente: nombre=%s apellido=%s ciudad:%s" % (
                        self.nombre,
                        self.apellido,
                        self.ciudad)
```

python

Table: docentes

	id	nombre	apellido	ciudad
	Filter	Filter	Filter	Filter
1	1	Tony	García	Loja
2	2	Luis	Borrero	Loja
3	3	Ana	Salcedo	Zamora

Limitaciones del uso de ORM

- Curva de aprendizaje, es difícil explotar en formar completa un ORM por la extensión de las mismas; por tal razón, se debe tomar un tiempo considerable para dominar y aplicarlo en una solución.
- Aplicaciones ralentizadas, la aplicación de ORM genera que los procesos sean más lentos en un porcentaje bajo; se debe considerar y sopesar con la velocidad del desarrollo.

```python
class Docente(Base):
    __tablename__ = 'docentes'
    id = Column(Integer, primary_key=True)
    nombre = Column(String)
    apellido = Column(String)
    ciudad = Column(String, nullable=False)

    def __repr__(self):
        return "Docente: nombre=%s apellido=%s ciudad:%s" % (
                            self.nombre,
                            self.apellido,
                            self.ciudad)
```

python

Table: docentes

	id	nombre	apellido	ciudad
	Filter	Filter	Filter	Filter
1	1	Tony	García	Loja
2	2	Luis	Borrero	Loja
3	3	Ana	Salcedo	Zamora

Librerías ORM según los lenguajes de programación

Lenguaje de programación PHP

- Doctrine
- Eloquent
- Active Record Class 36

Lenguaje de programación Java

- Hibernate
- Ebean

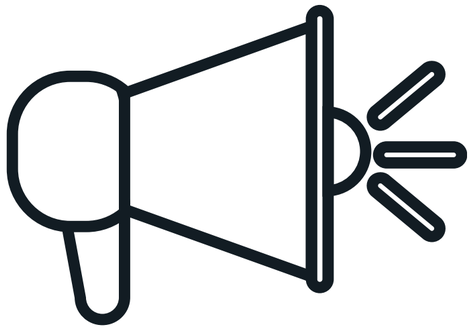
Lenguaje de programación Ruby

- Active Record
- Ruby Object Mapper
- Sequel

Lenguaje de programación Python

- SqlAlchemy
- Django-ORM
- Pony-ORM

Manejo de datos con ORM SqlAlchemy

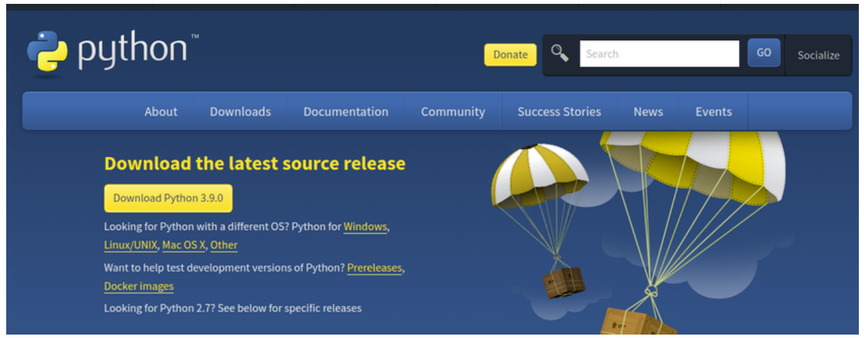


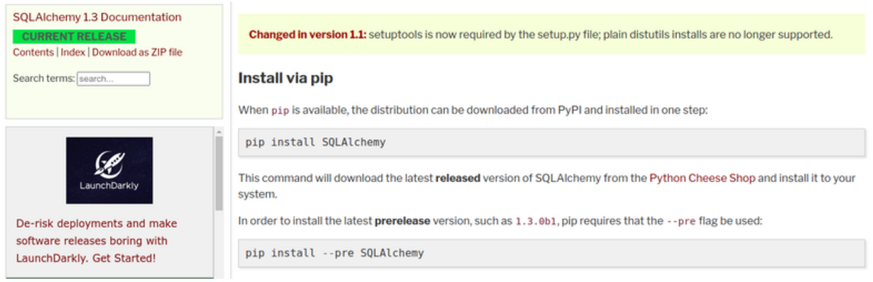
En el presente apartado se va a ejemplificar el uso de la librería ORM denominada SqlAlchemy que está desarrollada en lenguaje Python. El recurso educativo abierto denominado **Learning SqlAlchemy** será usado como base para el aprendizaje de la librería.

Manejo de datos con ORM SqlAlchemy

Se necesita instalar:

- Lenguaje de programación Python (puede revisar el enlace que permite la descarga e instalación en diversos sistemas operativos - https://www.python.org/downloads/).
- Instalación de la librería SqlAlchemy (revisar proceso de instalación); se recomienda usar el proceso vía el gestor de librerías de Python denominado pip. (https://docs.sqlalchemy.org/en/13/intro.html#installation-guide)

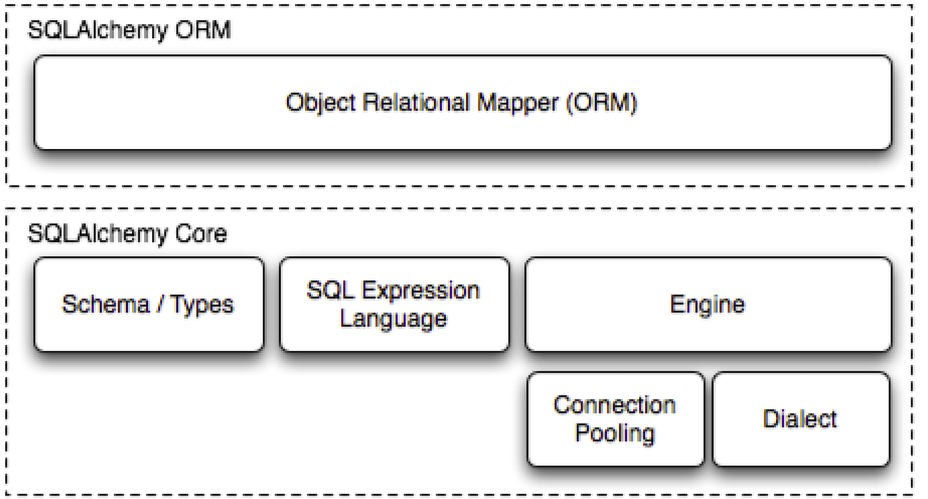




Manejo de datos con ORM SqlAlchemy

SqlAlchemy está dividido en dos grandes funcionalidades y áreas como lo indica el recurso Learning sqlAlchemy; la relacionada con SqlAlchemy Core y SqlAlchemy ORM. Se revisará la segunda área en la presente guía.

La SqlAlchemy ORM permite manejar un patrón que da la posibilidad de pasasr las clases desarrolladas en Python a un base de datos de forma simple y elegante



Manejo de datos con ORM SqlAlchemy

Manifestar que los ejemplos siguientes pueden ser desarrollados o probados de dos formas

- Agregando el código a través de un editor de texto en un archivo con extensión py;

y, luego ejecutar desde un terminal o consola con el patrón: python [nombre_archivo].py

- Usando la consola por defecto de python o la librería ipython

```
Python 3.6.8 (default, Oct  7 2019, 12:59:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
Type "copyright", "credits" or "license" for more information.

IPython 5.6.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Manejo de datos con ORM SqlAlchemy

Ejemplos de conexiones para algunos gestores de base de datos

Postgres

```
# por defecto
create_engine('postgresql://usuario:clave@localhost:5432/basedatos')

# usando la librería psycopg2
create_engine('postgresql+psycopg2://usuario:clave@localhost:5432/basedatos')
```

Mysql

```
# por defecto
create_engine (‘mysql://usuario:clave@localhost/base-de-datos’)

# usando la librería de Python mysqlclient 51
create_engine(‘mysql+mysql://usuario:clave@localhost/base-de-datos’)
```

Oracle

```
# por defecto
create_engine(‘oracle://usuario:clave@127.0.0.1:1521/esquema’)

# usando la librería de python 52
create_engine(‘oracle+cx_oracle://usuario:clave@esquema’)
```

Para hacer uso de la librería existen algunas consideraciones:

Conectarnos a la base de datos a través del siguiente código

```
1 from sqlalchemy import create_engine
2
3 # se genera enlace al gestor de base de
4 # datos
5 # para el ejemplo se usa la base de datos
6 # sqlite
7 engine = create_engine('sqlite:///demobase.db')
```

- Se importa el módulo create_engine (línea 1) que permite el enlace hacia un motor de base de datos.
- En la línea 7 se crea un variable llamada engine que hace uso de create_engine, a la cual se envía como parámetro una cadena de texto que indica el motor a usar.

- Para el ejemplo se usa un enlace para una base de datos SQLite, el nombre asignado para la base de datos es demobase.db

Para hacer uso de la librería existen algunas consideraciones:

Conectarnos a la base de datos a través del siguiente código

```
 8
 9 from sqlalchemy.ext.declarative import declarative_base
10 Base = declarative_base()
11
```

Declaración de las clases, mismas que en lo posterior se transforman en tablas de la base de datos

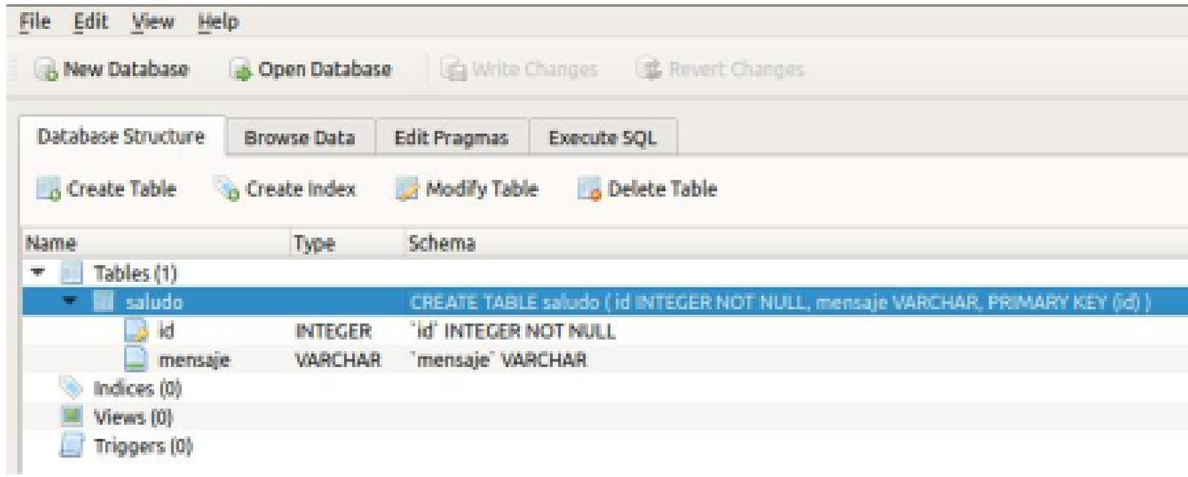
- **Para realizar el proceso se necesita importar la función declarative_base como se observa en la línea 9.**
- **En la línea 10 se crea una clase llamada Base para definir las clases en lenguaje Python.**

Para hacer uso de la librería existen algunas consideraciones:

Conectarnos a la base de datos a través del siguiente código

La clase Base se usa como superclase para todas las clases que se necesitan inicializar.

```
13 from sqlalchemy import Column, Integer, String
14
15 class Saludo(Base):
16     __tablename__ = 'saludo'
17     id = Column(Integer, primary_key=True)
18     mensaje = Column(String)
19
20  Base.metadata.create_all(engine)
```



Para hacer uso de la librería existen algunas consideraciones:

Ahora, se revisará algunos puntos para guardar información en la base de datos y entidad recién creada.

```
22 from sqlalchemy.orm import sessionmaker
23
24 Session = sessionmaker(bind=engine)
25 session = Session()
```

Algunas explicaciones, sessionmaker, es una clase generadora de clases de tipo Sesion, se usa como configuración los parámetros enviados a través del constructor. Para el ejemplo, se envía el enlace creado para la base de datos, engine.

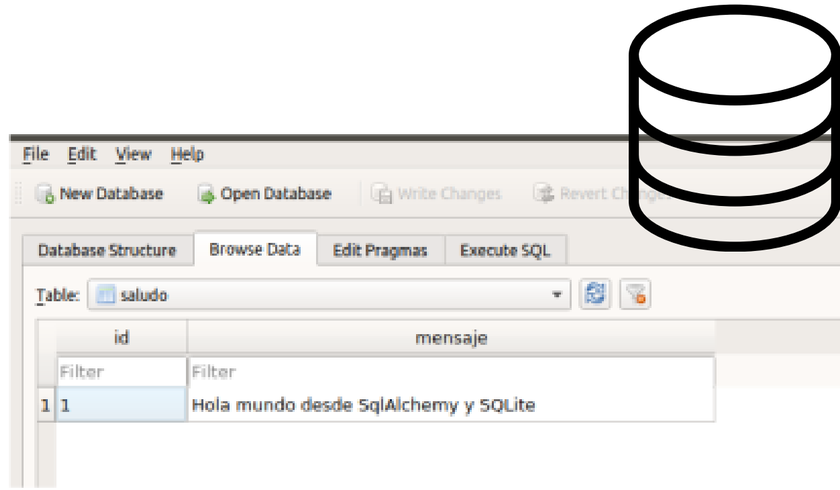
En la línea 24 se crea una clase de Python llamada Sesión, desde el generador sessionmaker.

En la línea 25 se crea un objeto session de tipo Session, mismo que va a permitir guardar, eliminar, actualizar, generar consultas en la base de datos respecto a las entidades creadas.

Para hacer uso de la librería existen algunas consideraciones:

Ahora se deja unas líneas de código que permiten crear un objeto de tipo Saludo y guardar dicho objeto como registro en la base de datos.

```
30 miSaludo = Saludo()
31 miSaludo.mensaje = "Hola mundo desde SqlAlchemy y SQLite"
32
33 # se agrega el objeto miSaludo
34 # a la entidad Saludo a la sesión
35 # a la espera de un commit
36 # para agregar un registro a la base de
37 # datos demobase.db
38 session.add(miSaludo)
39
40 # se confirma las transacciones
41 session.commit()
```



Consulta de datos con ORM SqlAlchemy

Para realizar la consulta de información a una base de datos a través del ORM de SqlAlchemy se deben considerar algunos puntos:

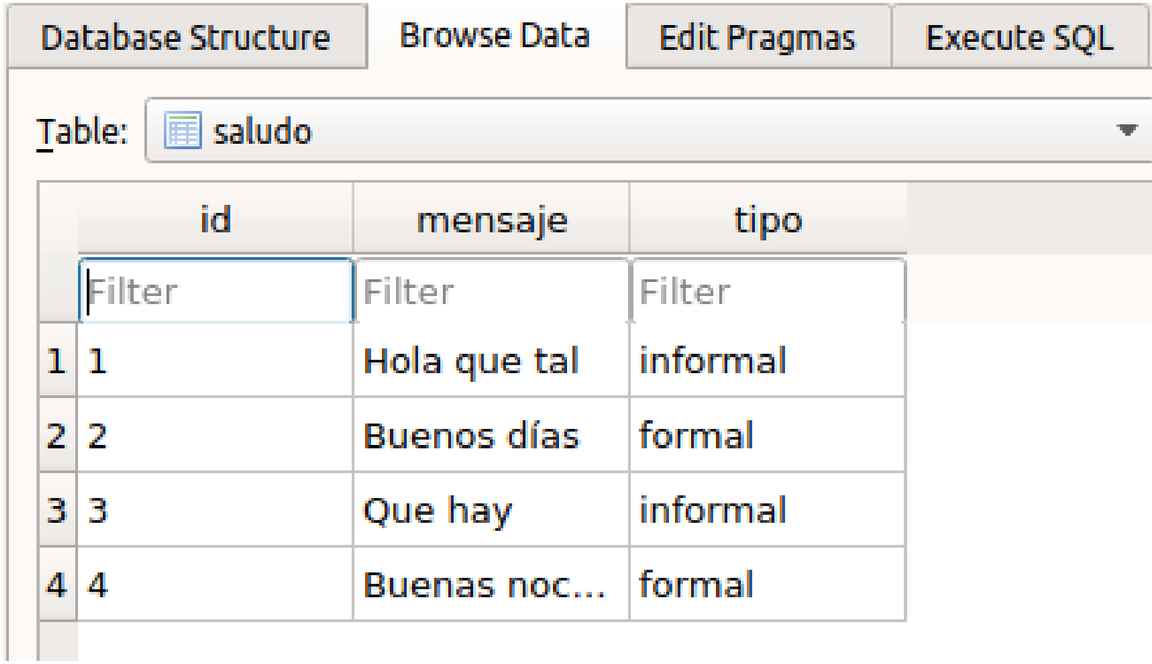
- Usar la variable session.
- Se usa el método query(), se accede desde session.
- Al método query se le puede enviar como argumentos clases o características de una clase.
- Al método query se le agrega algunas opciones como: all, order_by, filter, filter_by

Manejo de datos con ORM SqlAlchemy

Ejemplo 1:

```
1 from sqlalchemy import create_engine
2 from sqlalchemy.ext.declarative import declarative_base
3 from sqlalchemy.orm import sessionmaker
4 from sqlalchemy import Column, Integer, String
5
6 # se genera en enlace al gestor de base de
7 # datos
8 # para el ejemplo se usa la base de datos
9 # sqlite
10 engine = create_engine('sqlite:///demobase2.db')
11
12 Base = declarative_base()
13
14 class Saludo(Base):
15     __tablename__ = 'saludo'
16     #  __table_args__ = {'extend_existing': True}
17     id = Column(Integer, primary_key=True)
18     mensaje = Column(String)
19     tipo = Column(String)
20
21 Base.metadata.create_all(engine)
```

Se crea la clase y se crea la entidad en la base de datos



Manejo de datos con ORM SqlAlchemy

Ejemplo 1:

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

# se importa la clase(s) del
# archivo genera_tablas
from genera_tablas import Saludo

# se genera en enlace al gestor de base de
# datos
# para el ejemplo se usa la base de datos
# sqlite
engine = create_engine('sqlite:///demobase2.db')

Session = sessionmaker(bind=engine)
session = Session()

# se crea un objetos de tipo Saludo

saludo1 = Saludo()
saludo1.mensaje = "Hola que tal"
saludo1.tipo = "informal"

saludo2 = Saludo()
saludo2.mensaje = "Buenos días"
saludo2.tipo = "formal"

saludo3 = Saludo()
saludo3.mensaje = "Que hay"
saludo3.tipo= "informal"

saludo4 = Saludo()
saludo4.mensaje = "Buenas noches"
saludo4.tipo = "formal"
```

```python
# se agrega los objetos de tipo Saludo
# a la sesión
# a la espera de un commit
# para agregar un registro a la base de
# datos demobase2.db
session.add(saludo1)
session.add(saludo2)
session.add(saludo3)
session.add(saludo4)

# se confirma las transacciones
session.commit()
```

Se crea las instancias (registros) y se confirma los procesos para ver reflejados los datos en la base de datos

```
Hola que tal
informal
--------
Buenos días
formal
--------
Que hay
informal
--------
Buenas noches
formal
--------
```

Manejo de datos con ORM SqlAlchemy

Consulta de todos los datos o registros de una entidad

.all()

session.query(Saludo).all()

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

# se importa la clase(s) del
# archivo genera_tablas
from genera_tablas import Saludo

# se genera en enlace al gestor de base de
# datos
# para el ejemplo se usa la base de datos
# sqlite
engine = create_engine('sqlite:///demobase2.db')

Session = sessionmaker(bind=engine)
session = Session()

# Obtener todos los registros de
# la tabla saludo
losSaludos = session.query(Saludo).all()
# la consulta con .all(), devuelve
# una lista de objetos de tipo Saludo
# que se le asigna como valor a la variable
# losSaludos
# Se recorre la lista a través de un ciclo
# repetitivo for en python

for s in losSaludos:
    print(s.mensaje)
    print(s.tipo)
    print("---------")
```

Selección de todos los registros de la clase Saludo.

Manejo de datos con ORM SqlAlchemy

Consulta de ordenados en función de un atributo de la entidad o clase

.order_by(Entidad.atributo)

session.query(Saludo).order_by(Saludo.mensaje)

Selección de todos los registros de la clase Saludo, ordenados en función del atributo mensaje

```
1 from sqlalchemy import create_engine
2 from sqlalchemy.orm import sessionmaker
3
4 from genera_tablas import Saludo
5
6 # datos
7 # para el ejemplo se usa la base de datos
8 # sqlite
9 engine = create_engine('sqlite:///demobase2.db')
10
11 Session = sessionmaker(bind=engine)
12 session = Session()
13
14 # Obtener todos los registros de
15 # la tabla saludo ordenados por el atribut
16 # mensaje
17 losSaludos = session.query(Saludo).order_by(Saludo.mensaje)
18 # la consulta con .order_by, ordena los resultados en función del
19 # atributo de la clase Saludo, mensaje,
20 # Devuelve una lista de objetos de tipo Saludo
21 # y se le asigna como valor a la variable
22 # losSaludos
23
24 # Se recorre la lista a través de un ciclo
25 # repetitivo for en python
26
27 for s in losSaludos:
28     print("Mensaje: %s" % (s.mensaje))
29     print("Tipo: %s" % (s.tipo))
30     print("id: %s" % (s.id))
31     print("---------")
```

```
Mensaje: Buenas noches
Tipo: formal
id: 4
--------
Mensaje: Buenos días
Tipo: formal
id: 2
--------
Mensaje: Hola que tal
Tipo: informal
id: 1
--------
Mensaje: Que hay
Tipo: informal
id: 3
--------
```

Gracias

rrelizalde@utpl.edu.ec
@reroes