REGIONAL ANTIOQUIA
CENTRO DE SERVICIOS Y GESTION EMPRESARIAL
TECNOLOGIA EN ANALISIS Y DESARROLLO DE SOFTWARE
Programación Orientada Objetos (Python + Django)

CONCEPTOS DE ORM

¿Qué es un ORM?

Un **ORM** (Object-Relational Mapper) es una técnica de programación que permite interactuar con bases de datos relacionales utilizando objetos de programación en lugar de escribir directamente consultas SQL. En otras palabras, un ORM mapea las tablas de la base de datos a clases y las filas de esas tablas a instancias de esas clases.

¿Cómo funciona un ORM en Python?

En Python, existen varios ORM disponibles, siendo **SQLAlchemy** y **Django ORM** los más populares. Ambos permiten interactuar con bases de datos de manera más sencilla y eficiente.

Principales ventajas del uso de un ORM:

1. **Abstracción de SQL**: No necesitas escribir SQL manualmente para las operaciones básicas. El ORM se encarga de convertir tus métodos y atributos de Python en sentencias SQL.
2. **Portabilidad entre bases de datos**: Si tu aplicación está diseñada utilizando un ORM, es más sencillo cambiar entre diferentes sistemas de gestión de bases de datos (DBMS), como MySQL, PostgreSQL o SQLite, sin tener que reescribir toda la lógica SQL.
3. **Seguridad**: Los ORM ayudan a evitar problemas como inyecciones SQL, ya que generan consultas SQL de manera segura y parametrizada.
4. **Menos código repetitivo**: El ORM facilita la creación de la base de datos, las relaciones entre tablas, y las consultas, lo que reduce el código que tendrías que escribir manualmente.

Componentes básicos de un ORM:

1. **Modelo**: Representa una tabla en la base de datos. Cada clase que defines con un ORM corresponde a una tabla, y cada atributo de esa clase a una columna de la tabla.
2. **Consultas**: Los ORM permiten realizar consultas a la base de datos utilizando el mismo lenguaje que se usa para interactuar con las clases en Python.

Documento elaborado por: Alvaro Pérez Niño - Instructor

CONCEPTOS DE ORM

3. **Relaciones**: Los ORM también manejan relaciones entre tablas, como las relaciones uno a muchos, muchos a muchos, y muchos a uno.

ORM en Python más populares:

1. ***SQLAlchemy***

SQLAlchemy es uno de los ORM más completos y flexibles para Python. Ofrece dos modos de uso:

- **SQLAlchemy ORM**: Te permite trabajar con objetos y tablas, y realizar consultas de manera declarativa.
- **SQLAlchemy Core**: Proporciona una forma más cercana a SQL puro, pero aún con ciertas abstracciones.

Instalación:

```
pip install sqlalchemy
```

Ejemplo básico de cómo definir un modelo y realizar consultas con SQLAlchemy:

```python
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

# Base para la definición de las clases
Base = declarative_base()

# Definición del modelo (Tabla)
class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String)

# Crear una base de datos en memoria
```

CONCEPTOS DE ORM

```python
engine = create_engine('sqlite:///:memory:')

# Crear las tablas
Base.metadata.create_all(engine)

# Crear una sesión
Session = sessionmaker(bind=engine)
session = Session()

# Insertar datos
new_user = User(name='Alice')
session.add(new_user)
session.commit()

# Consultar datos
users = session.query(User).all()
for user in users:
    print(user.name)
```

2. ***Django ORM***

Django es un framework de desarrollo web que incluye su propio ORM. Es muy popular en el desarrollo web con Python y te permite crear aplicaciones web completas de manera rápida.

Instalación:

```
pip install django
```

Ejemplo básico de cómo definir un modelo en Django y realizar consultas:

REGIONAL ANTIOQUIA
CENTRO DE SERVICIOS Y GESTION EMPRESARIAL
TECNOLOGIA EN ANALISIS Y DESARROLLO DE SOFTWARE
Programación Orientada Objetos (Python + Django)

CONCEPTOS DE ORM

```python
from django.db import models

# Definición del modelo (Tabla)
class User(models.Model):
    name = models.CharField(max_length=100)

# Consultas
users = User.objects.all()
for user in users:
    print(user.name)
```

Relación de tablas en los ORM

Una de las grandes ventajas de los ORM es cómo manejan las relaciones entre tablas. Aquí te dejo algunos ejemplos comunes de relaciones en bases de datos:

1. ***Relación uno a muchos***

Supongamos que un Author puede tener muchos Books. En el modelo, un autor tendrá una relación con varios libros.

```python
class Author(Base):
    __tablename__ = 'authors'
    id = Column(Integer, primary_key=True)
    name = Column(String)

class Book(Base):
    __tablename__ = 'books'
    id = Column(Integer, primary_key=True)
    title = Column(String)
    author_id = Column(Integer, ForeignKey('authors.id'))

    author = relationship('Author', back_populates='books')

Author.books = relationship('Book', back_populates='author')
```

Documento elaborado por: Alvaro Pérez Niño - Instructor

REGIONAL ANTIOQUIA
CENTRO DE SERVICIOS Y GESTION EMPRESARIAL
TECNOLOGIA EN ANALISIS Y DESARROLLO DE SOFTWARE
Programación Orientada Objetos (Python + Django)

CONCEPTOS DE ORM

*2. **Relación muchos a muchos***

Imagina que tienes un modelo de Student y Course, donde un estudiante puede estar inscrito en varios cursos y un curso puede tener varios estudiantes.

```python
class Student(Base):
    __tablename__ = 'students'
    id = Column(Integer, primary_key=True)
    name = Column(String)

class Course(Base):
    __tablename__ = 'courses'
    id = Column(Integer, primary_key=True)
    title = Column(String)

class Enrollment(Base):
    __tablename__ = 'enrollments'
    student_id = Column(Integer, ForeignKey('students.id'),
primary_key=True)
    course_id = Column(Integer, ForeignKey('courses.id'), primary_key=True)
```

¿Cuándo usar un ORM?

1. **Aplicaciones de tamaño pequeño a mediano**: Los ORM son muy útiles cuando estás trabajando con bases de datos pequeñas o medianas y necesitas un acceso rápido a los datos sin preocuparte mucho por las optimizaciones.
2. **Cuando el rendimiento no es una preocupación crítica**: En algunas situaciones muy específicas donde necesitas un rendimiento extremadamente optimizado, escribir SQL puro puede ser más eficiente. Sin embargo, los ORM son lo suficientemente rápidos para la mayoría de las aplicaciones.
3. **Para mantener el código limpio y fácil de mantener**: Si estás trabajando en un proyecto donde varios desarrolladores estarán involucrados, los ORM ayudan a reducir el código repetitivo y hacer que el sistema sea más escalable.

Documento elaborado por: Alvaro Pérez Niño - Instructor

REGIONAL ANTIOQUIA
CENTRO DE SERVICIOS Y GESTION EMPRESARIAL
TECNOLOGIA EN ANALISIS Y DESARROLLO DE SOFTWARE
Programación Orientada Objetos (Python + Django)

CONCEPTOS DE ORM

Campos de los Modelos ORM

Tipo de Campo	Descripción	Atributos comunes	Ejemplo de uso
CharField	Cadena de texto corta	`max_length`, `unique`, `default`	`nombre = models.CharField(max_length=100)`
TextField	Texto largo	`blank`, `default`	`descripcion = models.TextField(blank=True)`
IntegerField	Número entero	`default`, `unique`	`edad = models.IntegerField(default=0)`
AutoField	Entero auto-incremental (generalmente `id`)	Se usa automáticamente como clave primaria	`id = models.AutoField(primary_key=True)`
BigAutoField	Igual que `AutoField`, pero para valores grandes	`primary_key`	`id = models.BigAutoField(primary_key=True)`
EmailField	Dirección de correo válida	`unique`, `blank`	`email = models.EmailField(unique=True)`
BooleanField	Verdadero/Falso	`default`	`activo = models.BooleanField(default=True)`
DateField	Fecha	`auto_now`, `auto_now_add`, `default`	`fecha_nac = models.DateField(default=date.today)`
DateTimeField	Fecha y hora	`auto_now`, `auto_now_add`	`creado = models.DateTimeField(auto_now_add=True)`
FloatField	Número decimal (coma flotante)	`default`	`precio = models.FloatField(default=0.0)`
DecimalField	Números con precisión decimal	`max_digits`, `decimal_places`, `default`	`precio = models.DecimalField(max_digits=8, decimal_places=2)`
URLField	Campo para URLs	`blank`, `default`	`sitio = models.URLField(blank=True)`
ForeignKey	Clave foránea (relación uno a muchos)	`on_delete`, `related_name`	`categoria = models.ForeignKey(Categoria, on_delete=models.CASCADE)`

Documento elaborado por: Alvaro Pérez Niño - Instructor

REGIONAL ANTIOQUIA
CENTRO DE SERVICIOS Y GESTION EMPRESARIAL
TECNOLOGIA EN ANALISIS Y DESARROLLO DE SOFTWARE
Programación Orientada Objetos (Python + Django)

CONCEPTOS DE ORM

OneToOneField	Relación uno a uno	on_delete	perfil = models.OneToOneField(Perfil, on_delete=models.CASCADE)
ManyToManyField	Relación muchos a muchos	related_name	productos = models.ManyToManyField(Producto)

Atributos de los Campos de los Modelos ORM

Atributo	Descripción
unique=True	Asegura que el valor sea único en la tabla (como en emails, usernames)
default=valor	Establece un valor por defecto si no se proporciona uno
blank=True	Permite que el campo quede vacío en formularios (afecta validación, no DB)
null=True	Permite que el campo quede vacío a nivel de base de datos
auto_now	Guarda automáticamente la fecha/hora cada vez que se actualiza el objeto
auto_now_add	Guarda automáticamente la fecha/hora solo cuando se crea el objeto

Referentes bibliográficos

1. Django Software Foundation. (2025, abril 20). *Django ORM tutorial*. https://docs.djangoproject.com/en/stable/topics/db/models/
2. Johnson, R. (2018, julio 14). Introduction to SQLAlchemy ORM in Python. *TechBlog*. https://techblog.com/sqlalchemy-introduction
3. Lee, J., & Lee, Y. (2019). Exploring the use of Object-Relational Mappers in modern software development. *Journal of Software Engineering*, 27(4), 323-335. https://doi.org/10.1016/j.josweng.2019.02.010
4. SQLAlchemy. (2021, diciembre 15). *SQLAlchemy ORM Tutorial*. https://docs.sqlalchemy.org/en/14/orm/tutorial.html
5. Smith, J. (2020, mayo 22). Understanding Object-Relational Mapping in Python. *Python Insights*. https://www.pythoninsights.com/understanding-orm
6. Beazley, D. M. (2015). *Python Essential Reference*. Addison-Wesley.

Documento elaborado por: Alvaro Pérez Niño - Instructor