ORM

ORM (Об'єктно-реляційне відображення) - це технологія, яка дозволяє взаємодіяти з базами даних, використовуючи об'єкти в програмному коді замість запитів на мові SQL. Кожен об'єкт представляє окремий запис у таблиці бази даних.

Приклад ORM моделі для бази даних може виглядати так:

Припустимо, у вас є таблиця "Користувачі" з колонками "id", "username" і "email".

```
# Приклад ORM моделі для Python з використанням SQLAlchemy

from sqlalchemy import Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    username = Column(String)
    email = Column(String)

    def __repr__(self):
        return f"<User(id={self.id}, username={self.username}, email={self.email})>"
```

У цьому прикладі ми створюємо клас `User`, який успадковується від `Base`, що є базовим класом для всіх моделей в SQLAlchemy. Потім ми визначаємо атрибути класу, які відповідають колонкам у таблиці бази даних.

`__tablename__` вказує назву таблиці у базі даних.

`__repr__` - це метод, який визначає, як об'єкт буде представлений у вигляді рядка при виклику `print`.

Тепер, коли у вас є ORM модель, ви можете використовувати її для створення, зміни, видалення та отримання даних з бази даних, використовуючи об'єкти цього класу.

Приклади операцій:

```
# Створення нового користувача
new_user = User(username='JohnDoe', email='john@example.com')

# Додавання користувача в базу даних
session.add(new_user)
session.commit()

# Отримання користувача
user = session.query(User).filter_by(username='JohnDoe').first()

# Зміна даних
user.email = 'john.doe@example.com'
session.commit()

# Видалення користувача
session.delete(user)
session.commit()
```

Це лише приклад, і кожна бібліотека ORM може мати свої власні особливості і синтаксис. Проте загальна концепція залишається приблизно такою ж.

Основні поняття

Моделі та сутності є ключовими поняттями в ORM (Об'єктно-реляційному відображенні). Вони дозволяють програмістам працювати з базами даних, використовуючи об'єктно-орієнтований підхід, а не традиційні SQL-запити.

1. Модель:
    - **Модель** - це об'єкт, який представляє сутність (таблицю) в базі даних. В програмуванні на високому рівні, модель визначає структуру даних та її взаємовідношення.
    - Наприклад, якщо у вас є таблиця "Користувачі" у базі даних, відповідна модель буде мати атрибути, які відображають колонки цієї таблиці (наприклад, ім'я, електронна пошта тощо).
    - У більш розширеному розумінні, модель може включати методи та функції, які надають додатковий функціонал для роботи з цією сутністю (наприклад, метод для збереження сутності в базі даних).
2. Сутність (Entity):

- **Сутність** - це конкретний об'єкт, який відповідає запису в базі даних. Кожна сутність відображається однією або кількома записами в таблиці.
- Наприклад, якщо у вас є модель "Користувачі" і у вас є два користувачі з іменами "John" і "Jane", кожен з цих користувачів є сутністю.
- Сутності можуть бути використані для отримання, збереження, зміни та видалення даних з бази даних.

3. Відображення відносин:
    - ORM дозволяє виразити взаємозв'язки між сутностями. Наприклад, відношення "багато-до-одного", "багато-до-багатьох" тощо. Це робиться за допомогою відповідних атрибутів та асоціацій між моделями.
    - Наприклад, у вас може бути модель "Замовлення" та модель "Товар", і кожне замовлення може мати відношення "багато-до-багатьох" з товарами, що замовляються.
4. Мови запитів та фільтри:
    - Один із плюсів використання ORM - це можливість використовувати мову програмування для написання запитів до бази даних. ORM надає API для створення та виконання запитів.
    - Наприклад, за допомогою ORM, ви можете використовувати Python для вибору всіх користувачів, які мають електронну пошту "@example.com".

В цілому, розуміння моделей та сутностей в ORM є важливим для ефективної роботи з базами даних. Це дозволяє вам працювати з даними як з об'єктами у вашому програмному коді, що полегшує розробку та управління даними.