# ORM in Flask

## Overview and Example

# Day 1

by Dev Muhammad Ahmad
muhammad.ahmadch16@

# What is an ORM?

An **Object-Relational Mapper (ORM)** is a tool that allows developers to interact with databases using an object-oriented approach, rather than writing raw SQL queries. It abstracts the database layer and enables the manipulation of database records using Python classes and methods.

**In Flask,** the most commonly used ORM is SQLAlchemy, which allows developers to map database tables to Python objects. Flask-SQLAlchemy is the Flask extension that integrates SQLAlchemy into Flask applications, providing a higher-level API to manage database interactions efficiently.

by Dev Muhammad Ahmad

muhammad.ahmadch16@

# Why Use ORM in Flask?

**1 Abstraction:** ORMs simplify database interactions by allowing you to work with Python objects rather than SQL queries.

**2 Maintainability:**By using models and objects, the code becomes more organized and easier to maintain.

**3 Portability:** You can switch between different database backends (e.g., PostgreSQL, MySQL, SQLite) with minimal changes to your code.

**4 Avoids SQL Injections:** Since the ORM generates SQL queries for you, it protects against common security vulnerabilities like SQL injection.

by Dev Muhammad Ahmad
muhammad.ahmadch16@

# Flask-SQLAlchemy Setup and Example

## Install Flask-SQLAlchemy

First, install Flask and Flask-SQLAlchemy using pip:

```
pip install Flask Flask-SQLAlchemy
```

## Setting Up the Flask Application with ORM

Create a new Python file (e.g., app.py) and configure Flask with SQLAlchemy:

by Dev Muhammad Ahmad
muhammad.ahmadch16@

```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

# Initialize Flask app
app = Flask(__name__)

# Configure the database (SQLite in this case)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# Initialize SQLAlchemy
db = SQLAlchemy(app)

# Define a model for the "User" table
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)

    def __repr__(self):
        return f"<User {self.username}>"

# Route to create a new user
@app.route('/create_user/<name>/<email>')
def create_user(name, email):
    new_user = User(username=name, email=email)
    db.session.add(new_user)
    db.session.commit()
    return f"User {name} has been created!"

# Route to fetch all users
@app.route('/users')
def users():
    all_users = User.query.all()
    return {user.id: user.username for user in all_users}

if __name__ == '__main__':
    app.run(debug=True)
```

by Dev Muhammad Ahmad
muhammad.ahmadch16@

# Explanation:

## Database Configuration:

- app.config['SQLALCHEMY_DATABASE_URI']: Configures the database connection (in this example, SQLite is used).
- db = SQLAlchemy(app): Initializes the ORM with the Flask app.

## Defining the Model:

- class User(db.Model): Represents the table User. Each class attribute (id, username, and email) maps to a column in the database table.
- db.Column(): Defines columns in the table. For example, id is the primary key, username is a string with a maximum length of 80, and email must be unique.

by Dev Muhammad Ahmad
muhammad.ahmadch16@

## Creating New Users

- The route /create_user/<name>/<email> allows you to create new users by passing a username and email as URL parameters.
- db.session.add(new_user) adds the new user to the session, and db.session.commit() saves the changes to the database.

## Fetching Users:

The /users route retrieves all users from the database using User.query.all() and returns them in a dictionary format.

by Dev Muhammad Ahmad
muhammad.ahmadch16@

Thanks For Reading The post.

# If You Get Information from the Post Like, Comment, and Follow

by Dev Muhammad Ahmad
muhammad.ahmadch16@