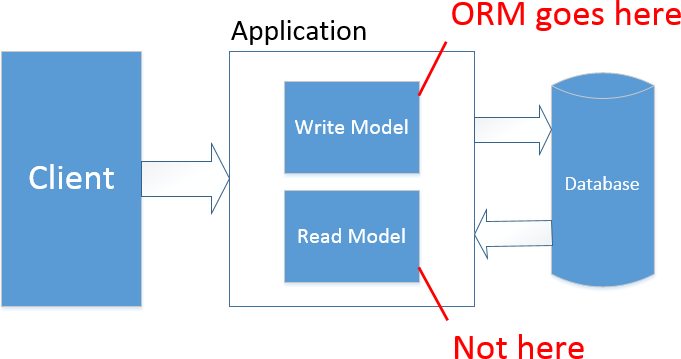


# План занятия

1. ORM
2. SQLAlchemy
3. CRUD
4. One-to-Many
5. Many-to-Many
6. Join
7. Order by
8. Select
9. Aggregation functions
10. Group By
11. Filter By Part of a name
12. In
13. Not in
14. Is null
15. Is not null
16. Or
17. And

**ORM - Object-relational mapping**

Технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных



# Flask SQLAlchemy

Технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных

**from** flask **import** Flask

**from** flask\_sqlalchemy **import** SQLAlchemy app = Flask**(** name **)**

app.config**[**'SQLALCHEMY\_DATABASE\_URI'**]** = 'sqlite:////tmp/test.db’ db = SQLAlchemy**(**app**)**

# Model

В моделях мы описываем структуру нашей таблицы.

**class** User**(**db.Model**):**

id = db.Column**(**db.Integer**,** primary\_key=**True)**

username = db.Column**(**db.String**(**80**),** unique=**True,** nullable=**False)** email = db.Column**(**db.String**(**120**),** unique=**True,** nullable=**False)**

Сессия устанавливает все запросы с базой данных и представляет собой «зону ожидания» для всех объектов, которые вы загрузили или связали с ними в течение их жизненного цикла. Он предоставляет точку входа для получения объекта Query, который отправляет запросы в базу данных с использованием текущего сеанса подключения к базе данных, заполняя результирующие строки в объекты, которые затем хранятся в сеансе, внутри структуры, называемой Identity Map - которая поддерживает уникальные копии каждого объекта, где «уникальный» означает «только один объект с определенным первичным ключом».

>>> **from** yourapplication **import** User

>>> admin = User**(**username='admin’**,** email='admin@example.com'**)**

>>> db.session.add**(**admin**)**

>>> db.session.add**(**guest**)**

>>> db.session.commit**()**

>>> User.query.all**()**

[<User u'admin'>, <User u'guest’>]

>>> User.query.filter\_by**(**username='admin'**)**.first**()** <User u'admin'>

>>> **from** yourapp **import** User

>>> me = User**(**'admin'**,** ['admin@example.com](mailto:%27admin@example.com)’**)**

>>> db.session.add**(**me**)**

>>> db.session.commit**()**

>>> me.id 1

>>> db.session.delete**(**me**)**

>>> db.session.commit**()**

### Get Records

user = UserTable.query.order\_by(UserTable.id).all() user = UserTable.query.all()

user = UserTable.query.first()

### Update Records

row = UserTable.query.filter\_by(id=id).first() row.address = address

# CRUD - Create, Read, Update, Delete

Акроним, обозначающий четыре базовые функции, используемые при работе с хранилищами данных:

***создание; чтение; редактирование; удаление.***

|  |  |  |
| --- | --- | --- |
| Операция | Оператор в языке [SQL](https://ru.wikipedia.org/wiki/SQL) | Операция в протоколе [HTTP](https://ru.wikipedia.org/wiki/HTTP) |
| Создание (create) | [INSERT](https://ru.wikipedia.org/wiki/Insert_(SQL)) | POST |
| Чтение (read) | [SELECT](https://ru.wikipedia.org/wiki/Select_(SQL)) | GET |
| Редактирование (update) | [UPDATE](https://ru.wikipedia.org/wiki/Update_(SQL)) | PUT или PATCH |
| Удаление (delete) | [DELETE](https://ru.wikipedia.org/wiki/Delete_(SQL)) | DELETE |

class Person(db.Model):

id = db.Column(db.Integer, primary\_key=True) name = db.Column(db.String(50), nullable=False)

addresses = db.relationship('Address', backref='person', lazy=True)

class Address(db.Model):

id = db.Column(db.Integer, primary\_key=True) email = db.Column(db.String(120), nullable=False)

person\_id = db.Column(db.Integer, db.ForeignKey('person.id'), nullable=False)

tags = db.Table('tags',

db.Column('tag\_id', db.Integer, db.ForeignKey('tag.id'), primary\_key=True),

db.Column('page\_id', db.Integer, db.ForeignKey('page.id'), primary\_key=True)

)

class Page(db.Model):

id = db.Column(db.Integer, primary\_key=True)

tags = db.relationship('Tag', secondary=tags, lazy='subquery', backref=db.backref('pages', lazy=True))

class Tag(db.Model):

id = db.Column(db.Integer, primary\_key=True)

Table1.query.join(Table2, Table1.id == Table2.table1\_id) db.session.query(Table1).join(Table2, Table1.id == Table2.table1\_id)

## Order by

Table1.query.order\_by(Table1.value) db.session.query(Table1).order\_by(Table1.value)

## Order by desc

Table1.query.order\_by(Table1.value.desc()) db.session.query(Table1).order\_by(Table1.value.desc())

Table1.query.with\_entities(Table1.value1, Table1.value2) db.session.query(Table1.value1, Table1.value2)

## Aggregation functions

db.session.query(db.func.count(Table1.id)) db.session.query(db.func.sum(Table1.id)) db.session.query(db.func.avg(Table1.id))

…

Table1.query.with\_entities(db.func.count(Table1.id)) Table1.query.with\_entities(db.func.sum(Table1.id)) Table1.query.with\_entities(db.func.avg(Table1.id))

…

Table1.query.with\_entities(db.func.sum(Table1.value)).group\_by(Table1.value) db.session.query(db.func.sum(Table1.value).group\_by(Table1.value)

## Filter By part of a name (like)

Table1.query.filter(Table1.value.like('%aaaa%')) db.session.query(Table1).filter(Table1.value.like('%aaaa%'))

Table1.query.filter(Table1.value.in\_(['value1', 'value2'])) db.session.query(Table1).filter(Table1.value.in\_(['value1', 'value2']))

## NOT IN

Table1.query.filter(~Table1.value.in\_(['value1', 'value2'])) db.session.query(Table1).filter(~Table1.value.in\_(['value1', 'value2']))

Table1.query.filter(Table1.value == None) db.session.query(Table1).filter(Table1.value == None)

## IS NOT NULL

Table1.query.filter(Table1.value != None) db.session.query(Table1).filter(Table1.value != None)

from sqlalchemy import or\_

Table1.query.filter(or\_(Table1.value == ‘value', Table1.value2 == 'value')) db.session.query(Table1).filter(or\_(Table1.value == ‘value', Table1.value2 == 'value'))

## AND

from sqlalchemy import and\_

Table1.query.filter(and\_(Table1.value == ‘value', Table1.value2 == 'value')) db.session.query(Table1).filter(and\_(Table1.value == ‘value', Table1.value2 == 'value'))