

Тема 9.4: SELECT-ЗАПРОСЫ, ВЫБОРКИ ИЗ ОДНОЙ ТАБЛИЦЫ

Цель занятия:

Ознакомиться с основами проектирования БД.



План занятия:

1. Основы SQLAlchemy
2. SELECT-запросы
3. Арифметические операции
4. Оператор WHERE и фильтрация по условиям:
 - a. сравнения
 - b. IN
 - c. BETWEEN
 - d. LIKE
5. Сортировка при помощи ORDER BY
6. Оператор LIMIT
7. INSERT/UPDATE/DELETE запросы



База данных Pagila:

База данных Pagila:

На лекции мы будем практиковаться на базе данных Pagila.

Документы для скачивания:

База данных:

<https://www.postgresqltutorial.com/postgresql-getting-started/postgresql-sample-database/>

Схема базы данных:

https://dataedo.com/samples/html/Pagila/doc/Pagila_10/modules/Pagila_database_diagram_103/module.html

Основы SQLAlchemy

Документация:

<https://el1sha-git.github.io/sqlalchemy-rusdocs/tutorial/>

<https://docs.sqlalchemy.org/en/20/>

SQLAlchemy

SQLAlchemy – это программная библиотека на языке Python для работы с реляционными СУБД с применением технологии ORM.

Для подключения к базе данных нужно создать экземпляр класса **Engine** с помощью **create_engine()**, которая в качестве аргумента принимает адрес в виде:

```
dialect+driver://username:password@host:port/database
```

Потом инициализировать подключение к базе при помощи метода **connect()**.
Например:

```
db = 'postgresql://postgres:admin@localhost:5432/postgres'  
engine = sqlalchemy.create_engine(db)  
connection = engine.connect()
```

Для корректной работы SQLAlchemy необходима также установка **psycopg2**

Официально поддерживаются все наиболее популярные системы баз данных: **PostgreSQL, MySQL, MariaDB, SQLite, Oracle и Microsoft SQL Server**. Кроме того, есть сторонние пакеты для SQLAlchemy, которые добавляют поддержку для менее распространенных систем, типа CockroachDB, Firebird, IBM DB2 и т.д.

Запросы через SQLAlchemy

Для выполнения SQL запросов к подключенной БД используется метод `execute()`. Для получения результатов можно использовать один из `fetch` методов, которые возвращают список объектов RowProxy (значений строк таблицы).

`fetchone()` – извлечение 1 строки,

`fetchall()` – извлечение всех строк,

`fetchmany(n)` – извлечение заданного количества строк.

Например:

```
connection.execute("SELECT * FROM database;").fetchmany(10)
```

Типы запросов в SQL

- **DDL** (Data Definition Language) – CREATE, ALTER, DROP
- **DML** (Data Manipulation Language) – SELECT, INSERT, UPDATE, DELETE
- **TCL** (Transaction Control Language) – COMMIT, ROLLBACK, SAVEPOINT
- **DCL** (Data Control Language) – GRANT, REVOKE, DENY

SELECT-запросы

SELECT-запросы предназначены для отбора необходимых строк или столбцов из одной или нескольких таблиц.

Общий вид структуры select-запроса:

```
SELECT [DISTINCT | ALL] table_columns  
FROM table
```

Обязательные элементы запроса

```
[WHERE condition]
```

```
[GROUP BY table_columns]
```

```
[HAVING condition]
```

```
[ORDER BY table_columns [ASC | DESC]]
```

```
[LIMIT number]
```

Необязательные элементы запроса
в строгой последовательности

SELECT-запросы, DISTINCT

```
SELECT * FROM table;
```

* – выбор всех столбцов таблицы;
table – имя нужной таблицы.

Через запятую можно перечислить имена необходимых столбцов:

```
SELECT title, release_year FROM film;
```

Оператор **DISTINCT** позволяет выбирать только уникальные значения из базы данных — он отсеивает дубли:

```
SELECT DISTINCT rating FROM film;
```

Арифметические операторы

Можно использовать различные арифметические операторы для данных, хранящихся в таблицах:

- **+** – сложение;
- **-** – вычитание;
- **/** – деление;
- ***** – умножение;
- **%** – взятие остатка от деления.

Оператор **WHERE** нужен для фильтрации таблиц по условиям.

В качестве условий можно использовать:

- сравнения (**=**, **>**, **<**, **>=**, **<=**, **!=**);
- оператор **IN**;
- оператор **BETWEEN**;
- оператор **LIKE**.

С условиями можно применять логические операторы **and**, **or** и **not**:

- Оператор **AND** отображает запись, если оба операнда истинны;
- Оператор **OR** отображает запись, если хотя бы один операнд истинен;
- Оператор **NOT** инвертирует исходный операнд.

Примеры запросов с отбором по сравнению

```
SELECT title, release_year FROM film  
WHERE release_year >= 2000;
```

```
SELECT first_name, last_name, active FROM staff  
WHERE NOT active = true;
```

```
SELECT title, rental_rate, replacement_cost FROM film  
WHERE rental_rate <= 0.99 AND replacement_cost <= 9.99;
```

Примеры использования IN

Оператор **IN** отбирает строки, в которых есть перечисленные значения. Если значений много — они перечисляются через запятую. Например:

```
SELECT title, description, rating FROM film  
WHERE rating IN ('R', 'NC-17');
```

```
SELECT title, description, rating FROM film  
WHERE rating NOT IN ('G', 'PG');
```

Примеры использования BETWEEN

Оператор **BETWEEN** позволяет проверить, находится ли выражение в диапазоне значений. Крайние значения включаются в диапазон.

```
SELECT title, rental_rate FROM film  
WHERE rental_rate BETWEEN 0.99 AND 1.99;
```



```
SELECT title, rental_rate FROM film  
WHERE rental_rate NOT BETWEEN 0.99 AND 1.99;
```



Примеры использования LIKE

Оператор `LIKE` позволяет проверить, находится ли в значении заданный текстовый шаблон.

В шаблонах кроме обычных символов могут использоваться два служебных:

- `%` – ноль, один или несколько любых символов,
- `_` – один любой символ.

```
SELECT title, description FROM film
WHERE description LIKE '%%Scientist%%';
```

Важно. В python знак `%` зарезервирован, поэтому необходимо использовать `%%`.

Сортировка при помощи ORDER BY

ORDER BY используется для сортировки таблицы по указанным столбцам.

```
SELECT title, rental_rate FROM film  
ORDER BY rental_rate;
```

```
SELECT title, rental_rate FROM film  
ORDER BY rental_rate DESC;
```

```
SELECT title, length, rental_rate FROM film  
ORDER BY length DESC, rental_rate;
```

Оператор LIMIT

LIMIT используется, чтобы ограничивать количество возвращаемых записей из одной или нескольких таблиц.

```
SELECT title, length, rental_rate FROM film
WHERE rental_rate > 2.99
ORDER BY length DESC, rental_rate
LIMIT 15;
```

INSERT/UPDATE/DELETE запросы

INSERT INTO

Инструкция `INSERT INTO` используется для вставки новых записей в таблицу.

Если необходимо вставить значения только для части полей, то используется следующий синтаксис:

```
INSERT INTO rental(rental_date, inventory_id, customer_id, staff_id)
VALUES(NOW(), 1, 3, 2);
```

Если необходимо вставить значения для всех полей, то нужно убедиться, что они все перечислены и их порядок соответствуют порядку полей:

```
INSERT INTO inventory
VALUES(999, 999, 999);
```

UPDATE

UPDATE используется для изменения существующих записей в одной или нескольких таблицах. После оператора **SET** указывается изменяемый столбец, при необходимости можно добавить изменения по условиям:

```
UPDATE rental  
  SET return_date = NOW()  
  WHERE rental id = 16053;
```

Можно изменять одновременно несколько полей. Без указания **WHERE** будут изменены все записи в столбце.

DELETE

DELETE используется для удаления существующих записей в таблице.

```
DELETE FROM rental  
WHERE rental_id = 16050;
```

Без указания WHERE будут удалены все записи в столбце.