Язык Python. Часть 2

Лекция 17

На чем остановились

- Есть SQL как общая технология
- Есть конкретная реализация - библиотека sqlite
- Начинаем с создания таблицы
- Для создания нужно задать схему

На чем остановились

- Схема - это набор полей
- У каждого поля - имя и тип данных
- Есть специальный запрос на создание таблицы
- CREATE TABLE

Персистентность

- Все, что создается в SQL - сохраняется "само собой"
- В том числе база данных
- База в целом в sqlite хранится в файле
- При открытии таблицы sqlite ищет файл с именем базы

Персистентность

- Не нашел - создает пустую
- Нашел - использует
- То, что создано в одном сеансе - сохрается при перезапуске
- Но можно удалить файл - тем самым удалив всю базу

Запуск на существующей базе

- Файл есть, база есть, а в ней таблица
- Повторное создание - ошибка
- Можно удалить файл 'db' и начать сначала
- Можно уточнить запрос на создание
- Можно удалить таблицу отдельным запросом

Запуск на существующей базе

- В запрос на создание можно добавить 'IF NOT EXISTS'
- Если таблица существует - останется нетронутой
- Запрос на удаление таблицы - 'DROP TABLE movie'
- Тут обратная проблема - не сработает, если таблицы нет

Запуск на существующей базе

- Можно уточнить запрос
- Добавить 'IF EXISTS'
- 'DROP TABLE movie IF EXISTS'
- Важно не удалить ничего полезного

Добавляем данные

- Запрос INSERT
- Пример: INSERT INTO movies VALUES ('Some title', 1989, '+++');
- После INSERT INTO - имя таблицы
- После VALUES - перечень значений в скобках

Добавляем данные

- Порядок полей соответствует определению таблицы
- Можно указать не все значения
- Неуказанные будут иметь значения по умолчанию
- В SQL оно называется NULL

Извлекаем данные

- Хотим увидеть результаты добавления
- Для этого есть свой запрос: SELECT
- Основной запрос SQL
- Простейшая форма: SELECT * from movies;

Пример

```
1 import sqlite3
2
3 with sqlite3.connect('db') as conn:
4     conn.execute("DROP TABLE IF EXISTS movies")
5     conn.execute("CREATE TABLE movies(title text, year int,
6     print(conn.execute("SELECT * from movies"))
```

Осознаем происходящее

- Интуитивно ожидаем чего-то пустого
- Пустого списка, например
- Получаем объект класса Cursor
- В общем случае он позволяет постепенно забирать результат
- Но можно и сразу все

Пример

```
1 import sqlite3
2
3 with sqlite3.connect('db') as conn:
4     conn.execute("DROP TABLE IF EXISTS movies")
5     conn.execute("CREATE TABLE movies(title text, year int,
6     print(conn.execute("SELECT * from movies").fetchall())
```

Пример

```
1 import sqlite3
2
3 with sqlite3.connect('db') as conn:
4     conn.execute("DROP TABLE IF EXISTS movies")
5     conn.execute("CREATE TABLE movies(title text, year int,
6     conn.execute("INSERT INTO movies VALUES ('some title', 1
7     print(conn.execute("SELECT * from movies").fetchall())
```

Уточняем вывод

- Звездочка означает все поля
- Иногда хочется не все
- Можно перечислить по именам
- Можно использовать выражения

Пример

```
1 import sqlite3
2
3 with sqlite3.connect('db') as conn:
4     conn.execute("DROP TABLE IF EXISTS movies")
5     conn.execute("CREATE TABLE movies(title text, year int,
6     conn.execute("INSERT INTO movies VALUES ('some title', 1
7     print(conn.execute("SELECT title, score from movies").fe
```

Пример

```
1 import sqlite3
2
3 with sqlite3.connect('db') as conn:
4     conn.execute("DROP TABLE IF EXISTS movies")
5     conn.execute("CREATE TABLE movies(title text, year int,
6     conn.execute("INSERT INTO movies VALUES ('some title', 1
7     print(conn.execute("SELECT title, 2023 - year  from movi
```

Выбор строк

- Таблица может быть огромной
- Мы не хотим выводить все данные
- Можно профильтровать в скрипте
- Но эффективнее - прямо в запросе

Выбор строк

- После FROM и имени таблицы указываем WHERE
- И дальше - условие
- Свой синтаксис
- Как правило - интуитивно понятный

Пример

```
1 import sqlite3
2
3 with sqlite3.connect('db') as conn:
4     conn.execute("DROP TABLE IF EXISTS movies")
5     conn.execute("CREATE TABLE movies(title text, year int,
6     conn.execute("INSERT INTO movies VALUES ('some title', 1
7     conn.execute("INSERT INTO movies VALUES ('another title'
8     conn.execute("INSERT INTO movies VALUES ('yet another',
9     print(conn.execute("SELECT title, 2023 - year  from movi
```

Пример

```
1 import sqlite3
2
3 with sqlite3.connect('db') as conn:
4     conn.execute("DROP TABLE IF EXISTS movies")
5     conn.execute("CREATE TABLE movies(title text, year int,
6     conn.execute("INSERT INTO movies VALUES ('some title', 1
7     conn.execute("INSERT INTO movies VALUES ('another title'
8     conn.execute("INSERT INTO movies VALUES ('yet another',
9     print(conn.execute("SELECT title, 2023 - year  from movi
```

Удаление строк

- Запрос DELETE
- Можно удалить все записи
  - DELETE FROM table;
- Это не DROP !

Удаление строк

- DROP - удаляет таблицу и структуру
- После DROP не работает SELECT
  - На удаленной таблице
- И INSERT - тоже не будет
- Надо создавать таблицу заново

Удаление строк

- DELETE FROM - удаляет записи
- Таблица остается со структурой
- Можно делать и SELECT, и INSERT
- И не обязательно удалять все

Удаление строк по критерию

- DELETE FROM table WHERE a = 5;
- Заметим: равенство в условиях - одинарное
- SQL - не Python, свои правила
- В SQL присваивания и сравнения всегда в разных контекстах

Пример

```
 1 import sqlite3
 2
 3 with sqlite3.connect('db') as conn:
 4     conn.execute("DROP TABLE IF EXISTS movies")
 5     conn.execute("CREATE TABLE movies(title text, year int,
 6     conn.execute("INSERT INTO movies VALUES ('some title',
 7     conn.execute("INSERT INTO movies VALUES ('another title
 8     conn.execute("INSERT INTO movies VALUES ('yet another',
 9     print(conn.execute("SELECT *  from movies").fetchall())
10     conn.execute("DELETE FROM movies")
11     print(conn.execute("SELECT * from movies WHERE year < 2
```

Пример

```
 1 import sqlite3
 2
 3 with sqlite3.connect('db') as conn:
 4     conn.execute("DROP TABLE IF EXISTS movies")
 5     conn.execute("CREATE TABLE movies(title text, year int,
 6     conn.execute("INSERT INTO movies VALUES ('some title',
 7     conn.execute("INSERT INTO movies VALUES ('another title
 8     conn.execute("INSERT INTO movies VALUES ('yet another',
 9     print(conn.execute("SELECT *  from movies").fetchall())
10     conn.execute("DELETE FROM movies WHERE year < 2000"")
11     print(conn.execute("SELECT * from movies").fetchall())
```

Трехзначная логика

- NULL в значениях отражается на логических операциях
- Любое сравнение с NULL дает не TRUE и не FALSE
- Любое сравнение с NULL дает NULL
- NULL - третий вариант логического значения

Трехзначная логика

- Делаем SELECT по условию name = 'vasya'
- Делаем SELECT по условию name != 'vasya'
- Считаем количество записей в обоих ответах, складываем
- Можем получить меньше, чем всего записей

Трехзначная логика

- Если в name NULL, то NULL = 'vasya' даст NULL
- И NULL != 'vasya' даст NULL
- А NULL - не истина
- Строки, в которых NULL в name и дадут разницу

Трехзначная логика

- Попробуем найти строки, в которых в name NULL
- Наивный вариант: SELECT ... WHERE name = NULL
- Ничего не найдется
- WHERE name != NULL - тоже ничего

Трехзначная логика

- Любое сравнение с NULL дает NULL
- Хоть на равенство, хоть на неравенство
- Даже NULL = NULL даст NULL
- И NULL != NULL