Язык Python. Часть 2

Лекция 18

CRUD

- CRUD - Create, Request, Update, Delete
- Умеем все, кроме Update
- Есть и такой запрос - UPDATE
- Указываем таблицу и условие

Изменение

- Условие определяет обновляемые строки
- Пропустить нельзя (в отличие от DELETE)
- Редко требуется обновить все
- Если нужно - можно указать true как условие

Изменение

- После условия - ключевое слово SET
- И присваивание
- В левой части имя поля таблицы
- В правой - выражение

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     conn.execute("UPDATE movies SET score = 'cool' WHERE ti
10     print(conn.execute("SELECT *  from movies").fetchall())
```

Изменение

- Присваиваний может быть несколько в запросе
- Через запятую
- В правой части могут быть поля той же записи
- И/или произвольные выражения

Реляционная модель

- Но кажется, что не все данные имеют табличную структуру
- Например, иерархия начальников и подчиненных
- А бывает так, что есть разные таблицы
- Но как-то связанные между собой

Реляционная модель

- Например легко представить себе таблицу студентов
- Или таблицу курсов
- Или в смысле курсов вообще
- Или в смысле конкретных прочтений
- Или того и другого

Реляционная модель

- Таблица факультетов, таблица преподавателей
- И можно представить себе таблицу записей студентов на курсы
- В ней и данные о студенте
- И данные о курсе

Реляционная модель

- Проблема: а как это все хранить в таблицах ?
- Заводить на каждое прочтение каждого курса свою таблицу ?
- И данные о студенте хранить в таблице каждого посещаемого им курса ?
- А если данные студента поменялись ?

Реляционная модель

- Все можно свести к базовым таблицам
- Не повторяя данные
- Связывая разные таблицы в момент запроса
- И иерархические данные тоже можно так представить

Отношение

- Отношение: набор свойств объекта реального мира
- Упорядоченный набор значений
- Синоним понятия "таблица"
- Иногда выделяется одно или несколько свойств

Отношение

- Бывает так, что элемент отношения однозначно определяется по свойству
- Например, для полиции человек определяется паспортом
- А автомобиль - номерным знаком
- По крайней мере, в первом приближении

Отношение

- Иногда нужно несколько свойств
- Например, бронирование в отеле
- Определяется номером и датой
- Это называется первичным ключом

Отношение

- Не всегда легко сказать, является ли набор полей первичным ключем
- Например, ФИО человека
- В таблице налогоплательщиков - точно нет
- В таблице родителей школьников данного класса - почти наверняка да

Отношение

- Иногда используется искуственный ключ
- Заводится отдельное поле
- И каждой новой строке таблицы присваивается уникальное значение
- Часто такая возможность встроена в СУБД

Соединение отношений

- Пусть есть таблица с данными об объектах
- Есть другая таблица с другими данными о тех же объектах
- Например, есть таблица с записями о сотрудниках
- Первичный ключ - уникальный идентификатор

Соединение отношений

- В таблице - общие данные о сотруднике
- Дата рождения, дата приема на работу
- Каждому сотруднику оформляется полис ДМС
- И эти данные - в отдельной таблице

Соединение отношений

- Ее первичный ключ - тоже идентификатор сотрудника
- И даже не факт, что один и тот же id - всегда в обоих таблицах есть
- Бывают сотрудники, на которых полис еще не формили
- А кто-то уволился, но данные о полисе остались

Соединение отношений

- Давайте возьмем строчку из первой таблицы
- И найдем соответствующую во второй
- С тем же идентификатором сотрудника
- И соединим две строки в одну

join

- SELECT может соединять таблицы
- Ключевое слово JOIN
- После имени таблицы
- Дальше - выражение, по которому соединяемся

Именование полей

- Изменяются правила именования полей
- И в выборе столбцов результата
- И в условиях (WHERE)
- И в критерии JOIN

join

- Полное имя поля состоит из имени таблицы
- И имени поля внутри таблицы
- Разделенных точкой
- Имя таблицы можно опустить
- Если имя поля не повторяется в других таблицах

Пример

```
1 conn.execute("CREATE TABLE info(id char(10), name text, posi
2 conn.execute("CREATE TABLE insurance(id char(10), police_id
3
4 conn.execute("INSERT INTO info VALUES ('12345', 'Ivan Petrov
5 conn.execute("INSERT INTO info VALUES ('23456', 'Nikolay Sid
6
7 conn.execute("INSERT INTO insurance VALUES ('12345', 'aaa-21
8 conn.execute("INSERT INTO insurance VALUES ('12345', 'aab-21
```

Пример

```
1 print(conn.execute("SELECT *  from info").fetchall())
2 print(conn.execute("SELECT *  from insurance").fetchall())
3
4 print(conn.execute("SELECT * from info JOIN insurance ON inf
5 print(conn.execute("SELECT info.id, name, position, level fr
6                     " JOIN insurance ON info.id = insurance.i
```

Пример

```
1 print(conn.execute("SELECT info.id, name, position, level fr
2                     " JOIN insurance ON info.id = insurance.i
3                     " WHERE level <= 2;").fetchall())
```

Левый и правый JOIN

- Уже встречалось в Pandas (с merge)
- По умолчанию в результат JOIN входит то, что сцепилось
- Для кого не нашлось пары - не входят
- Это называется INNER JOIN

Левый и правый JOIN

- Можно включить в результат не нашедшие пары записи из левой таблицы
- С пустыми значениями из столбцов правой
- Это называется LEFT JOIN
- Или наоборот - это называется RIGHT JOIN
- Одновременно - OUTER JOIN
- sqlite не поддерживает RIGHT и OUTER

Пример

```
print(conn.execute("SELECT info.id, name, position, level fr
                   " INNER JOIN insurance ON info.id = i
print(conn.execute("SELECT info.id, name, position, level fr
                   " LEFT JOIN insurance ON info.id = in
```

Отношение one-to-many

- Есть таблица пользователей интернет-магазина
- id пользователя, имя, дата рождения
- Есть таблица сделанных заказов
- Дата/время заказа, сумма заказа, id заказа

Отношение one-to-many

- Каждый заказ делает один пользователь
- Будем хранить его id
- Каждый пользователь может сделать много заказов
- Хранить заказы с пользователем не надо

Отношение one-to-many

- Если нужны заказы, сделанные пользователем
  - Делаем SELECT по JOIN
  - Соединяем таблицу заказов с таблицей пользователей
  - По id пользователя

Отношение one-to-many

- Добавим условие на id пользователя
- Можно добавить еще условия
- Например, на время заказа
- Или на стоимость

Отношение one-to-many

- Можно использовать LEFT JOIN
- В WHERE проверить поле правой таблицы на NULL
- Например, id заказа
- Получить список пользователей, не делавших заказы

Пример: создаем таблицы

```
1 import sqlite3
2
3 with sqlite3.connect('db') as conn:
4     conn.execute('DROP TABLE IF EXISTS users')
5     conn.execute('DROP TABLE IF EXISTS orders')
6
7     conn.execute("CREATE TABLE users(user_id char(10), name
8     conn.execute("CREATE TABLE orders(order_id char(10), "
9                  "user_id char(10), value int)")
```

Пример(продолжение): заполняем

```
conn.execute("INSERT INTO users VALUES ('12345', 'Ivan Petro
conn.execute("INSERT INTO users VALUES ('23456', 'Nikolay Si
conn.execute("INSERT INTO users VALUES ('34567', 'Vasily Iva

conn.execute("INSERT INTO orders VALUES ('111111', '12345',
conn.execute("INSERT INTO orders VALUES ('111112', '12345',
conn.execute("INSERT INTO orders VALUES ('111113', '34567',
conn.execute("INSERT INTO orders VALUES ('111114', '12345',
conn.execute("INSERT INTO orders VALUES ('111113', '34567',
```

Пример(продолжение): запрос с JOIN

```
result = conn.execute("SELECT * from users "
                      "JOIN orders ON users.user_id = orders
for record in result:
    print(record)
```

Пример(продолжение): запрос с LEFT JOIN

```
result = conn.execute("SELECT name from users "
                      "LEFT JOIN orders ON users.user_id = o
                      "WHERE orders.user_id IS NULL")
for record in result:
    print(record)
```

Иерархия

- Частный случай one-to-many
- JOIN таблицы с собой
- Пример: таблица сотрудников
- Ключ: id сотрудника

Иерархия

- Отдельное поле - id начальника (manager_id)
- Можно сделать JOIN по равенству полей id и manager_id
- Получим набор пар (работник, его непосредственный начальник)
- Дальше - вопрос критериев WHERE и выбранных полей

Иерархия

- Можно по конкретному значению manager_id
- Получим список непосредственных подчиненных менеджера
- Можно по конретному значению id
- Получим менеджера для конкретного сотрудника

Цепочки из JOIN

- Не обязательно ограничиваться одним JOIN-ом
- Таблицу студентов можно связать с таблицей групп
- А таблицу групп связать с таблицей факультетов
- А еще с таблицей факультетов можно связать таблицу преподавателей

Many-to-many

- Отношения one-to-many удобно представлять через поле-ссылку
- В many-части хранить id элемента one-части
- Но бывают отношения many-to-many
- Например, запись студентов на курсы

Many-to-many

- Заводим отдельную табличку
- Одно поле - ключ одной таблицы
- Другое - ключ другой
- Могут быть другие поля
- Например, итоговая оценка студента

Many-to-many

- Пусть нужно найти списокок пар
- Студентов и курсов
- По критерию высокого балла за курсы
- Решение: двойной JOIN и WHERE

Группировка и агрегация

- Пока мы обрабатывали данные по записям
- Фильтровали на уровне записи
- Преобразовывали на уровне записи
- Иногда нужен другой тип обработки

Группировка и агрегация

- Разбить записи на группы
- По значению поля или выражения
- Например записи о заказах на группы по пользователю
- По автору заказа

Группировка и агрегация

- И внутри группы посчитать агрегирующую функцию
- Простейший вариант - счетчик записей
- Другие: среднее по полю, сумма поля по записям из группы
- Специальный синтаксис: GROUP BY

Пример(продолжение): запрос с LEFT JOIN

```
result = conn.execute("SELECT user_id, COUNT(user_id), SUM(v
                      "FROM orders GROUP BY user_id "
                      )
for record in result:
    print(record)
```