* [ОБЗОР КУРСА](https://lyceum.yandex.ru/courses/165/groups/1257)

[Урок QT SQL 2](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1231)

**Введение в БД, работа с SQL-таблицами и отображение данных в PyQT. Часть 2**

1. [SQL.Продолжение](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1231/materials/2572#1)
2. [Модификация из Python](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1231/materials/2572#2)

**Аннотация**

*В уроке речь пойдёт про изменение и удаление данных в базах данных с помощью SQL и Python.*

**SQL. Продолжение**

В прошлом уроке мы познакомились только с одной частью работы с SQL — получение информации с помощью специального запроса SELECT. Но перед тем, как получать информацию из базы, необходимо ее туда поместить.

Для этого существует команда INSERT. Ее синтаксис в общем случае выглядит так:

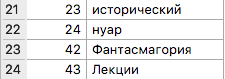
INSERT INTO имя\_таблицы(названия\_полей\*) VALUES(значения)

Названия полей могут не указываться, тогда значения по умолчанию подставятся в поля по порядку. Давайте рассмотрим несколько примеров:

INSERT INTO genres(id,title) VALUES(42,'Фантасмагория')

INSERT INTO genres(title) VALUES('Лекции')

И теперь, когда мы захотим вывести таблицу жанров, мы увидим, что там появились новые значения.



Но откуда взялось значение id, равное 43? Мы же его нигде не указывали? Дело в том, что в нашей таблице поле id является **автоинкрементным**, то есть при создании нового элемента берется максимальный из уже созданных индексов, увеличивается на единицу и присваивается новому элементу.

Часто в таблицу необходимо вставить не одно значение, а несколько. Для это не нужно вызывать команду INSERT несколько раз, а достаточно через запятую перечислить все значения, которые необходимо добавить. Например, так:

INSERT INTO genres VALUES (45, 'Научные'), (46, 'Сказки')

Часто случается, что информация в базе нуждается в изменении. Для этого используется команда UPDATE.

UPDATE имя\_таблицы

SET название\_колонки = новое\_значение

WHERE условие

Предположим, что мы посмотрели фильм «Аватар», и теперь хотим указать его продолжительность. Разумеется, можно сохранить информацию, удалить запись и создать новую. Можно, но это крайне неудобно. Так что обновим значение поля duration для фильма «Аватар» .

UPDATE films

SET duration = '162'

WHERE title = 'Аватар'

https://yastatic.net/s3/lyceum/content/images/second-year/pyqt-7/qt-7-2.png

Но бывает и так, что информация уже совершенно точно не нужна. Тогда ее необходимо удалить из БД, чтобы она не занимала лишнее пространство в памяти.

Например, чтобы удалить все фильмы, выпущенные до 1985 года, необходимо написать следующий запрос:

DELETE from films

where year < 1985

**Модификация из Python**

В прошлом уроке мы научились получать информацию из БД и отображать ее в таблице. Но достаточно часто пользователям нужно не только просматривать, но и модифицировать и удалять информацию. Напишем программу, которая позволит получить фильм по его идентификатору, изменить его поля и сохранить. Создадим интерфейс с помощью QtDesigner.

import sqlite3

import sys

from PyQt5.QtWidgets import QApplication, QMainWindow, QTableWidgetItem

from ui\_3 import Ui\_Form

class MyWidget(QMainWindow, Ui\_Form):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setupUi(self)

self.con = sqlite3.connect("films.db")

self.pushButton.clicked.connect(self.update\_result)

self.tableWidget.itemChanged.connect(self.item\_changed)

self.pushButton\_2.clicked.connect(self.save\_results)

self.modified = {}

self.titles = None

def update\_result(self):

cur = self.con.cursor()

# Получили результат запроса, который ввели в текстовое поле

result = cur.execute("Select \* from films WHERE id=?",

(self.spinBox.text(),)).fetchall()

# Заполнили размеры таблицы

self.tableWidget.setRowCount(len(result))

self.tableWidget.setColumnCount(len(result[0]))

self.titles = [description[0] for description in cur.description]

# Заполнили таблицу полученными элементами

for i, elem in enumerate(result):

for j, val in enumerate(elem):

self.tableWidget.setItem(i, j, QTableWidgetItem(str(val)))

self.modified = {}

def item\_changed(self, item):

# Если значение в ячейке было изменено,

# то в словарь записывается пара: название поля, новое значение

self.modified[self.titles[item.column()]] = item.text()

def save\_results(self):

if self.modified:

cur = self.con.cursor()

que = "UPDATE films SET\n"

for key in self.modified.keys():

que += "{}='{}'\n".format(key, self.modified.get(key))

que += "WHERE id = ?"

cur.execute(que, (self.spinBox.text(),))

self.con.commit()

app = QApplication(sys.argv)

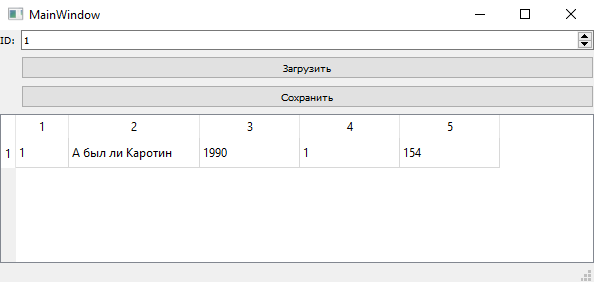
ex = MyWidget()

ex.show()

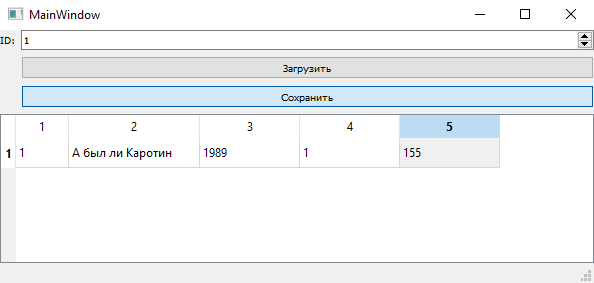
sys.exit(app.exec\_())

Для того чтобы обработать изменение содержимого ячейки, воспользуемся сигналом self.tableWidget.itemChanged. Будем записывать в словарь пару, состоящую из наименования поля и нового значения. Это происходит в функции item\_changed. Затем нам необходимо сохранить полученные результаты в БД. Так как заранее мы не знаем, какие поля были модифицированы, мы не можем использовать конструкцию с вопросительным знаком. Так что просто «склеим» необходимый нам запрос, используя обыкновенную конкатенацию строк. Это описывается в функции save\_results.

Изначальное отображение:



После изменения и сохранения:

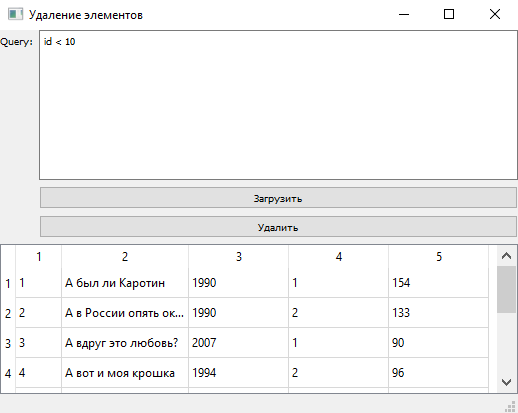


Очень часто для редактирования или создания нового элемента не используют основное окно программы, а открывают дополнительное. Такой подход позволяет оптимизировать ресурсы и не «пробегаться» по множеству элементов.

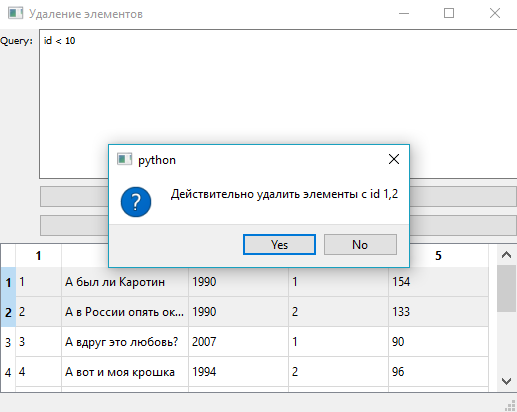
Теперь рассмотрим программу для удаления выделенного элемента. Часто перед удалением какой-либо информации различные программы запрашивают у пользователя подтверждение. Сделаем программу с похожей функциональностью. Будем работать только с таблицей «Фильмы». И теперь в поле для ввода будем вводить только условие. А сам запрос будет формироваться так:

que = "SELECT \* FROM Films WHERE " + self.textEdit.toPlainText()

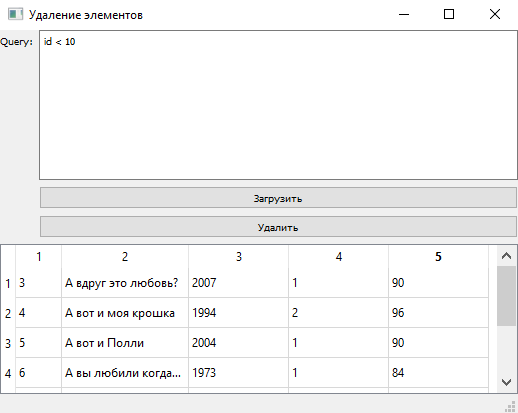
И теперь интерфейс нашей программы, выглядит вот так:



Пользователь может выделить одно или несколько значений, а затем нажать «Удалить». После этого открывается окно с подтверждением, в котором указаны все ID, которые были выделены.



В случае положительного ответа при повторном запросе удаленные элементы уже не отобразятся.



Основная функция, отличающая данную программу от предыдущих, — функция удаления выделенных элементов. Рассмотрим ее поподробнее.

def delete\_elem(self):

rows = list(set([i.row() for i in self.tableWidget.selectedItems()]))

ids = [self.tableWidget.item(i, 0).text() for i in rows]

valid = QMessageBox.question(self, '',

"Действительно удалить элементы с id " +

",".join(ids), QMessageBox.Yes, QMessageBox.No)

if valid == QMessageBox.Yes:

cur = self.con.cursor()

cur.execute("DELETE from films WHERE ID in (" +

", ".join('?' \* len(ids)) + ")", ids)

self.con.commit()

В первой строке мы получаем список строк без повторов. Откуда взялись повторы? Выделяя строку, мы выделяем все элементы строки, соответственно, в список добавляется номер строки столько раз, сколько в ней элементов.

Поскольку номера строк очень часто не совпадают с идентификаторами, во второй строке мы получаем ID записей, по которым мы и будем производить удаление.

В следующей строке мы создаем объект «Вопрос» класса **QMessageBox** (важно не забыть импортировать его из PyQt5.QtWidgets). В качестве параметров ему передаются:

* Родитель — self
* Заголовок — обычно передается пустое поле, если мы хотим задать пользователю вопрос
* Текст вопроса
* Варианты ответов — QMessageBox.Yes, QMessageBox.No

После того как пользователь нажмет на одну из кнопок, результат будет занесен в переменную valid. А затем будет выполнена проверка и удаление.

Важно обратить внимание на то, что текст запроса формируется с использованием и конкатенации строк, и оператора "?". В данной задаче мы впервые столкнулись с оператором self.con.commit(). Этот оператор очень важен при изменении БД: он сохраняет в текущем файле все изменения. Если забыть его вызвать, при следующем запуске программы все, что было исправлено, не сохранится.

[Справка](https://yandex.ru/support/lyceum-students)

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках проекта «Яндекс.Лицей», принадлежат АНО ДПО «ШАД». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «ШАД».

© 2018 – 2020  ООО «[Яндекс](https://yandex.ru/)»

Чаты