Урок QT SQL 1

Введение в БД, работа с SQL-таблицами и отображение данных в PyQT. Часть 1

Аннотация

На уроке мы начнем знакомство с базами данных и языком SQL. Это большая тема, которая будет сопровождать нас до конца обучения, периодически «обрастая» новыми подробностями.

1. Введение в базы данных

На прошлых уроках вы уже сталкивались с хранением данных во внешних источниках: простых текстовых документах, документах с особым форматированием, например, csv-таблицах. Однако такая организация хранения мало пригодна при большом объеме информации по нескольким причинам: в них много дублирования, из-за чего требуется значительно больше места на жестком диске, а поиск работает медленно и слишком «дорогой» при сколько-либо значимом количестве обращений. Разработчики промышленного программного обеспечения столкнулись с этими проблемами достаточно давно, и в качестве одного из решений еще в 1970 году **Эдгар Кодд** предложил реляционную модель данных (файловые базы данных появились еще раньше — в 1955 году). Эта идея развилась в привычные сегодня для почти каждого программиста реляционные базы данных (БД).

База данных — это непосредственное хранилище информации, которое без инструментов для взаимодействия с ним не очень то и полезно. Такой интерфейс для общения с БД разработчикам и системным администраторам предоставляет специальное программное обеспечение — Системы управления базами данных (**СУБД**).

Существует достаточно много различных коммерческих и бесплатных СУБД. В наших проектах мы будем

использовать компактную встраиваемую реляционную СУБД SQLite по нескольким причинам:

1. SQLite — встраиваемая СУБД, поэтому не требует установки дополнительного программного обеспечения, а движок SQLite представляет собой отдельную библиотеку, написанную на C, которую можно использовать как составную часть вашей программы.
2. SQLite база данных представляет собой один файл, с которым удобно работать.
3. Исходный код SQLite передан в общественное достояние, то есть не существует никаких лицензионных ограничений на использование СУБД, как в некоммерческих, так и в коммерческих целях.
4. Большая часть дополнительных инструментов для работы с SQLite бесплатна.
5. И, наконец, в составе стандартной библиотеки Python уже содержится библиотека для работы с SQLite, даже не придется ничего устанавливать с использованием pip.

Но не надо думать, что раз «SQLite» имеет слово «lite» в названии, то это какая-то «игрушечная» СУБД, которая используется только для обучения и при создании «настоящего» программного обеспечения не используется. Простота и удобство встраивания SQLite привели к тому, что библиотека используется в браузерах, музыкальных плеерах и многих других программах, например: Skype, Viber, Яндекс.Браузер, Google Chrome, Mozilla Firefox, Safari, Opera, Adobe Lightroom и т. д.

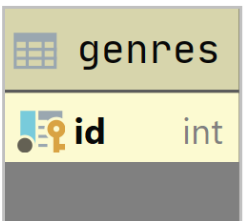
Данные в SQLite базе данных, как и в любых других реляционных БД, хранятся с помощью таблиц и связей между этими таблицами. Если говорить в терминах баз данных, то таблицы — это сущности, а связи — отношения между этими сущностями. Строго говоря, проектирование и использование баз данных — серьезная область знаний в информационных технологиях, существует несколько разных профессий, специалисты в каждой из которых отвечают за отдельные части жизненного цикла базы данных. Поэтому рассмотреть все детали в рамках нашего курса, очевидно, не получится, и мы постараемся сосредоточиться на основных понятиях этой предметной области, которые вам пригодятся с вероятностью около ста процентов.

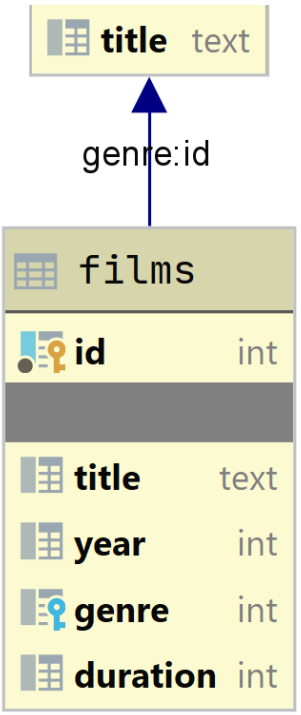
2. Основы SQL

Давайте рассмотрим модельный пример SQLite-базы данных. Она представляет собой хранилище информации о фильмах: название, год выпуска, жанр и продолжительность в минутах. Каждое поле занимает определенный объем в зависимости от типа данных: целочисленный, строковый или используемый для хранения времени. Подробнее о том, какие типы данных поддерживает SQLite, можно почитать **тут**.

Как мы уже говорили, отмечая недостатки хранения информации в csv-файлах, при хранении информации о каждом фильме полностью придется занимать место под название жанра. Поскольку у нас может быть в общем случае сотни тысяч записей о фильмах и всего несколько десятков разных названий жанров, хранить эту информацию в таком виде очень расточительно по отношению к ресурсам. В реляционных базах данных в подобных случаях создается дополнительная таблица, в которой в нашем случае будем хранить пару формата **Ключ: Значение**, где в качестве ключа используется уникальный целочисленный идентификатор (id), а в качестве значения — название жанра (title). А в первой таблице просто хранится ссылка на вторую.

Итак у нас есть база данных **films_db.sqlite**. Давайте сначала посмотрим, как она выглядит в виде визуализации:





Такие визуализации используют достаточно часто для наглядного представления таблиц и связей, и называются они ER-диаграммами (сокращение от Entity-Relation или Сущность-Отношение).

Итак, разберем, что есть в нашей базе данных. Есть две таблицы-сущности: films для хранения информации о фильмах и genres для хранения информации о жанрах. Таблица genres состоит всего из двух полей: id и title, в которых для каждого жанра хранится его идентификатор и название соответственно.

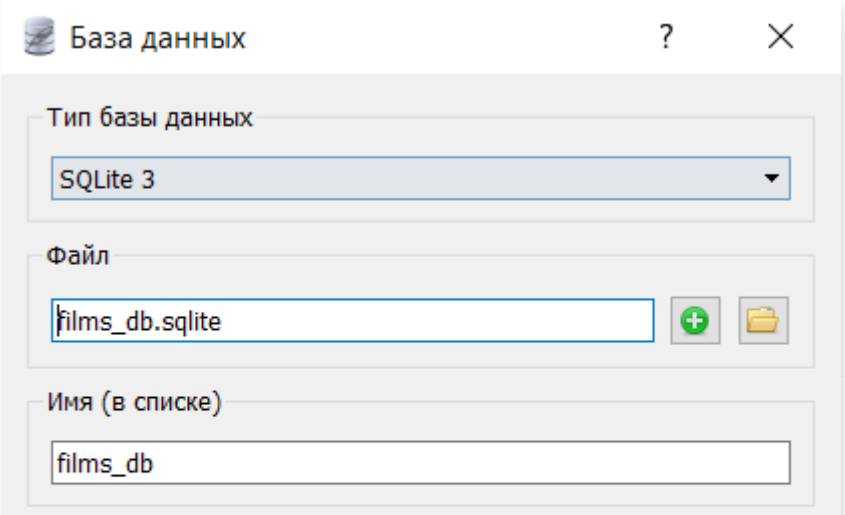
Таблица films чуть сложнее. Там есть поля id, title, year, genre, duration для хранения идентификатора фильма, его названия, года выпуска, идентификатора жанра и длительности в минутах.

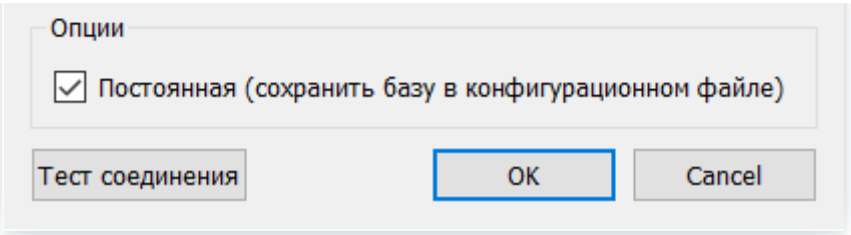
Кроме того, как видно на диаграмме, между таблицами есть связь, которая говорит о том, что номер жанра genre у записи films соответствует записи в таблице genres с таким же значением идентификатора.

Для работы с базами данных был придуман специальный язык — SQL (structured query language — «язык структурированных запросов»). Прежде чем начать работать с БД из Python, давайте немного попрактикуемся в написании запросов с помощью отдельного программного продукта. **SQLiteStudio** — официальный менеджер SQLite баз данных. Менеджеры баз данных — это специальный класс ПО, предназначенный для удобного создания и управления базами данных, написания и отладки SQL-запросов.

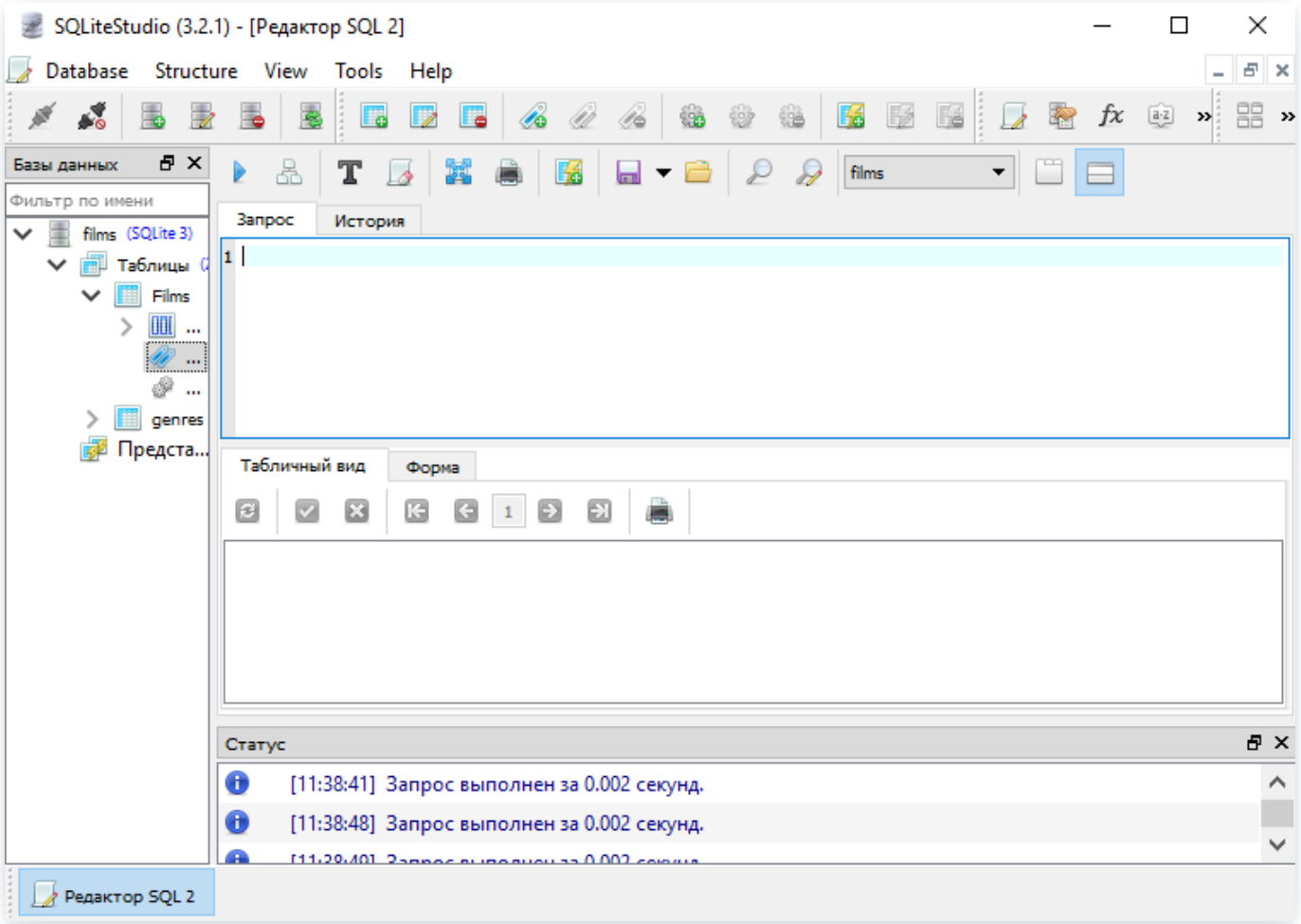
Скачайте и установите SQLiteStudio.

Первое, что необходимо сделать после установки — добавить нашу базу. Для этого в основном меню необходимо выбрать пункт **Add a Database** и в открывшемся окне указать путь к файлу нашей БД.





Если все прошло успешно, можно писать запросы. Для этого необходимо открыть редактор SQL. Его логотип выглядит как свиток бумаги с карандашом. Интерфейс редактора состоит из двух частей: первая, где пишется сам запрос, и вторая, где отображаются результаты.



Основной командой для получения какой-либо информации из БД является команда `SELECT`. Ее базовый синтаксис выглядит так:

```
SELECT перечень_полей FROM имя_таблицы
        WHERE условие
```

Кроме этого, есть и различные модификаторы этой команды. Например, `ORDER BY ПОЛЕ` — тогда результаты будут выведены в отсортированном виде по заданному полю или нескольким полям, а в условии может быть вложенный запрос.

Напишем наш первый запрос. Получим все фильмы, выпущенные в 2010 году.

```
SELECT * FROM Films
    WHERE year = 2010
```

	id	title	year	genre	duration
1	248	Алиса в стране чудес	2010	13	
2	4382	Железный человек 2	2010	11	
3	9138	Ноттингем	2010	11	
4	15495	Утомленные солнцем: Предстояние	2010	2	

Символ * обозначает, что нам необходимо получить все поля. Однако очень часто нам нужно получить только одно или два поля. Модифицируем запрос так, чтобы выводилось только название.

```
SELECT title FROM Films
    WHERE year = 2010
```

	title
1	Алиса в стране чудес
2	Железный человек 2
3	Ноттингем
4	Утомленные солнцем: Предстояние

Условий может быть и несколько: работают все знакомые нам логические операторы NOT, AND и OR. Например, выберем фильмы, выпущенные после 2005 года с продолжительностью от 40 минут до 1,5 часов:

```
SELECT * FROM Films
    WHERE year > 2005 AND duration >= 45 AND duration <= 90
```

	id	title	year	genre	duration
1	3	А вдруг это любовь?	2007	1	90
2	39	Абсурдистан	2008	1	88
3	148	Адреналин	2006	11	87
4	160	Адский бункер	2008	11	90
5	173	Азиат	2008	11	90
6	174	Азирис Нуна	2006	16	90
7	232	Александра	2007	2	90
8	414	Андрей Миронов. Обыкновенное чудо	2006	5	52
9	448	Антидурь	2007	11	90
10	529	Астерикс и викинги	2006	13	78

А как вывести все фильмы определенного жанра, например, **фантастика**? Конечно, можно сходить в таблицу genres и посмотреть, какой id у жанра фантастика, а потом написать запрос вроде такого:

```
SELECT title FROM films
    WHERE genre = 8
```

Но это плохой путь, потому что через некоторое время данные могут измениться, и id у фантастики может стать другим, тогда наш запрос будет давать ошибочный результат. Поэтому правильным решением в данной ситуации будет написать подзапрос, который сам найдет нам необходимое значение id.

```
SELECT title FROM Films
    WHERE genre=(
SELECT id FROM genres
    WHERE title = 'фантастика')
```

Сначала выполнится внутренний запрос: из таблицы `genres` будет получен id для записи с title«Фантастика», а затем будет выполнено сравнение и выведен результат.

Помимо того, может быть выполнено сравнение не с одним элементом, а проверка на попадание в список. Это делается с помощью уже знакомого нам оператора `IN`. Например, так можно выбрать фильмы, продолжительность которых строго 45 или 90 минут:

```
SELECT title, duration FROM Films
WHERE duration IN (45, 90)
```

Кроме уже операторов знакомых нам по Python, SQL содержит еще и ряд тех, которых в Python нет. Давайте

рассмотрим несколько из них.

Оператор `BETWEEN` — проверяет, попадает ли заданное значение в диапазон (включая границы).

```
SELECT * FROM Films
    WHERE (year > 2005) AND duration BETWEEN 45 AND 60
```

Оператор `LIKE` позволяет проверить, насколько похожа та или иная строка на заданный шаблон. Для шаблонов используются специальные символы:

- `%` — обозначает любое количество, в том числе нулевое, любых символов
- `_` — обозначает один любой символ

Давайте получим список фильмов, у которых первая буква в названии — **А** и третья — **к**.

```
SELECT * FROM Films
    WHERE title like 'А_к%'
```

	id	title	year	genre	duration
1	9	А к нам цирк приехал	1978	2	23
2	10	А как же Боб?	1991	1	97
3	11	А кто-то все видит	1994	4	88
4	196	Аккаттоне!	1961	2	120
5	197	Аккомпаниаторша	1993	2	111
6	198	Аккумулятор	1994	8	102
7	423	Анкор, еще анкор!	1992	1	101
8	491	Арктическая тоска	1993	11	91
9	556	Аукцион	1983	13	89

Оператор `LIKE` работает также в паре с `NOT`. Например, получим список фильмов, у которых третья буква в названии не равна **д**, а последняя не равна **а**.

```
SELECT * FROM films
    WHERE title NOT LIKE '__д%а'
```

Кроме этого, есть возможность избавиться от повторов, используя в запросе специальный оператор — `DISTINCT`. Например, вот так можно получить список годов, в которые выходили фильмы в нашей базе данных, без повторений.

```
SELECT DISTINCT year FROM Films
```

3. Работаем с SQLite базой данных из Python

Так как различных СУБД достаточно много, крайне неудобно было бы, если при переходе на новую СУБД приходилось бы с нуля изучать библиотеку для работы с ней. Чтобы избежать таких ситуаций, есть специальный стандарт **PEP 249** (Python Database API Specification v2.0), в котором, помимо всего прочего, описано, какой интерфейс должна предоставлять программисту любая библиотека для работы с базами данных. Поэтому, какую бы СУБД вы не выбрали для управления хранением данных вашего приложения, принципы работы с ней будут очень похожи. Для работы с SQLite из Python используется библиотека sqlite3, которая реализует этот стандарт.

PEP 249 оперирует такими понятиями, как подключения и курсоры:

– **Подключение** — объект, в котором чаще всего указывается либо путь к файлу, либо путь к серверу. Он отвечает только за подключение к БД и, соответственно, отключение от нее

– **Курсор** — объект, в котором непосредственно производится работа с БД

Напишем программу (пока что без графического интерфейса), которая получает результаты одного из рассмотренных выше запросов и выводит их в консоль.

```
# Импорт библиотеки
import sqlite3

# Подключение к БД
con = sqlite3.connect("films_db.sqlite")

# Создание курсора
cur = con.cursor()

# Выполнение запроса и получение всех результатов
result = cur.execute("""SELECT * FROM films
            WHERE year = 2010""").fetchall()

# Вывод результатов на экран
for elem in result:
    print(elem)

con.close()
```

```
(248, 'Алиса в стране чудес', 2010, 13, 201)
(4382, 'Железный человек 2', 2010, 11, 287)
(9138, 'Ноттингем', 2010, 11, 188)
(15495, 'Утомленные солнцем: Предстояние', 2010, 2, 139)
```

Как можно заметить, результат запроса — это список кортежей.

Метод `.fetchall()` возвращает все полученные элементы. Существует еще метод `.fetchone()`, возвращающий, как несложно догадаться, только первый элемент, и метод `.fetchmany(n)`, возвращающий n первых записей.

Для запросов очень часто необходимо указывать какие-либо параметры, в нашем случае: год выпуска, продолжительность фильма и т. д. Для этого существует удобный синтаксис. Вместо значения в запросе указывается вопросительный знак, а затем вторым параметром в итерируемом объекте (чаще всего в кортеже) указываются необходимые значения для подстановки.

```
result = cur.execute("""SELECT * FROM films
            WHERE year = ? and duration > ?""", (2010, 90)).fetchall()
```

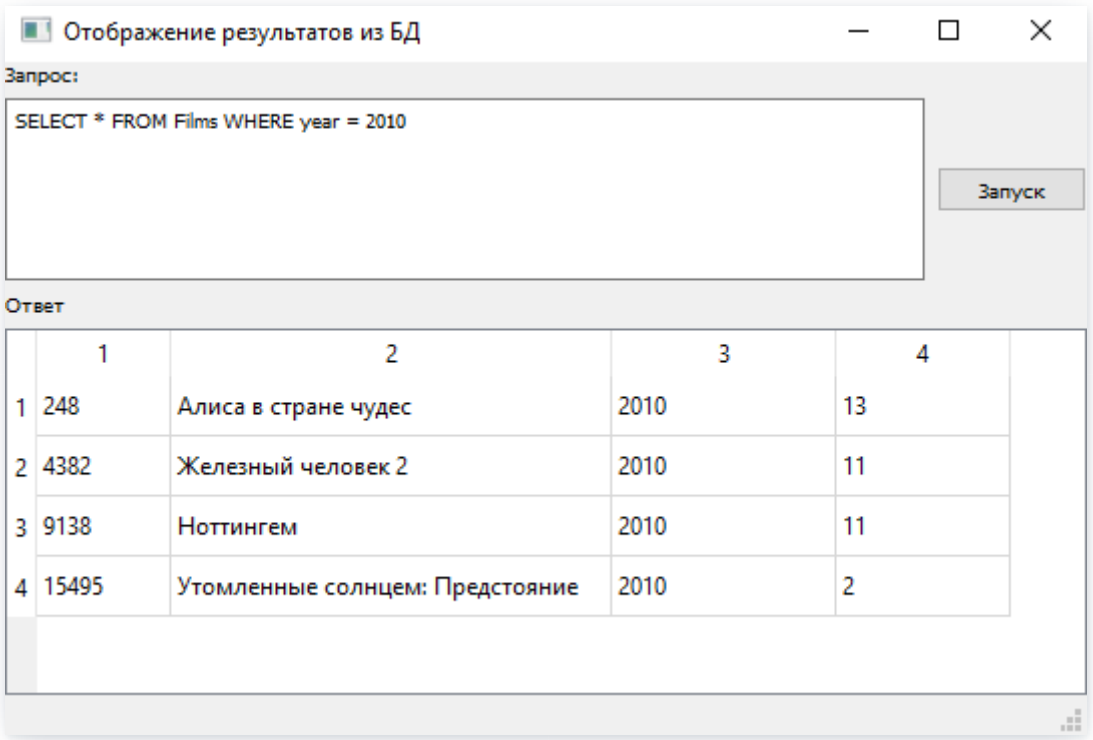
Важно не забыть, что, если мы указываем кортеж из одного элемента, нам все равно необходимо после него поставить запятую.

```
result = cur.execute("""SELECT * FROM Films
            WHERE year = ?""", (2009,)).fetchall()
```

Давайте добавим к нашему приложению графический пользовательский интерфейс. Напишем программу,

которая будет отображать результаты введенного запроса в таблице QTableWidget.

С помощью QtDesigner создадим интерфейс: поле для ввода запроса, таблица для отображения результатов и кнопка для запуска выполнения запроса. Работа с данными из таблицы базы данных с помощью виджета QTableWidget полностью аналогична тому, что мы рассматривали на прошлом уроке во время работы с .csv-файлами.




```
import sqlite3
import sys

from PyQt5 import uic
from PyQt5.QtWidgets import QApplication
from PyQt5.QtWidgets import QMainWindow, QTableWidgetItem


class DBSample(QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('UI1.ui', self)
        self.connection = sqlite3.connect("films_db.sqlite")
        self.pushButton.clicked.connect(self.select_data)
        # По умолчанию будем выводить все данные из таблицы films
        self.textEdit.setPlainText("SELECT * FROM films")
        self.select_data()

    def select_data(self):
        # Получим результат запроса,
        # который ввели в текстовое поле
        query = self.textEdit.toPlainText()
        res = self.connection.cursor().execute(query).fetchall()
        # Заполним размеры таблицы
        self.tableWidget.setColumnCount(5)
        self.tableWidget.setRowCount(0)
        # Заполняем таблицу элементами
```

```
        for i, row in enumerate(res):
            self.tableWidget.setRowCount(
                self.tableWidget.rowCount() + 1)
            for j, elem in enumerate(row):
                self.tableWidget.setItem(
                    i, j, QTableWidgetItem(str(elem)))

    def closeEvent(self, event):
        # При закрытии формы закроем и наше соединение
        # с базой данных
        self.connection.close()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = DBSample()
    ex.show()
    sys.exit(app.exec())
```

4. Возможности PyQT по работе с базами данных

В курсе мы и дальше будем говорить про работу с базами данных как можно более обще, но нельзя не отметить то, что у PyQT (как и у некоторых других больших библиотек) есть своя универсальная надстройка для работы с БД — модуль PyQt5.QtSql. Рассмотрим простейший пример: отобразим все данные из таблицы films.

```
import sys

from PyQt5.QtSql import QSqlDatabase, QSqlTableModel
from PyQt5.QtWidgets import QWidget, QTableView, QApplication


class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        # Зададим тип базы данных
        db = QSqlDatabase.addDatabase('QSQLITE')
        # Укажем имя базы данных
        db.setDatabaseName('films_db.sqlite')
        # И откроем подключение
        db.open()

        # QTableView - виджет для отображения данных из базы
        view = QTableView(self)
        # Создадим объект QSqlTableModel,
        # зададим таблицу, с которой он будет работать,
        #  и выберем все данные
        model = QSqlTableModel(self, db)
```

```
        model.setTable('films')
        model.select()

        # Для отображения данных на виджете
        # свяжем его и нашу модель данных
        view.setModel(model)
        view.move(10, 10)
        view.resize(617, 315)

        self.setGeometry(300, 100, 650, 450)
        self.setWindowTitle('Пример работы с QtSql')


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    ex.show()
    sys.exit(app.exec())
```

Общий принцип работы с модулем следующий:

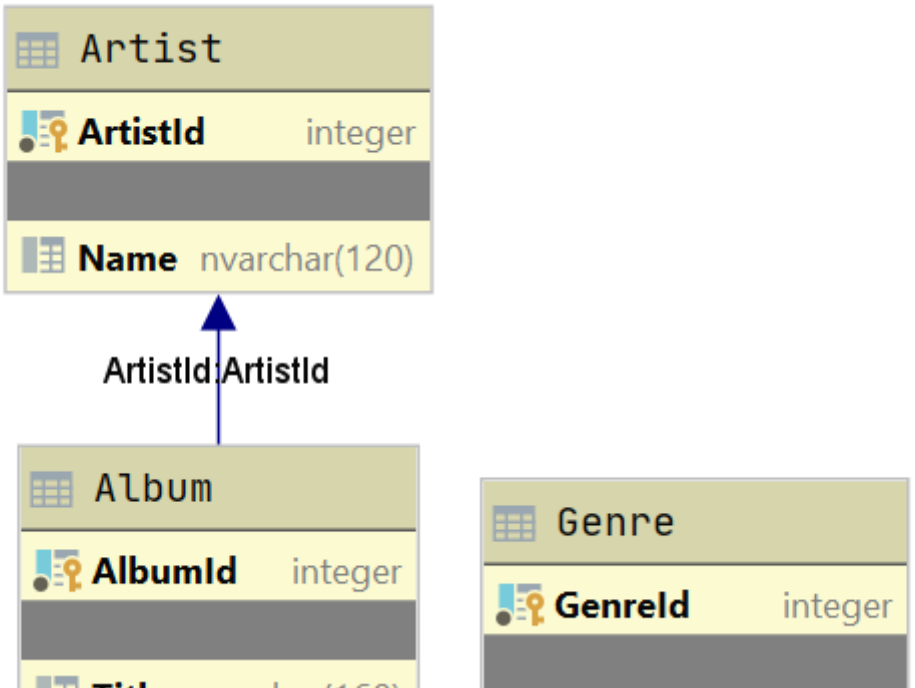
- Создаем и настраиваем объект QSqlDatabase для связи с базой данных
- С помощью QSqlTableModel или QSqlQueryModel получаем и управляем данными из базы данных
- С помощью виджета QTableView отображаем данные в табличном виде

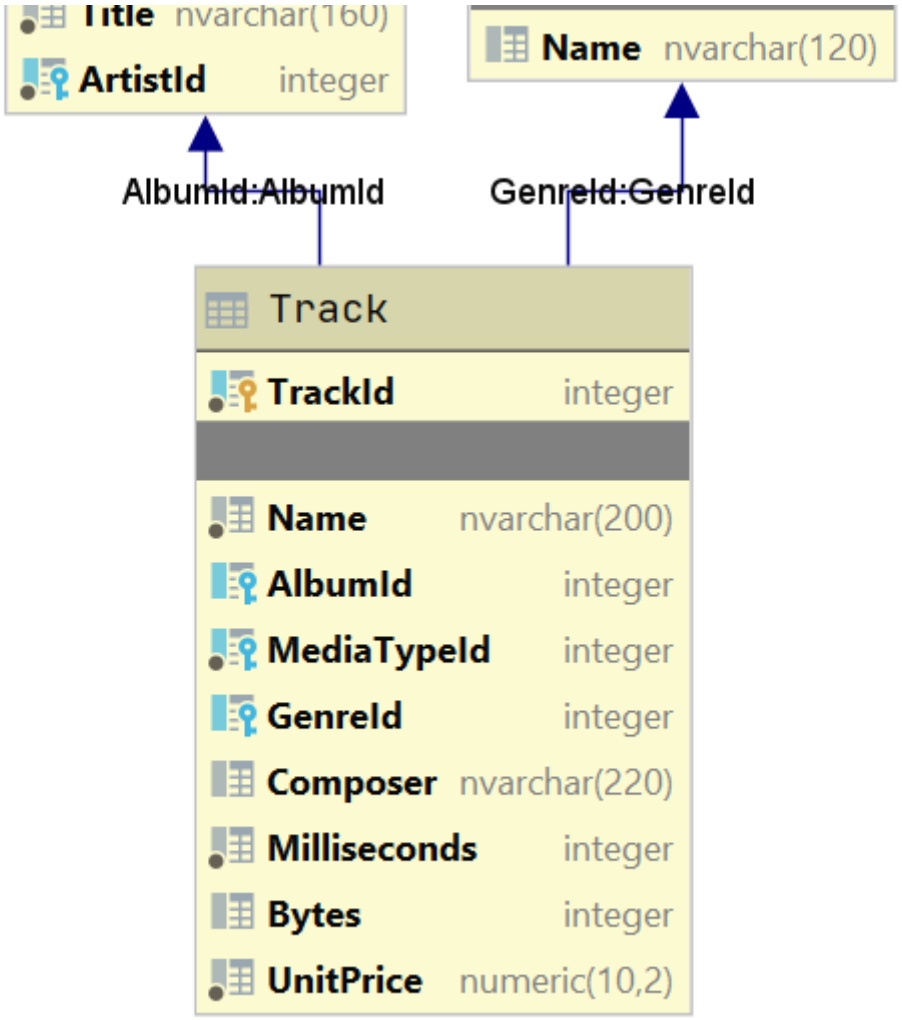
Подробнее почитать про работу с базами данных, использованием модуля QtSql можно почитать в официальной **документации**.

5. SQL: Получение данных из нескольких таблиц

Язык SQL позволяет при помощи одного запроса получать информацию сразу из нескольких таблиц в базе данных. Рассмотрим для примера базу данных **Chinook_Sqlite.sqlite**

В ней много таблиц, но нас интересуют только некоторые.





В таблице **Album** в колонке **ArtistId** содержится код артиста из таблицы **Artist**. Каждому артисту может принадлежать несколько или ноль альбомов. Чтобы получить сразу данные из двух таблиц **Album** и **Artist** можно использовать команды `INNER JOIN`, `LEFT JOIN`.

Приведенный ниже запрос вернет названия альбомов и соответствующие им имена артистов.

```
SELECT
    album.Title,
    artist.Name
FROM
    album
INNER JOIN artist
    ON artist.ArtistId = album.ArtistId;
```

Вот, что мы получим:

	Title	Name
1	For Those About To Rock We Salute You	AC/DC
2	Balls to the Wall	Accept
3	Restless and Wild	Accept
4	Let There Be Rock	AC/DC
5	Big Ones	Aerosmith
6	Jagged Little Pill	Alanis Morissette
7	Facelift	Alice In Chains
8	Warner 25 Anos	Antônio Carlos Jobim

8	Warner 25 Anos	Antonio Carlos Jobim
9	Plays Metallica By Four Cellos	Apocalyptica
10	Audioslave	Audioslave
11	Out Of Exile	Audioslave
12	BackBeat Soundtrack	BackBeat
13	The Best Of Billy Cobham	Billy Cobham
14	Alcohol Fueled Brewtality Live! [Disc 1]	Black Label Society
15	Alcohol Fueled Brewtality Live! [Disc 2]	Black Label Society
16	Black Sabbath	Black Sabbath

В этом примере команда `INNER JOIN` сопоставляет каждую строку из таблицы **Album** с каждой строкой из таблицы **Artist** основываясь на условии `artist.ArtistId = album.ArtistId`, которое указано после ключевого слова `ON`. Если условие выполняется, то значения из обеих таблиц для колонок, которые мы указали в секции `SELECT` запроса, добавляются в результат его выполнения. Таким образом, если для альбома не будет найдено соответствие в таблице с артистами или наоборот, то такие записи **не будут добавлены в результат выполнения запроса**.

Приведенный ниже запрос так же вернет названия альбомов и соответствующие им имена артистов.

```
SELECT
    album.Title,
    artist.Name
FROM
    album
LEFT JOIN artist
    ON artist.ArtistId = album.ArtistId;
```

В этом примере команда `LEFT JOIN` выберет **все записи из таблицы Album** (таблица находится слева от команды) и для тех строк, для которых выполняется условие `artist.ArtistId = album.ArtistId`, указанное после ключевого слова `ON`, сопоставит строку из таблицы **Artist**. Если же для каких-то строк условие не выполнится, то колонки, получаемые из таблицы **Artist останутся пустыми (null)**.

В одном запросе можно использовать неограниченное количество соединений (JOIN'ов). В запросе приведенном ниже мы выберем названия всех треков из таблицы **Track** добавим к нему название жанра из таблицы **Genre**, название альбома из таблицы **Album** и имя артиста из таблицы **Artist**, и упорядочим результат по имени трека.

```
SELECT
    Track.Name as TrackName,
    Genre.Name as GenreName,
    Album.Title as AlbumTitle,
    Artist.Name
FROM
    Track
LEFT JOIN Genre ON Track.GenreId = Genre.GenreId
LEFT JOIN Album ON Track.AlbumId = Album.AlbumId
LEFT JOIN Artist ON Album.ArtistId = Artist.ArtistId
ORDER BY TrackName;
```

Результат:

	TrackName	GenreName	AlbumTitle	Name
1	"40"	Rock	War	U2

1	"40"	Rock	War	U2
2	"?"	TV Shows	Lost, Season 2	Lost
3	"Eine Kleine Nachtmusik" Serenade In G, K...	Classical	Sir Neville Marriner: A Celebration	Academy of St. Martin in the Fields Chamb...
4	#1 Zero	Alternative & Punk	Out Of Exile	Audioslave
5	#9 Dream	Pop	Instant Karma: The Amnesty International ...	U2
6	'Round Midnight	Jazz	The Essential Miles Davis [Disc 1]	Miles Davis
7	(Anesthesia) Pulling Teeth	Metal	Kill 'Em All	Metallica
8	(Da Le) Yaleo	Rock	Supernatural	Santana
9	(I Can't Help) Falling In Love With You	Reggae	UB40 The Best Of - Volume Two [UK]	UB40
10	(Oh) Pretty Woman	Rock	Diver Down	Van Halen
11	(There Is) No Greater Love (Teo Licks)	Pop	Frank	Amy Winehouse
12	(We Are) The Road Crew	Metal	Ace Of Spades	Motörhead
13	(White Man) In Hammersmith Palais	Alternative & Punk	The Singles	The Clash
14	(Wish I Could) Hideaway	Rock	Chronicle, Vol. 2	Creedence Clearwater Revival
15	...And Found	TV Shows	Lost, Season 2	Lost
16	...And Justice For All	Metal	...And Justice For All	Metallica
17	...In Translation	TV Shows	Lost, Season 1	Lost
18	.07%	Drama	Heroes, Season 1	Heroes

Результаты запросов с соединениями (JOIN) также можно фильтровать при помощи команды `WHERE`. Добавим в предыдущий запрос условие, для получения только треков исполнителя "U2".

```
SELECT
    Track.Name as TrackName,
    Genre.Name as GenreName,
    Album.Title as AlbumTitle,
    Artist.Name
FROM
    Track
LEFT JOIN Genre ON Track.GenreId = Genre.GenreId
LEFT JOIN Album ON Track.AlbumId = Album.AlbumId
LEFT JOIN Artist ON Album.ArtistId = Artist.ArtistId
WHERE Artist.Name = "U2"
ORDER BY TrackName;
```

Результат:

	TrackName	GenreName	AlbumTitle	Name
1	"40"	Rock	War	U2
2	#9 Dream	Pop	Instant Karma: The Amnesty International ...	U2
3	A Man And A Woman	Rock	How To Dismantle An Atomic Bomb	U2
4	A Room At The Heartbreak Hotel	Rock	B-Sides 1980-1990	U2
5	Acrobat	Rock	Achtung Baby	U2
6	All Along The Watchtower	Rock	Rattle And Hum	U2
7	All Because Of You	Rock	How To Dismantle An Atomic Bomb	U2
8	All I Want Is You	Rock	Rattle And Hum	U2
9	All I Want Is You	Rock	The Best Of 1980-1990	U2
10	Angel Of Harlem	Rock	Rattle And Hum	U2
11	Angel Of Harlem	Rock	The Best Of 1980-1990	U2
12	Babyface	Rock	Zooropa	U2
13	Bad	Rock	The Best Of 1980-1990	U2
14	Bass Trap	Rock	B-Sides 1980-1990	U2
15	Beautiful Boy	Pop	Instant Karma: The Amnesty International ...	U2

16	Beautiful Day	Rock	All That You Can't Leave Behind	U2
17	Bullet The Blue Sky	Rock	Rattle And Hum	U2
18	City Of Blinding Lights	Rock	How To Dismantle An Atomic Bomb	U2

6. Заключение

Итак, сегодня мы познакомились с базами данных в целом, подробнее остановились на реляционных на примере SQLite. Изучили одну из основных конструкций SQL — запрос SELECT для выборки произвольных данных из БД (разумеется, увидели только небольшую часть возможностей, более детально можно почитать **тут**). Кроме того, посмотрели, как подключаться и взаимодействовать с базами данных из Python-приложений.

На следующем уроке мы продолжим работу с базами данных и посмотрим, как можно манипулировать данными: создавать новые записи, изменять и удалять существующие.