**Лабораторная работа №5**

**Организация взаимодействия с базой данных через консольное приложение**

**Цель работы:** получить навыки подключения к различным системам управления базами данных и взаимодействия с ними. Разработать консольное приложение для взаимодействия с базой данных.

**Основные теоретические сведения**

Python – высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью.

Доступ к вашим базам данных посредством языка программирования позволяет создавать приложения, которые могут хранить и извлекать данные непосредственно из пользовательского интерфейса или для него. Существуют различные библиотеки, которые выполняют эту задачу для языка программирования Python. Наиболее распространенными являются библиотеки для взаимодействия с такими распространёнными СУБД, как PostgreSQL, MySQL и SQLite.

PostgreSQL – свободная объектно-реляционная система управления базами данных. Существует в реализациях для множества UNIX-подобных платформ, включая AIX, различные BSD-системы, HP-UX, IRIX, Linux, macOS, Solaris/OpenSolaris, Tru64, QNX, а также для Microsoft Windows.

MySQL – свободная реляционная система управления базами данных. MySQL является решением для малых и средних приложений. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

SQLite – компактная [встраиваемая СУБД](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D0%BC%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94). Слово «встраиваемый» (embedded) означает, что SQLite не использует парадигму [клиент-сервер](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80), то есть [движок](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%85%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F) SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой [библиотеку](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), с которой программа компонуется, и движок становится составной частью программы.

Для выполнения данной лабораторной работы потребуется базовое понимание синтаксиса SQL и знание основ языка Python. Вы также должны иметь возможность загружать и импортировать пакеты в Python и знать, как устанавливать и запускать (локально или удаленно) различные серверы баз данных.

Для подключения к серверам баз данных SQLite используйте функцию create\_connection(), которая принимает путь к базе данных SQLite. Функция .connect() из модуля sqlite3 и принимает путь к базе данных SQLite в качестве параметра. Если база данных существует в указанном месте, то соединение с базой данных считается установленным. В противном случае создается новая база данных в указанном месте и устанавливается соединение.

Для выполнения запросов используется объект cursor, который позволяет взаимодействовать с базой данных, добавлять изменять и удалять записи, а также осуществлять выборку данных.  Запрос, который должен быть выполнен, передается в cursor.execute(). Метод execute() может выполнить любой запрос, переданный ему в виде строки. Например, для создания таблиц, их обновления и удаления. Также с помощью курсоров можно выполнять запросы на создание либо изменение структуры баз данных. Все, что нужно сделать, это сохранить запрос в строковой переменной и затем передать эту переменную в cursor.execute().

**Задание к работе**

1. Подключиться к системе управления базами данных с помощью SQL библиотек выбранного языка программирования.

2. Организовать взаимодействие с базами данных выбранной СУБД (PostgreSQL/MySQL/SQLite).

3. Разработать консольное приложение, в котором производится подключение к базе данных, разработанной на основе предыдущих лабораторных работ, а также обеспечивается выполнение запросов предшествующей лабораторной работы.

**Пример выполнения задания**

Для выполнения задания потребуется написать консольное приложение для взаимодействия с базой данных. Приложение состоит из двух функциональных частей: сервиса для взаимодействия с базой данных и сервиса для взаимодействия с пользователем. Сервисы представлены классами.

Для выполнения задания будет использоваться СУБД [SQLite](https://www.sqlite.org/docs.html). [SQLite](https://www.sqlite.org/docs.html), вероятно, самая простая система управления базами данных, к которой можно подключиться из приложения на языке Python и для этого нет необходимости подключать внешние модули. По умолчанию установка Python содержит SQL‑библиотеку с именем [sqlite3](https://docs.python.org/2/library/sqlite3.html), которую можно использовать для работы с SQLite.

В коде ниже импортируются необходимые модули для работы со встраиваемой СУБД SQLite и поддержки выполнения аннотации типов.

**import** sqlite3  
**from** typing **import** List, Optional, Tuple, Callable

В первую очередь нужно установить соединение с помощью функции connect и создать объект - cursor.

Код ниже описывает класс ServiceDB для взаимодействия с базой данных, внутри которого будет описано выполнение запросов на создание таблиц.

**class** ServiceDB:  
 **def** \_\_init\_\_(self, name\_db: str):  
 self.\_\_connection: sqlite3.Connection = sqlite3.connect(name\_db)  
 self.\_\_cursor: sqlite3.Cursor = self.\_\_connection.cursor()

При помощи метода cursor.execute() можно выполнять запросы. В коде ниже в качестве примера представлено создание таблицы «contact».

self.\_\_cursor.execute(  
 **'CREATE TABLE IF NOT EXISTS contact ('  
 'id INTEGER PRIMARY KEY,'  
 'fcs VARCHAR(255) NOT NULL,'  
 'phone VARCHAR(12) NOT NULL,'  
 'address VARCHAR(255) NOT NULL'  
 ');'**)

В рамках дальнейшего сопровождения кода приложения писать запрос при каждом взаимодействии с базой данных - неоправданно и неудобно. Функция execute\_select() принимает параметры, необходимые для составления SELECT-запроса, после чего формирует строку запроса, готовую к выполнению, и выполняет её.

**def** execute\_select(  
 self, table: str,  
 joins: Optional[List[Tuple[str, str, str]]] = **None**,  
 fields: Optional[List[str]] = **None**,  
 group\_by: Optional[str] = **None**,  
 order\_by: Optional[str] = **None**,  
 \*\*where  
):  
 fields = [**'\*'**] **if** fields **is None else** fields  
 query = **f'SELECT {", "**.join(fields)**} FROM {**table**}'  
 if** joins:  
 query += **' JOIN '** + **' JOIN '**.join([**f'{**tab**} AS {**short**} ON {**rule**}' for** tab, short, rule **in** joins])  
 **if** where:  
 query += **' WHERE '** + **' AND '**.join([**f'{**x**}={**y**}' for** x, y **in** where.items()])  
 **if** group\_by:  
 query += **f' GROUP BY {**group\_by**}'  
 if** order\_by **and not** order\_by.startswith(**'-'**):  
 query += **f' ORDER BY {**order\_by**} DESC'  
 elif** order\_by:  
 query += **f' ORDER BY {**order\_by[1:]**} ASC'** query += **';'** self.\_\_cursor.execute(query)  
 result = self.\_\_cursor.fetchall()  
 **return** result

В вышеприведенном фрагменте кода: **table** – имя таблицы; **joins** – список всех используемых JOIN. Элемент списка – кортеж, где (имя таблицы, её короткое название, правило соединения); **fields** ­­­– поля таблицы; **group\_by** – поле таблицы, по которому осуществляется группировка; **order\_by** - поле таблицы, по которому осуществляется сортировка; **where** – именованные аргументы, где ключ – это поле, а значение – значение поля таблицы; **result** – список кортежей, каждый элемент которых является значением соответствующего поля fields.

Код ниже описывает класс CommandLineHandler для взаимодействия с пользователем. Он предоставляет удобное меню для работы с базой данных, выбор пунктов меню реализован через консольный клиент. Описан пункт меню на примере «customer».

**class** CommandLineHandler:  
 **def** \_\_init\_\_(self):  
 *# Создаём класс для взаимодействия с БД* self.\_\_db = ServiceDB(**'sql.db'**)  
 *# Тут мы объявляет список всех пунктов меню, которые мы сможем выбрать через консольный клиент* self.\_\_entities = {  
 1: **'Заказчики'**,  
 2: **'Исполнители'**,  
 3: **'Квалификации'**,  
 4: **'Договоры подряда'**,  
 5: **'Виды работ'**,  
 6: **'Статусы договора'**,  
 0: **'Выход'**,  
 }  
 *# словарь, ключи которого есть соответствующий пункт меню, а значение есть кортеж из двух  
 # элементов; первый - это функция, которая получает все необходимые данные из соответствующей таблицы,  
 # и список имён столбцов для красивого представления данных в консоли* self.menu = {  
 1: (  
 **lambda**: self.\_\_db.execute\_select(  
 **'customer'**,  
 fields=[**'contact.fcs'**, **'contact.phone'**, **'contact.address'**],  
 joins=[[**'contact'**, **'contact'**, **'contact.id = contact\_id'**]],  
 order\_by=**'-contact.fcs'** ),  
 [**'ФИО'**, **'Телефон'**, **'Адрес'**]  
 ),

# … здесь следует описание остальных пунктов меню …

**def** \_\_choose\_menu(self) -> int:  
 *# Эта функция предлагает выбрать пункт меню, и после пользовательского ввода возвращает его номер* text = **f'Выберите пункт меню:\n'  
 for** number, name **in** self.\_\_entities.items():  
 text += **f'{**number**}. {**name**}\n'** text += **'\_> '  
 return** int(input(text))  
  
@staticmethod  
**def** \_\_show\_result(func: Callable, columns: List[str]):  
 *"""* **:param** *func: Функция, выполняющая получение данных из БД* **:param** *columns: Имена столбцов, в которые будут записаны результаты выполнения func* **:return***:  
 """* sizes = []  
 **for** column **in** columns:  
 length = len(column)  
 **if** length > 13:  
 sizes.append(length + 2)  
 **else**:  
 sizes.append(15)  
 line = **'+'** + **'='** \* (sum(sizes) + len(columns) - 2) + **'+\n'** text = line  
  
 **for** size, column **in** zip(sizes, columns):  
 text += **f'%{**size**}s'** % column + **'|'** text += **'\n'** text += line  
 **for** row **in** func():  
 **for** size, column **in** zip(sizes, row):  
 text += **f'%{**size**}s'** % column + **'|'** text += **'\n'** text += line  
 **print**(text)  
  
**def** handle(self):  
 *# Эта функция выполняет взаимодействие с пользователем. Она «слушает» от него команды и выполняет их* **while** True:  
 *# Выбираем пункт меню* choose = self.\_\_choose\_menu()  
 **if** choose == 0:  
 *# 0 = Выход из приложения* **return  
 elif** choose **not in** self.menu.keys():  
 *# Выбран несуществующий пункт меню* **raise** IndexError(**f'Ваш выбор {**choose**} не допустим: {**list(self.menu.keys())**}'**)  
 *# Получаем функцию и столбцы* get\_rows\_func, columns = self.menu[choose]  
 *# Выводим результат в красивом виде* self.\_\_show\_result(get\_rows\_func, columns)  
 *# Снова ожидаем ввод*

**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 cmd = CommandLineHandler()  
 cmd.handle()

Изображение выглядит как стол

Автоматически созданное описание Изображение выглядит как стол

Автоматически созданное описание

Изображение выглядит как стол

Автоматически созданное описание

Рис. 4. Снимки экрана выполнения программы

В процессе выполнения лабораторной работы были получены навыки подключения к различным системам управления базами данных и взаимодействия с ними. Получен опыт в разработке консольного приложения, способного обеспечить подключение к базе данных и выполнение запросов.