ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «КАЛИНИНГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Л. Г. Высоцкий

ЭКСПЛУАТАЦИОННАЯ ПРАКТИКА

Учебно-методическое пособие к выполнению практических заданий для студентов, обучающихся в бакалавриате по направлению подготовки 09.03.01 Информатика и вычислительная техника

Калининград

Издательство ФГБОУ ВО «КГТУ»

2022

УДК 004.9(75)

Рецензент:
кандидат педагогических наук, доцент кафедры прикладной информатики
ФГБОУ ВО «Калининградский государственный технический
университет»
Е. Ю. Заболотнова

Высоцкий, Л. Г.
Эксплуатационная практика: учебно-методическое пособие к выполнению практических заданий для студентов, обучающихся в бакалавриате по направлению подготовки 09.03.01 Информатика и вычислительная техника / **Л. Г. Высоцкий**. – Калининград: Изд-во ФГБОУ ВО «КГТУ», 2022. – 28 с.

Учебно-методическое пособие включает указания и материалы, необходимые для прохождения студентами эксплуатационной практики. Все задания выполняются по индивидуальным вариантам. В пособие также включены подробные требования к оформлению отчетов по работам.

Учебно-методическое пособие рассмотрено и одобрено в качестве локального электронного методического материала кафедрой прикладной информатики института цифровых технологий ФГБОУ ВО «Калининградский государственный технический университет» 19 сентября 2022 г., протокол № 3

Учебно-методическое пособие рекомендовано к использованию в качестве локального электронного методического материала в учебном процессе методической комиссией ИЦТ 20 сентября 2022 г., протокол № 6

УДК 004.9(075)

ОГЛАВЛЕНИЕ

Цели и задачи эксплуатационной практики 4

Общие указания к выполнению заданий на практику 5

Введение в проблематику задания 6

1. **Основные понятия** 6
2. **Краткая характеристика СУБД SQLite** 8
3. **Введение в язык SQL** 12
4. **Создание базы данных** 12
5. **Добавление информации в базу данных** 15
6. **Удаление данных из БД** 15
7. **Редактирование (обновление) записей** 16
8. **Выборка данных из БД** 17
9. **Выборка из связанных таблиц** 19

Задания на практику 21

Содержание отчета: 24

Правила оформления отчета по практике 25

Литература: 26

ПРИЛОЖЕНИЕ 27

Цели и задачи эксплуатационной практики

Целью учебной практики является систематизация, обобщение и углубление теоретических знаний, формирование практических умений, общекультурных, профессиональных компетенций и профессиональных компетенций профиля на основе изучения работы организаций, в которых студенты проходят **практику**.

Задачами учебной (производственной) практики по программированию являются:

- получение знаний, необходимых для выполнения выпускной квалификационной работы;

- получение навыков самостоятельного или в составе научно-производственного коллектива решения конкретных профессиональных задач;

- изучение научной литературы или научно-исследовательских проектов в соответствии с профилем объекта профессиональной деятельности;

- изучение информационных систем методами математического прогнозирования и системного анализа;

- изучение больших систем современными методами высокопроизводительных вычислительных технологий, применение современных суперкомпьютеров в проводимых исследованиях;

- составление научных обзоров, рефератов и библиографии по тематике проводимых исследований;

-разработка программного и информационного обеспечения;

- разработка и исследование алгоритмов, вычислительных моделей и моделей данных для реализации элементов новых (или известных) сервисов систем информационных технологий;

- разработка архитектуры, алгоритмических и программных решений системного и прикладного программного обеспечения;

- изучение языков программирования, алгоритмов, библиотек и пакетов программ, продуктов системного и прикладного программного обеспечения.

Общие указания к выполнению заданий на практику

1. Конечным результатом практики является создание полноценной информационной системы, включающей БД и графический пользовательский интерфейс для манипулирования базой, т. е. студент должен пройди весь процесс от постановки задачи до получения оттестированного программного продукта.
2. Выполнение задания предлагается реализовать на языке Python и СУБД SQLite. Но студент может использовать другой инструментарий по своему выбору, обеспечивая требуемый результат
3. Исполняемый вариант программной системы демонстрируется руководителю практики, и при получении положительного отзыва формируется отчет по практике, правила оформления которого указаны ниже.
4. Далее проводится защита отчета с выставлением оценки по пятибалльной системе.

Введение в проблематику задания

1. Основные понятия

Большинство программных приложений в настоящее время создаются для того, чтобы обрабатывать данные, которые предполагают длительное хранение в виде файловой системы или БД (рис. 1). Именно второй вариант является доминирующим в современных информационных системах.

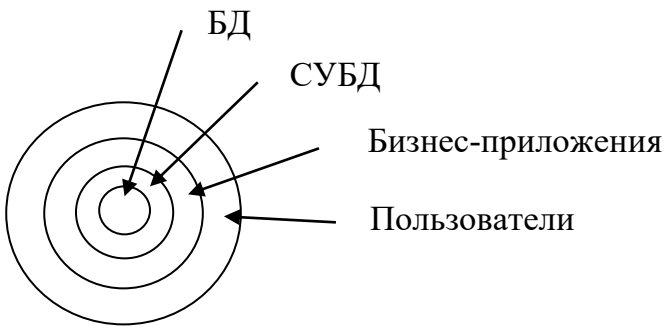

Рис. 1

База данных (БД) представляет собой организованный набор информации. В первую очередь, это набор плоских таблиц. Создаются и поддерживаются БД специальными программными системами, которые называются СУБД (системы управления базами данных). В настоящее время в мире существует примерно с десяток широко используемых СУБД. Причем известны два основных подхода: вся БД хранится в одном файле (реализуется, например, СУБД Microsoft Access) или вся БД хранится в виде нескольких файлов в одной папке (реализуется, например, СУБД Paradox).

Таблица является основой любой БД. Она состоит (рис. 2) из строк (записей) и именованных столбцов (полей).

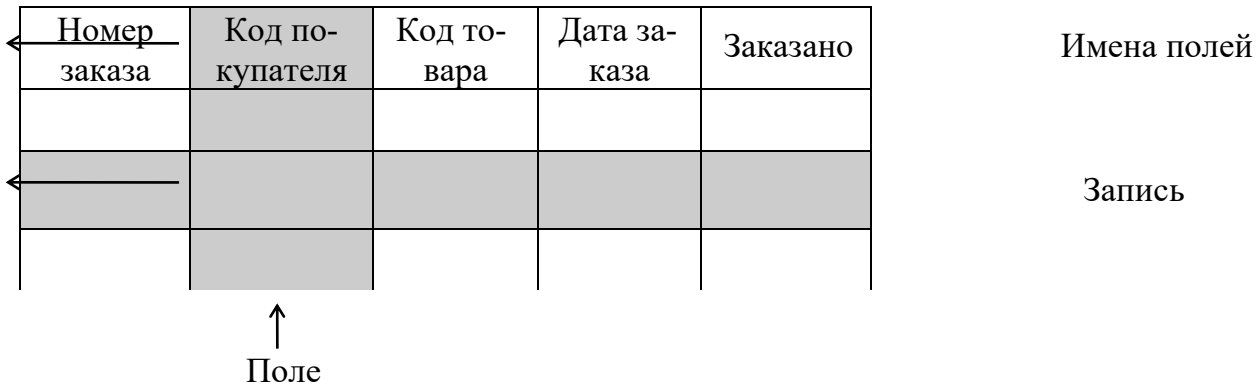

Рис. 2

Каждая таблица описывает некоторый класс объектов выбранной предметной области, например, студентов вуза или преподавателей, а каждая строка-запись содержит информацию о конкретном объекте (студенте или преподавателе). Каждый же столбец-поле описывает один из атрибутов данного объекта, например, должность или дату рождения. Поэтому обычно все данные одного столбца характеризуются

одинаковым типом - множеством допустимых значений и операций над ними. Каждая СУБД использует свой формат описания хранимых данных. Наиболее распространены следующие типы данных в БД:

* *текстовый* A (Alpha) - строка длиной от 1 до 255 символов. Количество символов определяется пользователем в процессе создания таблицы исходя из семантики атрибута. Так, например, для хранения фамилий целесообразно отвести не менее 15 позиций.
* *числовой с плавающей точкой* N(Number).
* *денежный* $ (Money) - вещественные числа с двумя десятичными знаками точности.
* *числовой целый* S (Short).
* *числовой целый длинный* I (Long Integer).
* D (Date) – даты, например, в диапазоне 01/01/999 до н.э. ÷ 31/12/9999.
* T (Time) - время.
* @ (TimeStamp) - дата и время.
* *комментарий* M (Memo) - текст, длина которого не ограничена (первые 240 символов строки хранятся в самой таблице, остальная строка заносится в файл со специальным расширением, имя которого совпадает с именем исходной таблицы).
* *логический* L (Logical).
* *автоинкрементный* + (Autoincrement) - при добавлении новой записи значение в поле с данным типом увеличивается на 1. Обычно поля с данным типом используются как ключевые.

При выборе формата типа необходимо стремиться к минимуму используемой памяти.

Для каждой реляционной таблицы существует понятие *ключа* - набора полей, которые однозначно идентифицируют каждую запись таблицы. Ключи нужны для сортировки содержимого таблицы (для быстрого поиска) и для реализации перекрестных ссылок между таблицами. В такой таблице не допускаются две или более записи с одинаковыми значениями ключевых полей. В общем случае ключ может состоять только из одного поля (*простой ключ*). Составной ключ предполагает идентификацию записи на основе комбинации значений нескольких полей. В этом случае все поля такого ключа должны располагаться последовательно друг за другом. Ключевые поля также должны располагать в начале таблице.

Кроме того, при описании структуры таблицы (формировании ее метаданных) можно задать ограничения на значения конкретных полей (их максимальное и минимальное значения, обязательность, значение по умолчанию) и маску, представляющую собой фильтр вводимых данных.

2. Краткая характеристика СУБД SQLite

СУБД SQLite3 — это часть стандартного пакета Python 3, т.е. она не может функционировать автономно, поэтому при ее использовании ничего дополнительно к пакету Python 3 устанавливать не надо.

SQLite использует более общую систему типизации — динамическую, когда тип данных значения связан с самим значением, а не с его контейнером, который представляет некоторый класс хранения. Динамическая система SQLite имеет обратную совместимость со статическими системами других СУБД. В том смысле, что SQL-запросы статически типизированных баз данных должны работать так же и с SQLite. Однако, динамическая типизация в SQLite позволяет выполнять операции, невозможные в традиционных жестко типизированных базах данных.

Каждое значение, хранящееся в базе данных SQLite (или обрабатываемое движком), имеет один из следующих **классов хранения**:

- NULL — значение NULL
- INTEGER — целое число
- REAL — число с плавающей точкой
- TEXT — текст
- BLOB — (large binary object — «большой бинарный объект») — это тип данных, который используется для хранения «тяжелых» файлов, таких как изображения, видео, музыка, документы и так далее. Перед сохранением в базе данных эти файлы нужно конвертировать в бинарные данные — т. е. массив байтов.

В реальности же класс хранения - это более широкое понятие, чем тип данных, это семейство совместимых по операциям и возможным значениям типов. В табл. 1 представлены варианты целого типа данных.

В табл. 2 приведены виды других типов данных без указания их диапазонов значений. Эту информацию можно найти на сайте [1].

Таблица 1

Тип	Диапазон
INT	$\pm 2^{31}$: 4 байта
INT UNSIGNED	0 ÷ 4294967295: 4 байта
INTEGER	не ограничен
TINYINT	-128 ÷ 127: 1 байт
TINYINT UNSIGNED	0 ÷ 255: 1 байт
SMALLINT	-32 768 ÷ 32 767: 2 байта
SMALLINT UNSIGNED	0 ÷ 65535: 2 байта
MEDIUMINT	-8388608 ÷ 8388607: 3 байта
MEDIUMINT UNSIGNED	0 ÷ 16777215: 3 байта
BIGINT	$\pm 2^{63:}$ 8 байт
BIGINT UNSIGNED	0 ÷ 184467440737095516145: 8 байт

SQLite не имеет отдельного логического класса хранения. Вместо этого, логические значения хранятся как целые числа 0 (false) и 1 (true), т.е. тип BOOLEAN является разновидностью целого типа.

SQLite не имеют классов, предназначенных для хранения дат и/или времени. Вместо этого встроенные функции даты и времени в SQLite способны работать с датами и временем, сохраненными в виде значений TEXT, REAL и INTEGER. В приложениях следует выбирать, в каком из указанных в табл. 2 форматов хранить даты и время, а затем можно свободно конвертировать из одного формата в другой с помощью встроенных функций даты и времени.

Для полей таблицы также можно задавать следующие ограничения:

NOT NULL – запрет на пустые значения. По умолчанию столбец может содержать значения NULL. Такое поле является полем без значения, принадлежащего одному из указанных ранее типов, т.е. значение NULL отличается от нулевого значения или поля, содержащего пробелы.

Если не желательно, чтобы столбец имел значение NULL, нужно определить ограничение, указав, что значение NULL теперь не разрешено для этого столбца. Пример:

NAME TEXT NOT NULL

Таблица 2

Тип	Класс
REAL	Вещественный
DOUBLE	
DOUBLE PRECISION	
FLOAT	
FIXED	
NUMERIC	
DECIMAL	
DEC	
BOOLEAN	Логический
DATE	Дата/время
TIME	
DATETIME	
TIMESTAMP	
YEAR	
CHAR	Символьный
VARCHAR	
TINYTEXT	
MEDIUMTEXT	
LARGETEXT)	
TEXT	
CLOB	
BLOB	Бинарный

DEFAULT – значение по умолчанию для столбца, когда при заполнении таблицы соответствующее поле в этом столбце будет пропущено. Пример:

SALARY REAL DEFAULT 50000.00

UNIQUE – ограничение, не позволяющее двум записям иметь одинаковые значения в определенном столбце. Например, можно запретить одинаковый возраст двух или более человек.

AGE INT NOT NULL UNIQUE

PRIMARY KEY - это поле в таблице, которое однозначно идентифицирует каждую строку/запись в таблице базы данных. Первичные ключи должны содержать уникальные значения. У столбца первичного ключа не может быть значений NULL. Могут быть несколько столбцов с ограничением UNIQUE, но только один из них может быть первичным ключом в таблице. В таблице может быть только один первичный ключ, который может состоять из одного или нескольких полей. Когда в качестве первичного ключа используются несколько полей, то ключ называется **составным**. Пример простого ключа:

ID INT PRIMARY KEY NOT NULL

Первичные ключи становятся внешними ключами в других таблицах при создании отношений между таблицами.

CHECK – ограничение, которое дает возможность проверить значение, введенное в запись. Если условие принимает значение false, запись нарушает ограничение и не вводится в таблицу.

SALARY REAL CHECK (SALARY > 0)

При этом в логическом выражении, задающем ограничение, могут использоваться логические операции &&, ||, !.

AUTOINCREMENT - это ключевое слово, используемое для автоматического увеличения значения поля в таблице. Это слово может использоваться только для полей с классом INTEGER.

column1 INTEGER **AUTOINCREMENT,**

CHAR(30) – ограничение длины строки для текстовых полей.

SQLite 3 имеет обычный набор операторов SQL для выражений сравнения, в том числе "=", "==", "<", "<=", ">", ">=", "!=", "<>", "BETWEEN", "IN", "IS", "NOT IN" и "IS NOT".

3. Введение в язык SQL

Практически все манипулирование базой данных ведется не через визуальный интерфейс, как например в СУБД MS Access, а посредством командного языка SQL (Structured Query Language - языка структурированных запросов).

Стандартные команды SQLite для взаимодействия с реляционными базами данных на языке SQL: CREATE, SELECT, INSERT, UPDATE, DELETE и DROP (табл. 3).

Таблица 3

Название команды	**Описание**
CREATE	Создание новой таблицы, представление таблицы или другой объект в базе данных
ALTER	Изменяет существующий объект базы данных, такой как таблица
DROP	Удаляет всю таблицу, представление таблицы или другого объекта в базе данных
INSERT	Создает запись
UPDATE	Изменяет записи
DELETE	Удаляет записи
SELECT	Извлекает определенные записи из одной или нескольких таблиц

4. Создание базы данных

Для загрузки библиотеки СУБД в Python-приложение нужно использовать следующую команду:

```
import sqlite3
```

Есть несколько способов создания базы данных в Python с помощью СУБД SQLite. Для этого используется объект Connection, который и представляет собой базу. Он создается с помощью функции connect().

В данной СУБД база данных создается как один файл с расширением .db. Пусть файл называется group.db. За соединение будет отвечать переменная conn.

```
conn = sqlite3.connect(' group.db')
```

Эта строка создает объект connection, а также новый файл group.db в рабочей директории языка Python. Если нужно использовать другую директорию, то ее нужно обозначить явно:

```
conn = sqlite3.connect(r'ПУТЬ-К-ПАПКЕ/ group.db')
```

(Примечание: *Перед строкой с путем стоит символ «r». Это дает понять Python, что речь идет о «сырой» строке, где символы «/» не отвечают за экранирование.)*

Если файл уже существует, то функция connect осуществит просто подключение к нему.

Таким образом, функция connect создает соединение с базой данных SQLite и возвращает объект, представляющий ее. Этот объект является логическим именем БД. С его помощью можно подсоединяться к разным физическим БД, не меняя алгоритмы и листинг программы обработки информации.

После создания объекта соединения с БД нужно создать объект cursor. Он позволяет делать SQL-запросы к базе. Используем переменную cur для хранения объекта:

```
cur = conn.cursor()
```

Теперь выполнять запросы можно следующим образом:

```
cur.execute("ВАШ-SQL-ЗАПРОС-ЗДЕСЬ;")
```

Кавычки могут быть одинарные, двойные или тройные. Последние используются в случае особенно длинных запросов, которые часто пишутся на нескольких строках.

Создадим две таблицы со следующей структурой (рис. 3):

STUDENT		
Имя	*Тип*	*Ограничения*
IDS IDK (Номер)	INT	>0
FIO	TEXT	20
POL	TEXT	1
GROUP	TEXT	6
KURS	INT	1
KAF(Аббревиатура)	TEXT	6

KAFEDRA		
Имя	*Тип*	*Ограничения*
IDK (Аббревиатура)	TEXT	6
NAME	TEXT	40
KABINET	INT	>0
ZAVK	TEXT	20
PHONE	INT	999999

Рис. 3

Начнем с таблицы **STUDENT**:

```
cur.execute("""CREATE TABLE IF NOT EXISTS STUDENT (
  IDS INT PRIMARY KEY, #уникальный номер студента - ключ
  FIO TEXT, #ФИО студента
  POL TEXT, #пол студента
  GROUP TEXT, #группа студента
  KURS INT, #курс студента
  KAF TEXT); #название кафедры студента
""")
  conn.commit()
```

В коде выше выполняются следующие операции:

1. Функция execute отвечает за SQL-запрос
2. SQL генерирует таблицу STUDENT
3. create проверит, существует ли таблица. Если да — проверит, ничего ли не поменялось?
4. Создаются первые шесть колонок: IDS, FIO, POL, GROUP, KURS и KAF. IDS — это основной ключ.
5. Сохраняются изменения с помощью функции commit для объекта соединения. commit - это транзакционная команда, используемая для сохранения изменений, вызванных транзакцией, в базу данных.

Для создания второй таблицы просто повторим последовательность действий, используя следующие команды:

```
cur.execute("""CREATE TABLE IF NOT EXISTS KAFEDRA (
  IDK INT PRIMARY KEY, #уникальный номер кафедры - ключ
  NAME TEXT, #название кафедры
  KABINET INT, #номер кабинета заведующего кафедрой
  ZAVK TEXT, #ФИО заведующего кафедрой
  PHONE INT); #телефон кафедры
""")
conn.commit()
```

После завершения работы с таблицей необходимо все используемые объекты удалить из программы:

```
cur.close()
conn.close()
```

5. Добавление информации в базу данных

По аналогии с запросом для создания таблиц для добавления данных также нужно использовать объект cursor.

```
cur.execute("""INSERT INTO STUDENT(IDS, FIO, POL, GROUP, KURS, KAF)
  VALUES('01', 'Грачев В.П.', 'М', '19-ВТ', '2', 'СУиВТ');""")
conn.commit()
```

В этом случае добавляется только одна запись. При этом ее можно предварительно оформлять в виде кортежа, а затем этот кортеж загружать в БД. Например:

```
CT = ('02','Сучилов Р.А.', 'М', '18-АП', '3', 'АПП')
cur.execute("INSERT INTO STUDENT VALUES(?, ?, ?, ?,?,?);", CT)
conn.commit()
```

В данном случае все значения заменены на знаки вопроса и добавлен компонент, содержащий все значения, которые нужно добавить.

В переменной может быть и список с набором кортежей. Таким образом можно добавить сразу несколько пользователей:

```
more_stud = [('03','Щукин В.А.', 'М', '20-АП', '1', 'АПП'),
             ('04','Саблина Р.Ф.', 'Ж', '18-МС', '3', 'АМС')]
```

Но в этом случае нужно использовать функцию executemany вместо обычной execute:

```
cur.executemany("INSERT INTO STUDENT VALUES(?, ?, ?, ?,?,?);", more_stud)
conn.commit()
```

6. Удаление данных из БД

Пример удаления конкретной записи по равенству значению (в данном случае это студент с фамилией «Грачев»):

```
cur.execute("DELETE FROM STUDENT WHERE NAME =' Грачев В.П.';")
conn.commit()
```

Если затем сделать следующей запрос:

```
cur.execute("select * FROM STUDENT WHERE NAME =' Грачев В.П.';")
print(cur.fetchall())
```

С помощью функции fetchall() будет выведен список, подтверждающий, что запись удалена. Для вывода также можно использовать функции fetchone() и fetchmany().

7. Редактирование (обновление) записей

Если нельзя редактировать данные в базе, тогда она достаточно скоро станет бесполезной. Простейший пример обновления одного поля:

```
sql = """UPDATE STUDENT SET FIO = 'Васнецова В.В.' WHERE ids = '45';"""
cur.execute(sql)
conn.commit()
```

Для обновления используется запрос SQL под названием UPDATE. В этом случае будет обновлено только одно поле у записи с конкретным ключом, т.е эта запись готовится заранее, до запуска программы. Если же обновляемые данные определяются в ходе работы программы, то требуется сформировать запрос с параметрами (параметризованный запрос). Для этого используют заполнители (?) прямо внутри инструкции SQL. Общая структура такого запроса имеет вид:

```
<имя запроса> = """Update <имя таблицы> set [{<имя поля> =?,}] <имя поля>=?,}] where <ключ>=?"""
<данные> = ({<значение>,} <значение ключа>)
cursor.execute(<имя запроса>, <данные>)
```

Пример:

```
strsam = """UPDATE STUDENT SET FIO = ?, GRUP = ? WHERE ids = ?;"""
data = (e2.get(), e3.get(), e1.get())
cur.execute(strsam, data)
```

В этом примере видно, что два новых значения для модифицируемых полей и ключ для модифицируемой записи получены уже в ходе работы программы. Они были введены в соответствующие редакторы.

Можно ли таким образом обновить и само ключевое поле? Оказывается, нельзя.

Иногда в приложениях Python нужно обновить несколько строк. Например, нужно увеличить стипендию студентам на 20%. Вместе выполнения операции UPDATE для каждой записи можно выполнить массовое обновление в один запрос. Это можно с помощью метода cursor.executemany(query, seq_param).

У него два аргумента: SQL-запрос (query) и список записей для обновления (seq_param). Пример:

```
record_list = [(3, 4), (2, 1), (3, 6)]
update_query = """Update STUDENT set KURS = ? where ids = ?;"""
cur.executemany(update_query, record_list)
```

В данном случае происходит изменение курса для студентов, у которых номера 1, 4 и 6. Т.е. формируется множество двоек, в которых указывается новое значение для поля, заданного в операторе обновления, а затем идентификатор обновляемого поля. Можно свести первый и третий операторы к одному:

```
cur.executemany(update_query, [(1, 4), (1, 5), (1, 6)])
```

Рассмотрим два примера сквозной (по всей таблице) модификации данных. В первом случае предполагается увеличить зарплату на 30 % тем работникам, у кого она ниже 25000.

```
update_query = """UPDATE job SET price = price * 1,3 WHERE price <= 25000;"""
```

Во втором случае зарплата увеличивается на 3000 тем, кто моложе 30 лет и у кого зарплата меньше 40000.

```
update_query = """UPDATE table1 SET sal = (sal + 3000) WHERE age < 30 AND sal < 40000;"""
```

8. Выборка данных из БД

Основное назначение баз данных – своевременное обеспечение пользователей хранящейся в них информацией. Выборка информации из БД по определенному критерию - самая распространенная операция, применяемая к БД. Реализуется данная операция посредством оператора выбора SELECT языка SQL, формируемого по следующему синтаксису:

```
SELECT <Список выбора>
FROM <Источник выбора>
[WHERE <Условия отбора>]
```

[GROUP BY <Виды группировки>]

В квадратных скобках указаны необязательные элементы запроса

Раздел SELECT запроса задает набор полей, которые должны выводиться в итоговой таблице. Здесь могут просто перечисляться названия полей исходной таблицы и/или результаты вычислений выражений, в которые входят названия полей таблицы. Если вместо списка указывается метасимвол *, то выводятся все поля подряд.

В разделе FROM перечисляются таблицы, из которых производится выборка данных.

Раздел WHERE используется для наложения вертикальных фильтров на данные, обрабатываемые запросом. Фильтр обычно представляет логическое выражение и срабатывает, когда выражение принимает значение TRUE. В логических выражениях можно использовать операции сравнения (=, >, <, >=, <=), логические операции (OR, AND, NOT) и круглые скобки. Для выборки записей с отсутствующими значениями по какому-то полю необходимо сравнивать значения в этом поле с константой NULL

Раздел GROUP BY позволяет выполнить группировку записей по определенным критериям. При этом к каждой группе можно применять операции агрегирования, например, нахождение суммы по группе или среднего значения. В простейшем случае в данном разделе перечисляются названия полей, по которым производится группировка.

Рассмотрим несколько примеров выборки данных из таблиц. Начнем с использования функции fetchone(). Создадим переменную one_result для получения только одного результата:

```
cur.execute("SELECT * FROM STUDENT;")
one_result = cur.fetchone()
print(one_result)
```

Она вернет следующий кортеж:

```
[(1, ‘Грачев В.П.', 'М', '19-ВТ', 2, 'СУиВТ’)]
```

Если же нужно получить много данных, то используется функция fetchmany(). Ниже приведен скрипт для генерации 3 результатов:

```
cur.execute("SELECT * FROM STUDENT;")
three_results = cur.fetchmany(3)
```

```
print(three_results)
```

В переменной three_results появится кортеж из трех первых записей таблицы.

Функцию fetchall() можно использовать для скачивания в одну переменную содержимого всей таблицы:

```
cur.execute("SELECT * FROM STUDENT;")
all_results = cur.fetchall()
print(all_results)
```

Запросы выборки также можно оформлять в виде строки, а затем подставлять в виде аргумента в оператор работы с БД:

```
query = 'SELECT * FROM ' + table + ' GROUP BY KURS;'
cur.execute(query)
```
(1)

Запросы выборки тоже можно параметризовывать, т.е менять значения для выборки в ходе работы программы. Исходно название кафедры жестко задано:

```
cur.execute("SELECT * FROM STUDENT WHERE KAF='АПП';")
tresult = cur.fetchall()
```

Но его можно менять по ходу работы:

```
ss = ew1.get()
cur.execute('SELECT * FROM STUDENT WHERE KAF=(?);', (ss,))
tresult = cur.fetchall()
```
(2)

В данном случае название кафедры предварительно вводится в редактор ew1, а затем добавляется в тест запроса. Причем здесь обязательна запятая после текущего значения названия кафедры. Можно еще короче:

```
cur.execute('SELECT * FROM STUDENT WHERE KAF=(?);', (ew1.get(),))
tresult = cur.fetchall()
```

9. Выборка из связанных таблиц

Для более сложных запросов из нескольких таблиц последние надо объединить. Для этого используется оператор **JOINS**.

SQL определяет три основных типа объединений:

- CROSS JOIN
- INNER JOIN
- OUTER JOIN

CROSS JOIN - сопоставляет каждую строку первой таблицы с каждой строкой второй таблицы. Если входные таблицы имеют строки x и y, соответственно, результирующая таблица будет иметь строку x * y., т. е. если размер 1-й таблицы N, а второй -M, то размер итоговой таблицы будет N * M (каждый к каждому).

При этом в списке для вывода указываются поля как 1-й, так и 2-й таблиц.

INNER JOIN - создает новую таблицу результатов, комбинируя значения столбцов двух таблиц (table1 и table2) на основе предиката соединения. Запрос сравнивает каждую строку таблицы1 с каждой строкой таблицы2, чтобы найти все пары строк, которые удовлетворяют предикату соединения. Когда предикат соединения выполняется, значения столбца для каждой парной пары строк из A и B объединяются в строку результатов.

INNER JOIN - наиболее распространенный тип соединения по умолчанию.

SELECT **...** FROM **table1 [**INNER**]** JOIN **table2** ON **conditional_expression ...**

Например, выводим фамилии зав.кафедрой для тех студентов, у которых кафедрf описаны в таблице KAFEDRA:

```
cur.execute("""SELECT FIO, KAFEDRA.ZAVK FROM STUDENT INNER JOIN
KAFEDRA ON STUDENT.KAF=KAFEDRA.IDK;""")                    (3)

tresult = cur.fetchall()

con.commit()

cur.close()

tk = len(tresult)

for i in range(0, tk)

        print(ttm = str(tresult[i][0])+str(tresult[i][1]))
```

OUTER JOIN - это расширение INNER JOIN. Хотя стандарт SQL определяет три типа OUTER JOIN: LEFT, RIGHT и FULL, SQLite поддерживает только LEFT OUTER JOIN.

Задания на практику

1. В соответствии с вариантом из табл. 4 выбирается предметная область будущей БД.
2. На основе заданной предметной области разрабатывается структура двух таблиц, аналогично рис. 3. В 1-й таблице должно быть не менее 4-х полей, во 2-й – не менее 3-х полей. В обеих таблицах обязательны ключевые поля, причем ключевое поле 2-й таблицы должно совпадать с одним из неключевых полей 1-й таблицы.

   В 1-й таблице минимум одно поле должно быть числовым, одно текстовым, одно типа даты и/или времени. Для полей, где это возможно, должны быть установлены ограничения.
3. Для 1-й таблицы разрабатывается программный интерфейс, предполагающий четыре режима доступа: просмотр, дополнение, модификация, удаление. Предусмотрен повторный запрос на подтверждение удаления. Режим задается компонентом, указанным для варианта в табл. 4.
4. 1-я таблица заполняется не менее чем 20 записями, 2-я – не менее трех любым способом.
5. Создаются три запроса выбора для БД:

   а) параметризованный запрос типа (2);

   б) запрос с группировкой по одному из неключевых полей типа (1);

   в) сложный запрос по двум таблицам типа (3).

   Запуск конкретного запроса производится виджетом, указанным в соответствии с вариантом в табл. 4.

Таблица 4

Номер варианта	**Предметная область**	**Переключение режимов работы с таблицей**	**Выбор вида запроса**
1	Речные рыбы	Главное меню	Главное меню
2	Морские рыбы	Контекстное меню	Главное меню
3	Предприятия рыбной промышленности	Ниспадающее меню	Главное меню
4	Рыболовные суда	Набор кнопок	Главное меню
5	Рыбные блюда	Радионабор	Главное меню
6	Орудия рыбной ловли	Listbox	Главное меню
7	Рыбная продукция	Combobox	Главное меню
8	Сотрудники вуза	Главное меню	Контекстное меню
9	Студенты	Контекстное меню	Контекстное меню

Продолжение табл. 4

10	Учебные курсы	Ниспадающее меню	Контекстное меню
11	Помещения вуза	Набор кнопок	Контекстное меню
12	Принтеры	Радионабор	Контекстное меню
13	Фирмы	Listbox	Контекстное меню
14	Программные пакеты	Combobox	Контекстное меню
15	Города	Главное меню	Ниспадающее меню
16	Водоемы	Контекстное меню	Ниспадающее меню
17	Медицинские учреждения	Ниспадающее меню	Ниспадающее меню
18	Страны мира	Набор кнопок	Ниспадающее меню
19	Учебные заведения	Радионабор	Ниспадающее меню
20	Плавательные средства	Listbox	Ниспадающее меню
21	Птицы	Combobox	Ниспадающее меню
22	Одежда	Главное меню	Набор кнопок
23	Виды спорта	Контекстное меню	Набор кнопок
24	Бытовые электроприборы	Ниспадающее меню	Набор кнопок
25	Магазины	Набор кнопок	Набор кнопок
26	Небесные тела	Радионабор	Набор кнопок
27	Детские игрушки	Listbox	Набор кнопок
28	Развлекательные заведения	Combobox	Набор кнопок
29	Места отдыха	Главное меню	Радионабор
30	Игры с мячом	Контекстное меню	Радионабор
31	Животный мир	Ниспадающее меню	Радионабор
32	Улицы города	Набор кнопок	Радионабор
33	Автомобили	Радионабор	Радионабор
34	Летательные аппараты	Listbox	Радионабор
35	Средства вычислительной техники	Combobox	Радионабор
36	Периодические издания	Главное меню	Listbox
37	Книги	Контекстное меню	Listbox
38	Сайты	Ниспадающее меню	Listbox
39	Породы собак	Набор кнопок	Listbox
40	Породы кошек	Радионабор	Listbox
41	Водоплавающие птицы	Listbox	Listbox
42	Рыбы	Combobox	Listbox
43	Произведения живописи	Главное меню	Combobox
44	Печатная продукция	Контекстное меню	Combobox
45	Животный мир	Ниспадающее меню	Combobox
46	Речные рыбы	Главное меню	Главное меню
47	Морские рыбы	Контекстное меню	Главное меню
48	Рыбная продукция	Combobox	Главное меню
49	Студенты	Контекстное меню	Контекстное меню
50	Города	Главное меню	Ниспадающее меню

Окончание табл. 4

51	Медицинские учреждения	Ниспадающее меню	Ниспадающее меню
52	Страны мира	Набор кнопок	Ниспадающее меню
53	Плавательные средства	Listbox	Ниспадающее меню
54	Бытовые электроприборы	Ниспадающее меню	Набор кнопок
55	Магазины	Набор кнопок	Набор кнопок
56	Детские игрушки	Listbox	Набор кнопок
57	Развлекательные заведения	Combobox	Набор кнопок
58	Игры с мячом	Контекстное меню	Радионабор
59	Животный мир	Ниспадающее меню	Радионабор
60	Улицы города	Набор кнопок	Радионабор
61	Игры с мячом	Контекстное меню	Радионабор

6. Реализованный проект демонстрируется руководителю практики. При его положительном решении создается отчет в соответствии с требованиями, изложенными ниже.
7. Отчет защищается практикантом с оценкой по пятибалльной системе.

Содержание отчета:

1) Вариант задания на выполнение проекта;
2) Структура проекта;
3) Структуры таблиц в соответствии с рис. 3;
4) Скриншоты обеих заполненных таблиц
5) Тексты всех трех запросов со скриншотами результатов их выполнения;
6) Листинг программы, максимально снабженный комментариями;
7) Список использованной литературы.

Правила оформления отчета по практике

Текстовые документы оформляются на листах формата А4, графический материал допускается представлять на листах формата А3.

Поля на листе: левое – не менее 30 мм, правое – не менее 10, верхнее – не менее 15, нижнее – не менее 20 мм.

Нумерация страниц сквозная. Номера проставляются сверху справа арабской цифрой. Первой страницей считается титульный лист, на котором номер страницы не проставляется.

Шрифт основных заголовков – 14, полужирный, подзаголовков – 13,5. Шрифт основного текста – не менее 12 пунктов (включая рисунки и диаграммы).

На все рисунки, таблицы и формулы в документе должны быть ссылки в виде: «рис. 23» или «блок- схема данного модуля приведена на рис. 31». Рисунки не должны опережать текст, в котором находится ссылка на них.

Образец титульного листа отчета представлен в Приложении.

Литература:

1. https://pythonru.com/osnovy/sqlite-v-python (дата обращения: 16.05.2021).
2. https://www.severcart.ru/blog/all/python_sqlite3/ (дата обращения: 13.02.2021).
3. https://pyneng.readthedocs.io/ru/latest/book/25_db/sqlite3.html (дата обращения: 07.09.2021).
4. https://python-scripts.com/sqlite (дата обращения: 24.06.2021).
5. https://digitology.tech/docs/python_3/library/sqlite3.html (дата обращения: 22.08.2021).

ПРИЛОЖЕНИЕ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО РЫБОЛОВСТВУ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Калининградский государственный технический университет»

Институт цифровых технологий

Кафедра прикладной информатики
наименование кафедры

ОТЧЕТ
по эксплуатационной практике

База практики: _____
(наименование предприятия)

Выполнил студент группы _____,

Форма обучения _____, курс _____

_____ _____
Подпись *(И.О. Фамилия)*

Руководитель практики от предприятия _____ _____
(при наличии) *Подпись* *(И.О. Фамилия)*

Руководитель практики от кафедры _____ _____
_____*Сокращенное наименование кафедры* *Подпись* *(И.О. Фамилия)*

Отчет защищен с оценкой _____

Дата защиты отчета _____

Локальный электронный методический материал

Леонид Григорьевич Высоцкий

ЭКСПЛУАТАЦИОННАЯ ПРАКТИКА

Редактор Г. А. Смирнова

Уч.-изд. л. 1,75. Печ. л. 1,3

Издательство федерального государственного бюджетного образовательного учреждения высшего образования
«Калининградский государственный технический университет».
236022, Калининград, Советский проспект, 1