ИССЛЕДОВАНИЕ И УПРАВЛЕНИЕ БАЗОЙ ДАННЫХ ДЛЯ КОФЕЙНИ С ПОМОЩЬЮ PYTHON И SQLITE.

Введение. В мире современного программирования создание информационных систем, обеспечивающих эффективное управление бизнес-процессами, является важным этапом разработки. В данной статье мы рассмотрим использование базы данных SQLite в контексте кофейни и рассмотрим пример Python-скрипта, который управляет созданием и заполнением базы данных, может выполнять запросы.

Основная часть. Необходимо разработать базу данных для работы с кофейнями и рассмотреть разные сценария её применения. Задачами лабораторной работы являются создание таблиц при помощи SQLite, заполнение их данными, формулировка некоторых запросов к базе данных и дальнейшая интеграция Python в взаимодействие с базой данных. Основной целью лабораторной работы является изучение и отработка навыков работы с базами данных с использованием SQL на примере создания, наполнения и анализа базы данных кофейни. Все языки SQL имеют много общего, но SQLite самый понятный и распространенный. Кроме того, SQLite является компактной, встроенной СУБД, идеальной для небольших проектов. В контексте кофейни, где база данных скорее всего не будет огромной, SQLite предоставляет простой и эффективный способ хранения данных. Также можно отметить, что лабораторная работа требует выполнять взаимодействие с базой данных из консоли при помощи интеграции Python, а SQLite подходит под эту задачу несколько лучше, чем другие языки. Выполнять интеграцию взаимодействия через другой язык программирования в рамках этой лабораторной важно, так как при повседневном использовании базы данных не используются сами по себе, и простейший навык их “подключения” оказывается действительно необходимым.

В качестве технических требований к лабораторной работе можно предъявить следующие пункты:

— Целостность базы данных,

— Простота обновления данных,

— Присутствие компилятора для базы данных.

В рамках выполнения работы будет требоваться отчет, в котором необходимо представить листинг кода создания таблиц, наполнения данными и запросами к таблицам. В ходе всей работы должны присутствовать промежуточные комментариями и выводы.

Ниже приведено выполнение лабораторной работы.

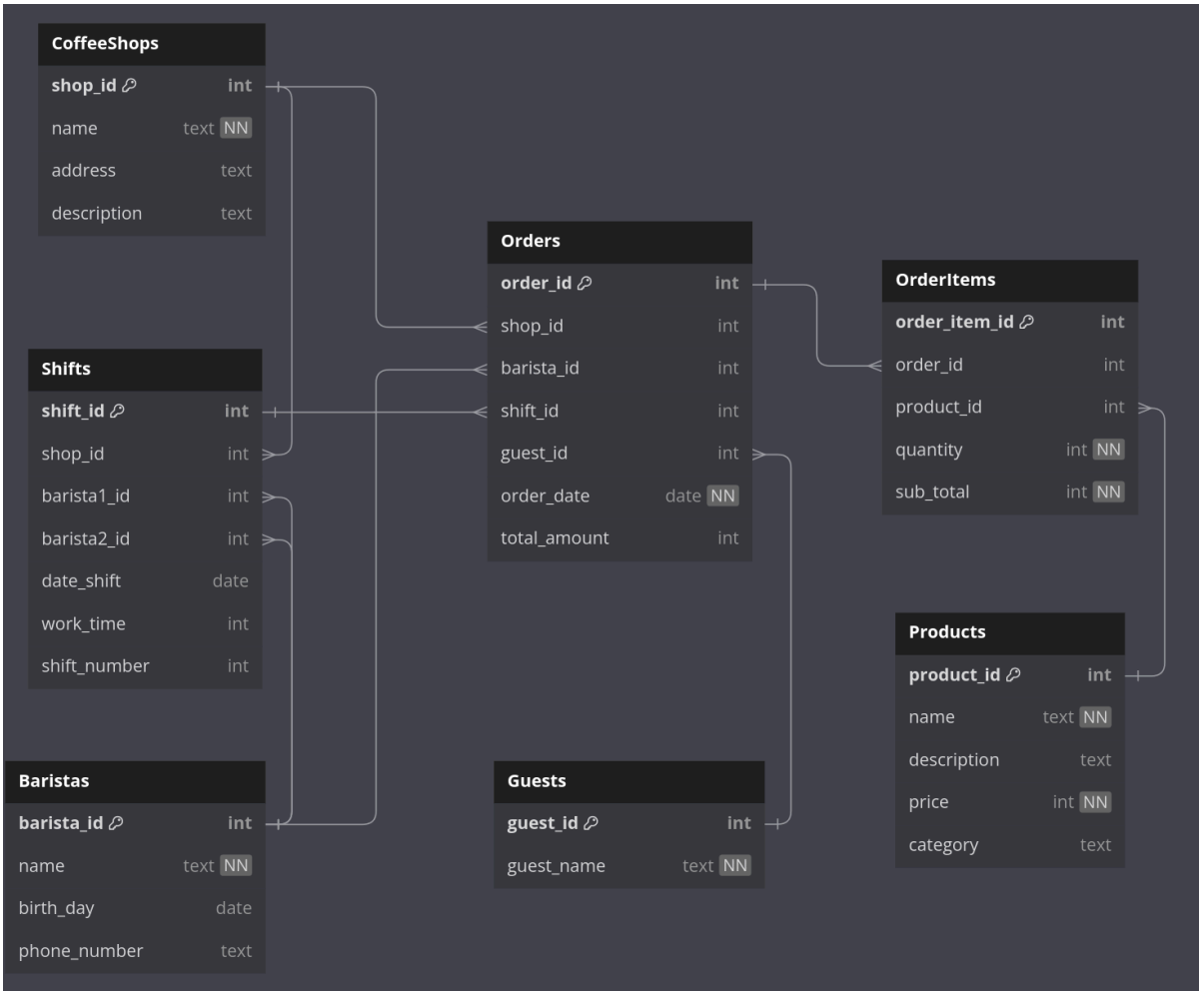
База данных SQLite для кофейни

SQLite поддерживает стандартный SQL, что делает проектирование базы данных простым и интуитивно понятным. Давайте создадим таблицы для нашей кофейни, используя внешние ключи для определения связей между таблицами:

```
CREATE TABLE CoffeeShops (
    shop_id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    address TEXT,
    description TEXT
);

CREATE TABLE Guests (
    guest_id INTEGER PRIMARY KEY,
    guest_name TEXT NOT NULL
);

CREATE TABLE Baristas (
    barista_id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    birth_day DATE,
    phone_number TEXT
);

CREATE TABLE Shifts (
    shift_id INTEGER PRIMARY KEY,
    shop_id INTEGER,
    barista1_id INTEGER,
    barista2_id INTEGER,
    date_shift DATE,
    work_time INTEGER,
    shift_number INTEGER,
    FOREIGN KEY (shop_id) REFERENCES CoffeeShops(shop_id),
    FOREIGN KEY (barista1_id) REFERENCES Baristas(barista_id),
    FOREIGN KEY (barista2_id) REFERENCES Baristas(barista_id)
);

CREATE TABLE Products (
    product_id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    description TEXT,
    price INTEGER NOT NULL,
```

```
    category TEXT
);


CREATE TABLE Orders (
    order_id INTEGER PRIMARY KEY,
    shop_id INTEGER,
        barista_id INTEGER,
        shift_id INTEGER,
    guest_id INTEGER,
    order_date DATETIME NOT NULL,
    total_amount INTEGER CHECK (total_amount >= 0),
        FOREIGN KEY (barista_id) REFERENCES Baristas(barista_id),
        FOREIGN KEY (shift_id) REFERENCES Shifts(shift_id),
    FOREIGN KEY (guest_id) REFERENCES Guests(guest_id),
    FOREIGN KEY (shop_id) REFERENCES CoffeeShops(shop_id)
);

CREATE TABLE OrderItems (
    order_item_id INTEGER PRIMARY KEY,
    order_id INTEGER,
    product_id INTEGER,
    quantity INTEGER NOT NULL,
    sub_total INTEGER NOT NULL,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id),
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

Ниже можно увидеть схему базы данных, созданную скриптом ранее.



Ниже приведен код наполнения таблицы данными.

```
INSERT INTO CoffeeShops (shop_id, name, address, description)
VALUES
  (1, 'Кафе "Ароматное утро"', 'ул. Цветная, 1', 'Уютное место для начала дня'),
  (2, 'Кофейня "Эспрессо Лайф"', 'пр. Солнечный, 15', 'Лучшие сорта эспрессо в
городе'),
  (3, 'Кофехаус "Зеленый чайник"', 'ул. Гармония, 8', 'Кофе с любовью к каждому
гостю'),
  (4, 'Кафетерий "Путешествие во вкус"', 'пл. Центральная, 7', 'Открой для себя вкусы
мира'),
  (5, 'Кофе-бар "Лунный свет"', 'пер. Тихий, 3', 'Атмосфера мистики и волшебства');


INSERT INTO Baristas (barista_id, name, birth_day, phone_number)
VALUES
  (1, 'Иван Иванович', '1992-03-25', '+7 (111) 222-3333'),
```

```
  (2, 'Мария Петровна', '1988-08-10', '+7 (444) 555-6666'),
  (3, 'Алексей Сергеевич', '1995-02-05', '+7 (777) 888-9999'),
  (4, 'Елена Викторовна', '1991-11-30', '+7 (123) 456-7890'),
  (5, 'Дмитрий Николаевич', '1987-07-15', '+7 (987) 654-3210'),
  (6, 'Анна Алексеевна', '1993-04-18', '+7 (555) 123-4567'),
  (7, 'Сергей Игоревич', '1996-09-02', '+7 (999) 000-1111'),
  (8, 'Ксения Владимировна', '1990-12-08', '+7 (222) 333-4444'),
  (9, 'Павел Артемович', '1989-06-20', '+7 (666) 777-8888'),
  (10, 'Ольга Анатольевна', '1994-01-12', '+7 (888) 999-0000'),
  (11, 'Артем Станиславович', '1997-05-05', '+7 (111) 222-3333'),
  (12, 'Екатерина Дмитриевна', '1998-10-15', '+7 (444) 555-6666'),
  (13, 'Игорь Валентинович', '1992-11-05', '+7 (777) 888-9999'),
  (14, 'Наталья Леонидовна', '1993-08-30', '+7 (123) 456-7890'),
  (15, 'Александр Степанович', '1987-03-15', '+7 (987) 654-3210'),
  (16, 'Маргарита Валерьевна', '1995-07-25', '+7 (555) 123-4567'),
  (17, 'Владимир Павлович', '1991-02-15', '+7 (999) 000-1111'),
  (18, 'Татьяна Сергеевна', '1990-09-10', '+7 (222) 333-4444'),
  (19, 'Илья Викторович', '1994-12-08', '+7 (666) 777-8888'),
  (20, 'Ангелина Артемовна', '1988-06-20', '+7 (888) 999-0000');


INSERT INTO Shifts (shift_id, shop_id, barista1_id, barista2_id, date_shift, work_time,
shift_number)
VALUES
  (1, 1, 1, 2, '2023-11-11', 8, 1),
  (2, 1, 3, 4, '2023-11-11', 6, 2),
  (3, 2, 5, 6, '2023-11-11', 7, 1),
  (4, 2, 7, 8, '2023-11-11', 8, 2),
  (5, 3, 9, 10, '2023-11-11', 6, 1),
  (6, 3, 11, 12, '2023-11-11', 7, 2),
  (7, 4, 13, 14, '2023-11-11', 8, 1),
  (8, 4, 15, 16, '2023-11-11', 6, 2),
  (9, 5, 17, 18, '2023-11-11', 7, 1),
  (10, 5, 19, 20, '2023-11-11', 8, 2);



INSERT INTO Products (product_id, name, price, category)
VALUES
  (1, 'Капучино', 150, 'Напитки'),
  (2, 'Латте', 120, 'Напитки'),
  (3, 'Американо', 100, 'Напитки'),
  (4, 'Эспрессо', 80, 'Напитки'),
```

```
  (5, 'Мокко', 180, 'Напитки'),
  (6, 'Чай зеленый', 90, 'Напитки'),
  (7, 'Чай черный', 80, 'Напитки'),
  (8, 'Капкейк', 200, 'Десерты'),
  (9, 'Тирамису', 220, 'Десерты'),
  (10, 'Чизкейк', 250, 'Десерты'),
  (11, 'Панна котта', 180, 'Десерты'),
  (12, 'Шоколадный мусс', 190, 'Десерты'),
  (13, 'Фруктовый салат', 150, 'Десерты'),
  (14, 'Маффин', 120, 'Десерты'),
  (15, 'Фреш апельсиновый', 100, 'Напитки');


INSERT INTO Guests (guest_id, guest_name)
VALUES
  (1, 'Иванов Иван'),
  (2, 'Петрова Мария'),
  (3, 'Сидоров Алексей'),
  (4, 'Козлова Елена'),
  (5, 'Новиков Дмитрий'),
  (6, 'Лебедева Анна'),
  (7, 'Васнецов Сергей'),
  (8, 'Жукова Ксения'),
  (9, 'Игнатьев Павел'),
  (10, 'Федорова Ольга'),
  (11, 'Лисов Артем'),
  (12, 'Миронова Екатерина'),
  (13, 'Семенов Игорь'),
  (14, 'Александрова Наталья'),
  (15, 'Кузнецов Владимир'),
  (16, 'Михайлова Татьяна'),
  (17, 'Павлов Илья'),
  (18, 'Сергеева Ангелина'),
  (19, 'Карпов Валентин'),
  (20, 'Григорьева Маргарита');


INSERT INTO Orders (order_id, shop_id, barista_id, shift_id, guest_id, order_date,
total_amount)
VALUES
        (1, 1, 1, 1, 1, '2023-11-11 08:30', 670),
        (2, 1, 2, 1, 2, '2023-11-11 09:15', 100),
        (3, 1, 1, 1, 3, '2023-11-11 10:00', 580),
```

```
    (4, 1, 3, 2, 4, '2023-11-11 11:30', 180),
    (5, 1, 4, 2, 5, '2023-11-11 12:15', 160),
    (6, 2, 5, 3, 6, '2023-11-11 13:00', 100),
    (7, 2, 5, 3, 7, '2023-11-11 14:30', 120),
    (8, 2, 5, 3, 8, '2023-11-11 15:15', 150),
    (9, 2, 7, 4, 9, '2023-11-11 16:00', 80),
    (10, 3, 9, 5, 10, '2023-11-11 17:30', 100),
    (11, 3, 10, 5, 11, '2023-11-11 18:15', 300),
    (12, 3, 11, 6, 12, '2023-11-11 19:00', 370),
    (13, 3, 11, 6, 1, '2023-11-11 20:30', 1010),
    (14, 4, 14, 7, 14, '2023-11-11 21:15', 450),
    (15, 4, 13, 7, 5, '2023-11-11 22:00', 280),
    (16, 4, 16, 8, 5, '2023-11-11 08:30', 1060),
    (17, 4, 16, 8, 17, '2023-11-11 09:15', 300),
    (18, 4, 16, 8, 2, '2023-11-11 10:00', 100),
    (19, 4, 15, 8, 19, '2023-11-11 11:30', 80),
    (20, 4, 16, 8, 20, '2023-11-11 12:15', 270);


INSERT INTO OrderItems (order_item_id, order_id, product_id, quantity, sub_total)
VALUES
    (1, 1, 2, 1, 120),
    (2, 1, 1, 2, 300),
    (3, 1, 10, 1, 250),
    (4, 2, 15, 1, 100),
    (5, 3, 12, 1, 190),
    (6, 3, 13, 1, 150),
    (7, 3, 14, 2, 240),
    (8, 4, 15, 1, 100),
    (9, 4, 7, 1, 80),
    (10, 5, 4, 2, 160),
    (11, 6, 3, 1, 100),
    (12, 7, 2, 1, 120),
    (13, 8, 1, 1, 150),
    (14, 9, 4, 1, 80),
    (15, 10, 3, 1, 100),
    (16, 11, 2, 1, 120),
    (17, 11, 5, 1, 180),
    (18, 12, 10, 1, 250),
    (19, 12, 2, 1, 120),
    (20, 13, 9, 3, 660),
    (21, 13, 8, 1, 200),
    (22, 13, 13, 1, 150),
```

```
        (23, 14, 12, 1, 190),
        (24, 14, 5, 1, 180),
        (25, 14, 7, 1, 80),
        (26, 15, 7, 1, 80),
        (27, 15, 8, 1, 200),
        (28, 16, 1, 4, 600),
        (29, 16, 12, 1, 190),
        (30, 16, 13, 1, 150),
        (31, 16, 14, 1, 120),
        (32, 17, 15, 3, 300),
        (33, 18, 3, 1, 100),
        (34, 19, 4, 1, 80),
        (35, 20, 5, 1, 180),
        (36, 20, 6, 1, 90);
```

Как можно заметить в нашей бд для надёжности есть повторяющиеся столбцы, но не все данные можно указывать явно. Например, в таблицы можно добавлять только состав заказа, а сумму подсчитает скрипт.

Ниже представлен код запроса, подсчитывающего сумму заказа.

```
UPDATE OrderItems
SET sub_total = (
        SELECT quantity * price
        FROM Products p
        WHERE OrderItems.product_id = p.product_id
)
WHERE EXISTS (
        SELECT 1
        FROM Products p
        WHERE OrderItems.product_id = p.product_id
);

UPDATE Orders
SET total_amount = (
        SELECT COALESCE(SUM(sub_total), 0)
        FROM OrderItems
        WHERE OrderItems.order_id = Orders.order_id
)
WHERE EXISTS (
        SELECT 1
```

```
        FROM OrderItems
        WHERE OrderItems.order_id = Orders.order_id
);
```

Далее представлены примеры обычных запросов.

```
-- сколько заказов сделали клиенты
SELECT Guests.guest_name, COUNT(Orders.order_id) AS total_orders
FROM Guests
LEFT JOIN Orders ON Guests.guest_id = Orders.guest_id
GROUP BY Guests.guest_name;


-- топ 5 бариста по сумме обработанных заказов
SELECT
  b.barista_id,
  b.name AS barista_name,
  SUM(o.total_amount) AS total_amount
FROM
  Baristas b
JOIN
  Orders o ON b.barista_id = o.barista_id
GROUP BY
  b.barista_id, b.name
ORDER BY
  total_amount DESC
LIMIT 5;

--средний чек в каждой кофейне
SELECT
  cs.name AS coffee_shop,
  COUNT(o.order_id) AS total_orders,
  AVG(o.total_amount) AS average_amount
FROM
  CoffeeShops cs
LEFT JOIN
  Orders o ON cs.shop_id = o.shop_id
GROUP BY
  cs.shop_id, cs.name
ORDER BY
  average_amount DESC;
```

Python

Для реальной работой с базой данных пользоваться онлайн-компилятором sql нельзя. Поэтому мы выбрали python библиотеку sqlite для удобной работы с базой данных в консоли. Конечно, можно было бы масштабировать проект и пользоваться асинхронными веб-библиотеками для отображения таблиц в HTML, но наша основная задача — изучение SQL.

В работе мы использовали Python вместе с библиотекой sqlite3 для взаимодействия с базой данных. Код позволяет создавать таблицы, добавлять данные, просматривать содержимое таблиц и выполнять запросы из файла input.txt.

Пример выполнения запроса:

```
cursor.execute(el)
result = cursor.fetchall()
for el in result:
    print(el)
```

Приведенный выше код Python представляет из себя скрипт для работы с базой данных. Он позволяет управлять созданием таблиц, добавлением данных и выполнением запросов. Использование файлов для хранения SQL-запросов делает код более гибким и позволяет легко изменять запросы без изменения кода.

В коде ниже мы используем работу с файлами, чтобы указывать sql команды. Пункты 0 и 1 были описаны выше. Во 2 пункте в функции show_db() показывается содержимое всех таблиц. Пункты 3 и 4 различаются предназначением для работы с бд. Конечно это можно было реализовать одной командой, но бывают сценарии, когда нельзя давать доступ на изменение таблиц.

```
import sqlite3

def show_db():
        cursor = conn.cursor()

        cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
        tables = cursor.fetchall()

        for table in tables:
        table_name = table[0]
        cursor.execute(f"SELECT * FROM {table_name}")
        rows = cursor.fetchall()
        print(f"\nСодержимое таблицы {table_name}:")
```

```
        cursor.execute(f"PRAGMA table_info({table_name})")
        print([i[1] for i in cursor.fetchall()])
        for row in rows:
        print(row)


try:
        conn = sqlite3.connect('coffeeshop.db')
except Exception as e:
        file = open("coffeeshop.db", "w+")
        file.close()
        conn = sqlite3.connect('coffeeshop.db')

text = '''
Создать таблицы - 0
Заполнить таблицы - 1
Посмотреть таблицы - 2
input.txt - запрос без редактирования бд - 3
input.txt - изменение бд - 4
Завершить работу - всё остальное'''
print(text)

while True:
        try:
        com = input('\n')
        if com in ["0", "1"]:
        file_name = "data/create_table.txt"
        if com == "1":
                file_name = "data/add_data.txt"
        file = open(file_name)
        commands = file.read().split('\n\n')
        file.close()
        cursor = conn.cursor()
        for el in commands:
                cursor.execute(el)
        conn.commit()

        elif com == "2":
        show_db()
        elif com in ["3", "4"] :
        cursor = conn.cursor()
        file = open('input.txt')
        el = file.read()
```

```
            file.close()
            cursor.execute(el)
            if com == "3":
                        result = cursor.fetchall()
                        for el in result:
                        print(el)
            else:
                        conn.commit()
            else:
            break

            except Exception as e:
            print("Ошибка: " + str(e))

conn.close()
```

Заключение

Таким образом, можно сделать вывод, что использование SQLite вместе с Python для управления базой данных кофейни предоставляет простое и эффективное решение. Такой подход позволяет быстро создать систему, способную обслуживать небольшие и средние бизнес-процессы. Предоставленный пример кода служит основой для дальнейших разработок и расширений функциональности информационной системы для кофейни.

Источники

1. Лучшие библиотеки Python для создания баз данных SQL // eternalhos URL: https://eternalhost.net/blog/razrabotka/python-mysql-postgresql-sqlite (дата обращения: 22.12.2023).

2. Руководство по SQLite // METANIT URL: https://metanit.com/sql/sqlite/ (дата обращения: 22.12.2023).

3. Работа с SQLite в Python (для чайников) // Хабр URL: https://habr.com/ru/articles/754400/ (дата обращения: 22.12.2023).

4. sqlite3 — DB-API 2.0 interface for SQLite databases // Python URL: https://docs.python.org/3/library/sqlite3.html (дата обращения: 22.12.2023).