

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии  
Департамент цифровых, робототехнических систем и электроники

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №12**  
**дисциплины**  
**«Объектно-ориентированное программирование»**  
**Вариант № 18**

Выполнил:  
Текеева Мадина Азрет-Алиевна  
3 курс, группа ИВТ-б-о-23-2,  
09.03.01 «Информатика и  
вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем», очная  
форма обучения

---

(подпись)

Проверил:  
Доцент департамента цифровых,  
робототехнических систем и  
электроники института перспективной  
инженерии Воронкин Р.А

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2025 г.

**Тема:** Взаимодействие с базами данных SQLite3 в Python.

**Цель работы:** освоить взаимодействие с базами данных SQLite3 с помощью языка программирования Python.

**Репозиторий:** [https://github.com/bickrosss/TM\\_oop\\_lab12](https://github.com/bickrosss/TM_oop_lab12)

### Порядок выполнения работы:

1. Создание объекта соединения с базой данных SQLite с использованием функции connect(). База может быть сохранена в файл или создана в оперативной памяти.

```
import sqlite3

# Создание соединения с файлом базы данных
con = sqlite3.connect('mydatabase.db')

# Или создание базы данных в оперативной памяти
con = sqlite3.connect(':memory:')
```

2. Создание таблицы employees с помощью SQL-запроса CREATE TABLE, выполненного через метод execute() курсора.

```
import sqlite3

con = sqlite3.connect('mydatabase.db')
cursor_obj = con.cursor()

cursor_obj.execute("""
    CREATE TABLE employees (
        id integer PRIMARY KEY,
        name text,
        salary real,
        department text,
        position text,
        hireDate text)
""")
con.commit()
```

3. Добавление данных в таблицу с использованием оператора INSERT INTO.

```
import sqlite3

con = sqlite3.connect('mydatabase.db')
cursor_obj = con.cursor()
```

```
# Вставка данных напрямую в запрос
cursor_obj.execute("INSERT INTO employees VALUES(1, 'John', 700, 'HR', 'Manager',
'2017-01-04')")

# Вставка данных с использованием заполнителей ?
entities = (2, 'Andrew', 800, 'IT', 'Tech', '2018-02-06')
cursor_obj.execute("INSERT INTO employees VALUES (?, ?, ?, ?, ?, ?)", entities)

con.commit()
```

4. Обновление записи в таблице employees с помощью оператора UPDATE и условия WHERE.

```
import sqlite3

con = sqlite3.connect('mydatabase.db')
cursor_obj = con.cursor()

cursor_obj.execute("UPDATE employees SET name = 'Rogers' WHERE id = 2")
con.commit()
```

5. Получение всех записей из таблицы employees с помощью оператора SELECT и метода fetchall().

```
import sqlite3

con = sqlite3.connect('mydatabase.db')
cursor_obj = con.cursor()

cursor_obj.execute("SELECT * FROM employees")
rows = cursor_obj.fetchall()

for row in rows:
    print(row)
```

```
@bickrosss →/workspaces/TM_oop_lab12 (main) $ python examples/select_data.py
(1, 'John', 700.0, 'HR', 'Manager', '2017-01-04')
(2, 'Rogers', 800.0, 'IT', 'Tech', '2018-02-06')
```

Рисунок 1. Получение записей

6. Использование модуля datetime для работы с датами в SQLite и вставка данных с датами.

```
import sqlite3
import datetime
def adapt_date(date_obj):
    return date_obj.isoformat()
```

```

sqlite3.register_adapter(datetime.date, adapt_date)

con = sqlite3.connect('mydatabase.db')
cursor_obj = con.cursor()

cursor_obj.execute("""
    CREATE TABLE IF NOT EXISTS assignments(
        id INTEGER,
        name TEXT,
        date TEXT -- Используем TEXT вместо DATE для совместимости
    )
""")

data = [
    (1, "Ridesharing", datetime.date(2017, 1, 2)),
    (2, "Water Purifying", datetime.date(2018, 3, 4))
]

cursor_obj.executemany("INSERT INTO assignments VALUES(?, ?, ?)", data)
con.commit()
print("Данные с датами успешно добавлены!")
con.close()

```

7. Создание полноценного консольного приложения для учёта сотрудников с использованием SQLite, датаклассов и модуля argparse. Поддерживает добавление, отображение и выборку сотрудников по стажу.

```

import sqlite3
import argparse
from pathlib import Path
from dataclasses import dataclass

@dataclass
class Post:
    id: int
    title: str

@dataclass
class Worker:
    name: str
    post: str
    year: int

class StaffRepository:
    def __init__(self, db_path: str):
        self.db_path = db_path
        self._create_tables()

```

```

def _create_tables(self):
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS posts (
            post_id INTEGER PRIMARY KEY AUTOINCREMENT,
            post_title TEXT UNIQUE NOT NULL
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS workers (
            worker_id INTEGER PRIMARY KEY AUTOINCREMENT,
            worker_name TEXT NOT NULL,
            post_id INTEGER NOT NULL,
            worker_year INTEGER NOT NULL,
            FOREIGN KEY(post_id) REFERENCES posts(post_id)
        )
    """)
    conn.commit()
    conn.close()

def get_or_create_post(self, title: str) -> int:
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    cursor.execute("SELECT post_id FROM posts WHERE post_title = ?",
(title,))
    row = cursor.fetchone()
    if row is None:
        cursor.execute("INSERT INTO posts (post_title) VALUES (?)", (title,))
        post_id = cursor.lastrowid
        conn.commit()
    else:
        post_id = row[0]
    conn.close()
    return post_id

def add_worker(self, name: str, post: str, year: int):
    post_id = self.get_or_create_post(post)
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO workers (worker_name, post_id, worker_year)
        VALUES (?, ?, ?)
    """, (name, post_id, year))
    conn.commit()
    conn.close()

def get_all_workers(self) -> list[Worker]:
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    cursor.execute("""

```

```

        SELECT workers.worker_name, posts.post_title, workers.worker_year
        FROM workers
        INNER JOIN posts ON posts.post_id = workers.post_id
    """
    rows = cursor.fetchall()
    conn.close()
    return [Worker(name=row[0], post=row[1], year=row[2]) for row in rows]

def select_by_period(self, period: int) -> list[Worker]:
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    cursor.execute("""
        SELECT workers.worker_name, posts.post_title, workers.worker_year
        FROM workers
        INNER JOIN posts ON posts.post_id = workers.post_id
        WHERE (strftime('%Y', date('now')) - workers.worker_year) >= ?
    """, (period,))
    rows = cursor.fetchall()
    conn.close()
    return [Worker(name=row[0], post=row[1], year=row[2]) for row in rows]

def display_workers(workers: list[Worker]):
    if not workers:
        print("Список работников пуст.")
        return
    # Вывод таблицы в консоль
    # ... (код для форматирования таблицы)

def main():
    parser = argparse.ArgumentParser(description="Учёт сотрудников")
    parser.add_argument("--db", default="workers.db", help="Путь к базе данных")
    subparsers = parser.add_subparsers(dest="command")

    # Подкоманда add
    add_parser = subparsers.add_parser("add")
    add_parser.add_argument("--name", required=True)
    add_parser.add_argument("--post", required=True)
    add_parser.add_argument("--year", type=int, required=True)

    # Подкоманда display
    subparsers.add_parser("display")

    # Подкоманда select
    select_parser = subparsers.add_parser("select")
    select_parser.add_argument("--period", type=int, required=True)

    args = parser.parse_args()
    repo = StaffRepository(args.db)

    if args.command == "add":
        repo.add_worker(args.name, args.post, args.year)

```

```

        print("Сотрудник добавлен.")
    elif args.command == "display":
        display_workers(repo.get_all_workers())
    elif args.command == "select":
        display_workers(repo.select_by_period(args.period))

if __name__ == "__main__":
    main()

```

```

usage: full_application.py [-h] [--db DB] {add,display,select} ...

Учёт сотрудников

positional arguments:
  {add,display,select}

options:
  -h, --help            show this help message and exit
  --db DB               Путь к базе данных

```

Рисунок 2. Консольное приложение

#### 8. Выполнили задание 1.

Во всех заданиях требуется разработать консольное приложение на Python, реализующее хранение, обработку и отображение структурированных данных. Каждая программа должна использовать объектно-ориентированный подход, а описания основных сущностей необходимо выполнить с помощью декоратора `@dataclass`. Все данные должны сохраняться в базе SQLite, при этом структура базы должна включать не менее двух таблиц, связанных внешним ключом или логической зависимостью. Приложение должно автоматически создавать базу данных и её таблицы при первом запуске, если файл базы отсутствует.

Работу с базой данных следует инкапсулировать в отдельном классе-репозитории, содержащем методы для добавления, выборки и обработки данных. Логику формирования SQL-запросов, а также создание и настройку соединения со SQLite рекомендуется разместить внутри этого класса. Приложение должно обеспечивать корректное преобразование строк таблиц в объекты соответствующих датаклассов и обратно.

Взаимодействие пользователя с программой необходимо организовать через интерфейс командной строки с использованием модуля `argparse` или `click`. Каждая программа должна поддерживать подкоманды для добавления новых объектов, отображения всех записей, выполнения выборок по определённым критериям и указания пути к файлу базы. Должна быть предусмотрена команда для вывода справки и, при необходимости, информации о версии программы.

Вывод данных в консоль следует оформлять в аккуратном табличном виде или в ином структурированном формате, обеспечивающем удобство чтения. Все операции должны выполняться устойчиво к ошибкам отсутствующих данных или пустых выборок. Структура кода должна быть модульной и расширяемой.

## 18. Учёт рецептов

Разработать базу рецептов: таблицы `recipes` и `ingredients`. Позволить добавлять рецепт, ингредиент, выводить рецепты по ингредиенту.

Консольное приложение:

```
import argparse
from pathlib import Path

from recipe_repository import RecipeRepository, display_recipes

def main() -> None:
    parser = argparse.ArgumentParser(description="Учёт рецептов")
    subparsers = parser.add_subparsers(dest="command", help="Команды")

    add_recipe = subparsers.add_parser("add_recipe", help="Добавить рецепт")
    add_recipe.add_argument("name", help="Название рецепта")
    add_recipe.add_argument("description", help="Описание")

    add_ingredient = subparsers.add_parser("add_ingredient", help="Добавить
ингредиент")
    add_ingredient.add_argument("recipe_id", type=int, help="ID рецепта")
    add_ingredient.add_argument("name", help="Название ингредиента")
    add_ingredient.add_argument("amount", help="Количество")

    subparsers.add_parser("show_recipes", help="Показать все рецепты")

    by_ing = subparsers.add_parser(
```

```

        "find_by_ingredient", help="Найти рецепты по ингредиенту"
    )
    by_ing.add_argument("name", help="Название ингредиента")

    args = parser.parse_args()

    repo = RecipeRepository(Path("recipes.db"))

    if args.command == "add_recipe":
        repo.add_recipe(args.name, args.description)
        print(f"Рецепт '{args.name}' добавлен")

    elif args.command == "add_ingredient":
        repo.add_ingredient(args.recipe_id, args.name, args.amount)
        print("Ингредиент добавлен")

    elif args.command == "show_recipes":
        display_recipes(repo.get_recipes())

    elif args.command == "find_by_ingredient":
        recipes = repo.get_recipes_by_ingredient(args.name)
        display_recipes(recipes)

    else:
        parser.print_help()

if __name__ == "__main__":
    main()

```

### Репозиторий:

```

import sqlite3
from dataclasses import dataclass
from pathlib import Path

@dataclass(frozen=True)
class Recipe:
    id: int
    name: str
    description: str

    def str(self) -> str:
        return f"ID: {self.id:>3} | Название: {self.name:<20} | Описание: {self.description}"

@dataclass(frozen=True)
class Ingredient:
    id: int

```

```

recipe_id: int
name: str
amount: str

def str(self) -> str:
    return (
        f"ID: {self.id:>3} | Рецепт: {self.recipe_id:>3} | "
        f"Ингредиент: {self.name:<15} | Кол-во: {self.amount}"
    )

```

```

class RecipeRepository:
    def __init__(self, db_path: Path) -> None:
        self.db_path = db_path
        self._create_db()

    def _connect(self) -> sqlite3.Connection:
        return sqlite3.connect(self.db_path)

    def _create_db(self) -> None:
        with self._connect() as conn:
            cursor = conn.cursor()

            cursor.execute(
                """
                CREATE TABLE IF NOT EXISTS recipes (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT NOT NULL,
                    description TEXT
                )
            """
            )

            cursor.execute(
                """
                CREATE TABLE IF NOT EXISTS ingredients (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    recipe_id INTEGER NOT NULL,
                    name TEXT NOT NULL,
                    amount TEXT,
                    FOREIGN KEY (recipe_id) REFERENCES recipes(id)
                )
            """
            )

    def add_recipe(self, name: str, description: str) -> None:
        with self._connect() as conn:
            cursor = conn.cursor()
            cursor.execute(
                "INSERT INTO recipes (name, description) VALUES (?, ?)",
                (name, description),
            )

```

```

    )

def add_ingredient(self, recipe_id: int, name: str, amount: str) -> None:
    with self._connect() as conn:
        cursor = conn.cursor()
        cursor.execute(
            """
            INSERT INTO ingredients (recipe_id, name, amount)
            VALUES (?, ?, ?)
            """,
            (recipe_id, name, amount),
        )

def get_recipes(self) -> list[Recipe]:
    with self._connect() as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT id, name, description FROM recipes")
        rows = cursor.fetchall()
        return [Recipe(*row) for row in rows]

def get_recipes_by_ingredient(self, ingredient_name: str) -> list[Recipe]:
    with self._connect() as conn:
        cursor = conn.cursor()
        cursor.execute(
            """
            SELECT DISTINCT r.id, r.name, r.description
            FROM recipes r
            JOIN ingredients i ON r.id = i.recipe_id
            WHERE i.name LIKE ?
            """,
            (f"%{ingredient_name}%",),
        )
        rows = cursor.fetchall()
        return [Recipe(*row) for row in rows]

def display_recipes(recipes: list[Recipe]) -> None:
    if not recipes:
        print("Список рецептов пуст.")
        return

    line = f"++{'-' * 4}+-{'-' * 20}+-{'-' * 25}+"
    print(line)
    print(f"| {'ID':^4} | {'Название':^20} | {'Описание':^25} |")
    print(line)
    for r in recipes:
        print(f"| {r.id:^4} | {r.name:^20} | {r.description:^25} |")

    print(line)

```

```

@bickrosss →/workspaces/ТМ_oop_lab12 (main) $ python tasks/task1/task_1.py --help
usage: task_1.py [-h]
                {add_recipe,add_ingredient,show_recipes,find_by_ingredient}
                ...

Учёт рецептов

positional arguments:
  {add_recipe,add_ingredient,show_recipes,find_by_ingredient}
                                Команды
  add_recipe                  Добавить рецепт
  add_ingredient              Добавить ингредиент
  show_recipes                Показать все рецепты
  find_by_ingredient          Найти рецепты по ингредиенту

options:
  -h, --help                show this help message and exit
@bickrosss →/workspaces/ТМ_oop_lab12 (main) $ python tasks/task1/task_1.py add_recipe "Паста" "С сыром"
Рецепт 'Паста' добавлен
@bickrosss →/workspaces/ТМ_oop_lab12 (main) $ python tasks/task1/task_1.py add_ingredient 1 "Сыр" "200 г"
Ингредиент добавлен
@bickrosss →/workspaces/ТМ_oop_lab12 (main) $ python tasks/task1/task_1.py show_recipes
+-----+-----+-----+
| ID | Название | Описание |
+-----+-----+-----+
| 1 | Паста | С сыром |
+-----+-----+-----+
@bickrosss →/workspaces/ТМ_oop_lab12 (main) $ python tasks/task1/task_1.py find_by_ingredient "Сыр"
+-----+-----+-----+
| ID | Название | Описание |
+-----+-----+-----+
| 1 | Паста | С сыром |
+-----+-----+-----+

```

Рисунок 3. Система учета рецептов

## 9. Решили задание 2.

Изучите, как работать с базами данных SQLite с помощью пакета sqlalchemy. После этого возьмите ваш вариант задания №1 и выполните его заново, заменив всю работу с базой данных на механизм, предоставляемый SQLAlchemy. Ваша программа должна использовать модели, описанные через SQLAlchemy, связи между таблицами и сессию для выполнения операций с данными. Это задание относится к заданиям повышенной сложности и предполагает самостоятельное изучение нового инструмента.

Консольное приложение:

```

import argparse

from db import SessionLocal
from recipe_repository import RecipeRepository, display_recipes

def main() -> None:

```

```

parser = argparse.ArgumentParser(description="Учёт рецептов (SQLAlchemy)")
subparsers = parser.add_subparsers(dest="command")

add_recipe = subparsers.add_parser("add_recipe")
add_recipe.add_argument("name")
add_recipe.add_argument("description")

add_ingredient = subparsers.add_parser("add_ingredient")
add_ingredient.add_argument("recipe_id", type=int)
add_ingredient.add_argument("name")
add_ingredient.add_argument("amount")

subparsers.add_parser("show_recipes")

find = subparsers.add_parser("find_by_ingredient")
find.add_argument("name")

args = parser.parse_args()

if not args.command:
    parser.print_help()
    return

session = SessionLocal()
try:
    repo = RecipeRepository(session)

    if args.command == "add_recipe":
        repo.add_recipe(args.name, args.description)
        print("Рецепт добавлен")

    elif args.command == "add_ingredient":
        repo.add_ingredient(args.recipe_id, args.name, args.amount)
        print("Ингредиент добавлен")

    elif args.command == "show_recipes":
        display_recipes(repo.get_recipes())

    elif args.command == "find_by_ingredient":
        display_recipes(repo.get_recipes_by_ingredient(args.name))

finally:
    session.close()

if __name__ == "__main__":
    main()

```

Репозиторий:

```

from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship, Session

try:
    from tasks.task2.db import Base, engine
except ImportError:
    from db import Base, engine

class Recipe(Base):
    __tablename__ = "recipes"

    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    description = Column(String)

    ingredients = relationship("Ingredient", back_populates="recipe")

class Ingredient(Base):
    __tablename__ = "ingredients"

    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    amount = Column(String)

    recipe_id = Column(Integer, ForeignKey("recipes.id"))
    recipe = relationship("Recipe", back_populates="ingredients")

Base.metadata.create_all(engine)

class RecipeRepository:
    def __init__(self, session: Session):
        self.session = session

    def add_recipe(self, name: str, description: str) -> None:
        recipe = Recipe(name=name, description=description)
        self.session.add(recipe)
        self.session.commit()

    def add_ingredient(self, recipe_id: int, name: str, amount: str) -> None:
        ing = Ingredient(recipe_id=recipe_id, name=name, amount=amount)
        self.session.add(ing)
        self.session.commit()

    def get_recipes(self):
        return self.session.query(Recipe).all()

    def get_recipes_by_ingredient(self, ingredient_name: str):

```

```

        return (
            self.session.query(Recipe)
                .join(Ingredient)
                .filter(Ingredient.name.like(f"%{ingredient_name}%"))
                .all()
        )

def display_recipes(recipes):
    if not recipes:
        print("Список рецептов пуст.")
        return

    line = f"+-{'-' * 4}-+-{'-' * 20}-+-{'-' * 25}+"
    print(line)
    print(f"| {'ID':^4} | {'Название':^20} | {'Описание':^25} |")
    print(line)
    for r in recipes:
        print(f"| {r.id:^4} | {r.name:^20} | {r.description:^25} |")

    print(line)

```

```

@bickrosss →/workspaces/ТМ_ооп_lab12 (main) $ python tasks/task2/task_2.py
usage: task_2.py [-h]
                 {add_recipe,add_ingredient,show_recipes,find_by_ingredient}
                 ...

Учёт рецептов (SQLAlchemy)

positional arguments:
  {add_recipe,add_ingredient,show_recipes,find_by_ingredient}

options:
  -h, --help            show this help message and exit
@bickrosss →/workspaces/ТМ_ооп_lab12 (main) $ python tasks/task2/task_2.py add_recipe "Салат" "Овощной"
Рецепт добавлен
@bickrosss →/workspaces/ТМ_ооп_lab12 (main) $ python tasks/task2/task_2.py add_ingredient 1 "Огурец" "2 шт"
Ингредиент добавлен
@bickrosss →/workspaces/ТМ_ооп_lab12 (main) $ python tasks/task2/task_2.py show_recipes
+-----+-----+-----+
| ID |      Название      |      Описание      |
+-----+-----+-----+
|  1  |      Паста         |      С сыром       |
|  2  |      Салат         |      Овощной       |
+-----+-----+-----+
@bickrosss →/workspaces/ТМ_ооп_lab12 (main) $ python tasks/task2/task_2.py find_by_ingredient "Огурец"
+-----+-----+-----+
| ID |      Название      |      Описание      |
+-----+-----+-----+
|  1  |      Паста         |      С сыром       |
+-----+-----+-----+

```

Рисунок 4. Система учета рецептов с использованием SQLAlchemy

**Вывод:** в ходе выполнения лабораторной работы было освоено взаимодействие с базами данных SQLite3 с помощью языка программирования Python.