МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по рубежному контролю №2

Выполнила:
Студентка группы ИУ5-36Б
Быков К. А.
Преподаватель:
Нардид А. Н.

Москва 2025

Условия рубежного контроля №2 по курсу ПиК ЯП

Рубежный контроль представляет собой разработку тестов на языке Python.

1) Проведите рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования.

2) Для текста программы рубежного контроля №1 создайте модульные тесты с применением TDD - фреймворка (3 теста).

Library.py

```
import sqlite3
from datetime import datetime

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

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS Авторы (
            id_автора INTEGER PRIMARY KEY,
            фамилия TEXT NOT NULL,
            страна TEXT NOT NULL
        )
    ''')

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS Книги (
            id_книги INTEGER PRIMARY KEY,
            название TEXT NOT NULL,
            год_издания INTEGER NOT NULL,
            id_автора INTEGER NOT NULL,
            FOREIGN KEY (id_автора) REFERENCES Авторы (id_автора)
        )
    ''')

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS Читатели (
            id_читателя INTEGER PRIMARY KEY,
            фамилия TEXT NOT NULL,
            адрес TEXT NOT NULL
        )
    ''')

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS Выдачи (
            id_выдачи INTEGER PRIMARY KEY,
            id_книги INTEGER NOT NULL,
            id_читателя INTEGER NOT NULL,
            дата_выдачи DATE NOT NULL,
            дата_возврата DATE,
            FOREIGN KEY (id_книги) REFERENCES Книги (id_книги),
```

```
            FOREIGN KEY (id_читателя) REFERENCES Читатели (id_читателя)
        )
    ''')

    conn.commit()

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

    authors = [
        (1, 'Толстой', 'Россия'),
        (2, 'Достоевский', 'Россия'),
        (3, 'Оруэлл', 'Великобритания'),
        (4, 'Хемингуэй', 'США'),
        (5, 'Маркес', 'Колумбия')
    ]

    books = [
        (1, 'Война и мир', 1869, 1),
        (2, 'Преступление и наказание', 1866, 2),
        (3, '1984', 1949, 3),
        (4, 'Старик и море', 1952, 4),
        (5, 'Сто лет одиночества', 1967, 5),
        (6, 'Анна Каренина', 1877, 1),
        (7, 'Идиот', 1869, 2)
    ]

    readers = [
        (1, 'Иванов', 'ул. Ленина, 10'),
        (2, 'Петров', 'ул. Пушкина, 25'),
        (3, 'Сидорова', 'пр. Мира, 15'),
        (4, 'Кузнецов', 'ул. Садовая, 8')
    ]

    issues = [
        (1, 1, 1, '2024-01-15', '2024-02-01'),
        (2, 3, 2, '2024-01-20', '2024-02-10'),
        (3, 2, 1, '2024-02-01', None),
        (4, 4, 3, '2024-02-05', '2024-02-20'),
        (5, 5, 4, '2024-02-10', None),
        (6, 6, 2, '2024-02-15', None),
        (7, 7, 3, '2024-02-18', None)
    ]

    cursor.executemany('INSERT OR IGNORE INTO Авторы VALUES (?, ?, ?)', authors)
    cursor.executemany('INSERT OR IGNORE INTO Книги VALUES (?, ?, ?, ?)', books)
    cursor.executemany('INSERT OR IGNORE INTO Читатели VALUES (?, ?, ?)',
readers)
    cursor.executemany('INSERT OR IGNORE INTO Выдачи VALUES (?, ?, ?, ?, ?)',
issues)
```

```
    conn.commit()

def query_1(conn):
    cursor = conn.cursor()
    cursor.execute('''
        SELECT DISTINCT Авторы.фамилия
        FROM Авторы
        JOIN Книги ON Авторы.id_автора = Книги.id_автора
        WHERE Книги.год_издания BETWEEN 1900 AND 2000
        ORDER BY Авторы.фамилия
    ''')
    return cursor.fetchall()

def query_2(conn):
    cursor = conn.cursor()
    cursor.execute('''
        SELECT Книги.название, Авторы.фамилия
        FROM Книги
        JOIN Авторы ON Книги.id_автора = Авторы.id_автора
        JOIN Выдачи ON Книги.id_книги = Выдачи.id_книги
        WHERE Выдачи.дата_возврата IS NULL
        ORDER BY Книги.название
    ''')
    return cursor.fetchall()

def query_3(conn):
    cursor = conn.cursor()
    cursor.execute('''
        SELECT Читатели.фамилия, COUNT(Выдачи.id_выдачи) as количество_книг
        FROM Читатели
        LEFT JOIN Выдачи ON Читатели.id_читателя = Выдачи.id_читателя
        GROUP BY Читатели.id_читателя, Читатели.фамилия
        ORDER BY количество_книг DESC
    ''')
    return cursor.fetchall()

def get_all_results(conn):
    """Возвращает результаты всех запросов для тестирования"""
    return (
        query_1(conn),
        query_2(conn),
        query_3(conn)
    )

def main():
    conn = sqlite3.connect('library.db')

    create_tables(conn)
    insert_data(conn)

    print("Запрос 1: Авторы, чьи книги изданы в период с 1900 по 2000 год")
```

```
    result1 = query_1(conn)
    for row in result1:
        print(row[0])

    print("\nЗапрос 2: Книги, которые сейчас на руках")
    result2 = query_2(conn)
    for row in result2:
        print(f"{row[0]} - {row[1]}")

    print("\nЗапрос 3: Количество книг на руках у каждого читателя")
    result3 = query_3(conn)
    for row in result3:
        print(f"{row[0]}: {row[1]}")

    conn.close()

if __name__ == "__main__":
    main()
```

library_test.py

```
import unittest
import sqlite3
from library import (
    create_tables,
    insert_data,
    query_1,
    query_2,
    query_3
)

class TestLibraryQueries(unittest.TestCase):

    def setUp(self):
        """Создаем in-memory базу данных перед каждым тестом"""
        self.conn = sqlite3.connect(':memory:')
        create_tables(self.conn)
        insert_data(self.conn)

    def tearDown(self):
        """Закрываем соединение после каждого теста"""
        self.conn.close()

    def test_query_1_returns_correct_authors(self):
        """Проверяем, что запрос 1 возвращает правильных авторов"""
        result = query_1(self.conn)

        # Ожидаемые результаты (авторы с книгами 1900-2000 гг)
        expected = [
            ('Маркес',),
            ('Оруэлл',),
```

```
            ('Хемингуэй',)
        ]

        self.assertEqual(result, expected)

    def test_query_2_returns_current_issues(self):
        """Проверяем, что запрос 2 возвращает книги на руках"""
        result = query_2(self.conn)

        # Ожидаемые результаты (название, автор)
        expected = [
            ('Анна Каренина', 'Толстой'),
            ('Идиот', 'Достоевский'),
            ('Преступление и наказание', 'Достоевский'),
            ('Сто лет одиночества', 'Маркес')
        ]

        self.assertEqual(result, expected)

    def test_query_3_counts_books_correctly(self):
        """Проверяем, что запрос 3 правильно считает книги"""
        result = query_3(self.conn)

        # Ожидаемые результаты (фамилия, количество)
        expected = {
            'Иванов': 2,
            'Петров': 2,
            'Сидорова': 2,
            'Кузнецов': 1
        }

        # Проверяем соответствие данных
        self.assertEqual(len(result), 4)
        for surname, count in result:
            self.assertEqual(count, expected[surname])

if __name__ == '__main__':
    unittest.main()
```