Практическая работа №3

1.Код класса для тестирования:

```
import pytest
import sqlite3
from auth_manager import AuthManager

# Фикстуры для базы данных
@pytest.fixture
def database():
    """Создание временной БД в памяти"""
    conn = sqlite3.connect(":memory:")
    yield conn
    conn.close()

@pytest.fixture
def manager(database):
    """Создание менеджера аутентификации"""
    return AuthManager(database)

@pytest.fixture
def preloaded_manager(manager):
    """Менеджер с предустановленными данными"""
    manager.register_user("alex", "secret123", "Canada", 2500.0)
    manager.register_user("maria", "mypass", "Brazil", 1800.0)
    manager.register_user("john", "john123", "Canada", 3200.0)
    manager.register_user("sara", "sara_pass", "Japan", 1500.0)
    return manager
```

```
# Базовые тесты (3 штуки)
def test_table_creation(manager):
    """Проверка создания таблиц в БД"""
    cursor = manager.connection.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users'")
    table = cursor.fetchone()

    assert table is not None
    assert table[0] == 'users'

def test_user_registration(manager):
    """Тестирование регистрации нового пользователя"""
    manager.register_user("demo_user", "demo_pass", "Australia", 5000.0)

    cursor = manager.connection.cursor()
    cursor.execute("SELECT * FROM users WHERE username = 'demo_user'")
    user_data = cursor.fetchone()

    assert user_data is not None
    assert user_data[1] == "demo_user"
    assert user_data[2] == "demo_pass"
    assert user_data[3] == "Australia"
    assert user_data[4] == 5000.0

def test_successful_login(manager):
    """Проверка успешного входа в систему"""
```

```
    manager.register_user("test_login", "login_pass", "Germany", 1000.0)
    user = manager.authenticate_user("test_login", "login_pass")

    assert user is not None
    assert user[1] == 'test_login'

# Параметризованные тесты (3 штуки)
@pytest.mark.parametrize("user, pwd, country, funds", [
    ("client1", "cl1_pass", "Italy", 4200.0),
    ("client2", "cl2_pass", "France", 3800.0),
    ("client3", "cl3_pass", "Spain", 2900.0),
])
def test_multiple_registrations(manager, user, pwd, country, funds):
    """Параметризованная проверка регистрации пользователей"""
    manager.register_user(user, pwd, country, funds)
    result = manager.authenticate_user(user, pwd)

    assert result is not None
    assert result[1] == user
    assert result[3] == country
    assert result[4] == funds

@pytest.mark.parametrize("login, secret, result", [
    ("unknown", "wrong", None),
    ("alex", "incorrect", None),
    ("", "", None),
])
def test_failed_authentication(preloaded_manager, login, secret, result):
    """Параметризованная проверка неудачных попыток входа"""
```

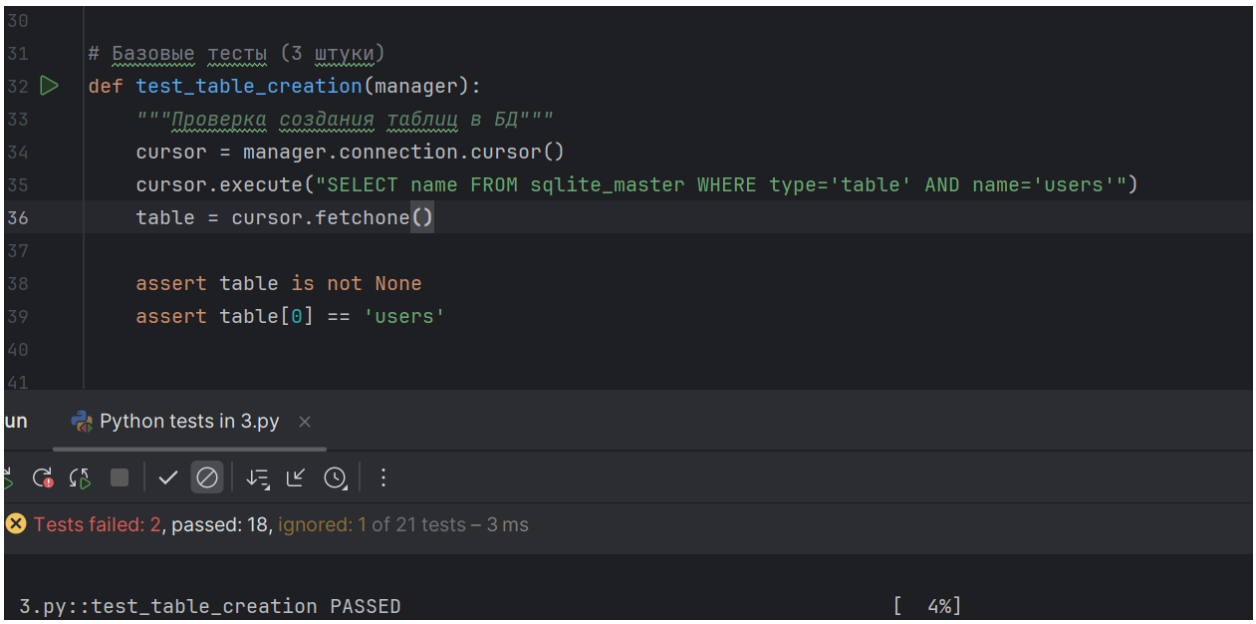
```
    auth_result = preloaded_manager.authenticate_user(login, secret)
    assert auth_result == result


@pytest.mark.parametrize("nation, expected", [
    ("Canada", 2),
    ("Brazil", 1),
    ("Japan", 1),
    ("China", 0),
])
def test_national_user_count(preloaded_manager, nation, expected):
    """Параметризованный подсчет пользователей по национальности"""
    count = preloaded_manager.count_users_by_country(nation)
    assert count == expected


# Тестирование исключений (2 штуки)
def test_insufficient_balance_transfer(preloaded_manager):
    """Проверка исключения при недостаточном балансе"""
    with pytest.raises(ValueError, match="Insufficient funds"):
        preloaded_manager.transfer_balance(2, 1, 5000.0)


def test_transfer_to_invalid_user(preloaded_manager):
    """Проверка поведения при переводе несуществующему пользователю"""
    try:
        preloaded_manager.transfer_balance(1, 999, 300.0)
        pytest.fail("Ожидалось исключение для несуществующего пользователя")
    except Exception:
        # Ожидаемое поведение из-за SQL уязвимости
```

```
        pass


# Тесты с использованием фикстур
def test_user_removal(preloaded_manager):
    """Тестирование удаления пользователя с использованием фикстур"""
    # Проверяем существование пользователя
    user = preloaded_manager.get_user_by_id(1)
    assert user is not None

    # Удаляем пользователя
    preloaded_manager.delete_user(1)

    # Проверяем удаление
    removed_user = preloaded_manager.get_user_by_id(1)
    assert removed_user is None


# Тесты с метками (минимум 2)
@pytest.mark.stress
def test_high_volume_registrations(manager):
    """Стресс-тест массовой регистрации пользователей"""
    for i in range(200):
        manager.register_user(f"load_user_{i}", f"pass_{i}", f"nation_{i % 15}", 1000 + i)

    total = manager.count_users_by_country("nation_0")
    assert total >= 13  # 200 пользователей / 15 стран


@pytest.mark.vulnerability
```

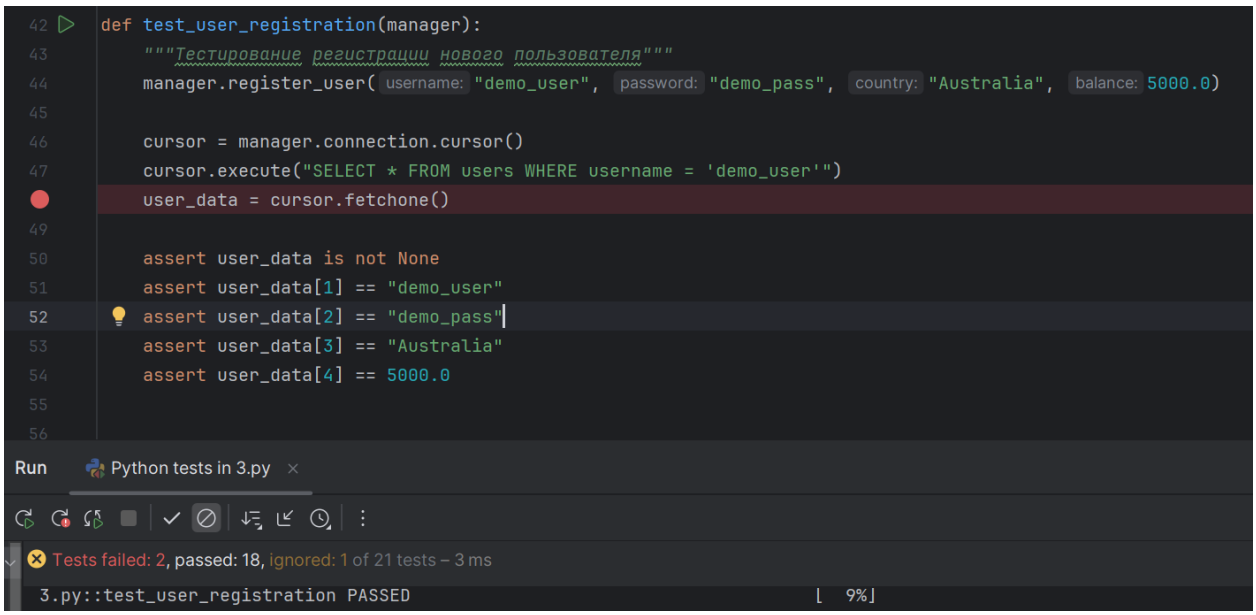
```
def test_sql_injection_in_registration(manager):
    """Тестирование уязвимости SQL инъекции при регистрации"""
    manager.register_user("hacker'; DROP TABLE users; --", "attack", "USA", 1000)

    cursor = manager.connection.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users';")
    tables = cursor.fetchall()

    assert len(tables) == 1  # Таблица должна сохраниться

@pytest.mark.vulnerability
def test_sql_injection_in_authentication(manager):
    """Тестирование уязвимости SQL инъекции при аутентификации"""
    manager.register_user("legit_user", "safe_pass", "UK", 2000.0)

    # SQL инъекция для обхода аутентификации
    user = manager.authenticate_user("legit_user' OR '1'='1", "any_password")
    assert user is not None  # Инъекция должна сработать

# Пропускаемый тест
@pytest.mark.skip(reason="Требует дополнительной конфигурации окружения")
def test_environment_specific():
    """Тест, зависящий от специфичного окружения"""
    assert False

# Дополнительные security тесты
```

```
@pytest.mark.vulnerability
def test_sql_injection_vectors(manager):
    """Тестирование различных векторов SQL инъекций"""
    injection_patterns = [
        "test' OR '1'='1",
        "admin'; DELETE FROM users; --",
        "user' UNION SELECT * FROM users --",
    ]

    for pattern in injection_patterns:
        try:
            manager.register_user(pattern, "test", "test_country", 1000)
            # Проверяем сохранность таблицы
            cursor = manager.connection.cursor()
            cursor.execute("SELECT name FROM sqlite_master WHERE
type='table' AND name='users'")
            table_exists = cursor.fetchone() is not None
            assert table_exists
        except Exception:
            # Игнорируем возможные ошибки выполнения
            pass
```

2.Базовые тесты:



2.1.БАЗОВЫЙ-ТЕСТ



2.2.БАЗОВЫЙ-ТЕСТ

```python
def test_successful_login(manager):
    """Проверка успешного входа в систему"""
    manager.register_user( username: "test_login",  password: "login_pass",  country: "Germany",  balance: 1000.0)
    user = manager.authenticate_user( username: "test_login",  password: "login_pass")

    assert user is not None
    assert user[1] == 'test_login'
```

```
un  Python tests in 3.py
Tests failed: 2, passed: 18, ignored: 1 of 21 tests – 3 ms
3.py::test_successful_login PASSED                    [ 14%]
```

2.3.БАЗОВЫЙ-ТЕСТ

3.Параметризованные тесты:

```python
# Параметризованные тесты (3 штуки)
@pytest.mark.parametrize("user, pwd, country, funds", [
    ("client1", "cl1_pass", "Italy", 4200.0),
    ("client2", "cl2_pass", "France", 3800.0),
    ("client3", "cl3_pass", "Spain", 2900.0),
])
def test_multiple_registrations(manager, user, pwd, country, funds):
    """Параметризованная проверка регистрации пользователей"""
    manager.register_user(user, pwd, country, funds)
    result = manager.authenticate_user(user, pwd)

    assert result is not None
    assert result[1] == user
    assert result[3] == country
    assert result[4] == funds
```

```
Run  Python tests in 3.py
Tests failed: 2, passed: 18, ignored: 1 of 21 tests – 3 ms
3.py::test_multiple_registrations[client1-cl1_pass-Italy-4200.0] PASSED  [ 19%]
3.py::test_multiple_registrations[client2-cl2_pass-France-3800.0] PASSED [ 23%]
3.py::test_multiple_registrations[client3-cl3_pass-Spain-2900.0] PASSED  [ 28%]
```

3.1.ПАРАМЕТРИЗОВАННЫЙ-ТЕСТ

```python
@pytest.mark.parametrize("login, secret, result", [
    ("unknown", "wrong", None),
    ("alex", "incorrect", None),
    ("", "", None),
])
def test_failed_authentication(preloaded_manager, login, secret, result):
    """Параметризованная проверка неудачных попыток входа"""
    auth_result = preloaded_manager.authenticate_user(login, secret)
    assert auth_result == result
```

Run Python tests in 3.py

Tests failed: 2, passed: 18, ignored: 1 of 21 tests – 3 ms

```
3.py::test_failed_authentication[unknown-wrong-None] PASSED          [ 33%]
3.py::test_failed_authentication[alex-incorrect-None] PASSED         [ 38%]
3.py::test_failed_authentication[--None] PASSED                      [ 42%]
```

3.2.ПАРАМЕТРИЗОВАННЫЙ-ТЕСТ

```python
@pytest.mark.parametrize("nation, expected", [
    ("Canada", 2),
    ("Brazil", 1),
    ("Japan", 1),
    ("China", 0),
])
def test_national_user_count(preloaded_manager, nation, expected):
    """Параметризованный подсчет пользователей по национальности"""
    count = preloaded_manager.count_users_by_country(nation)
    assert count == expected
```

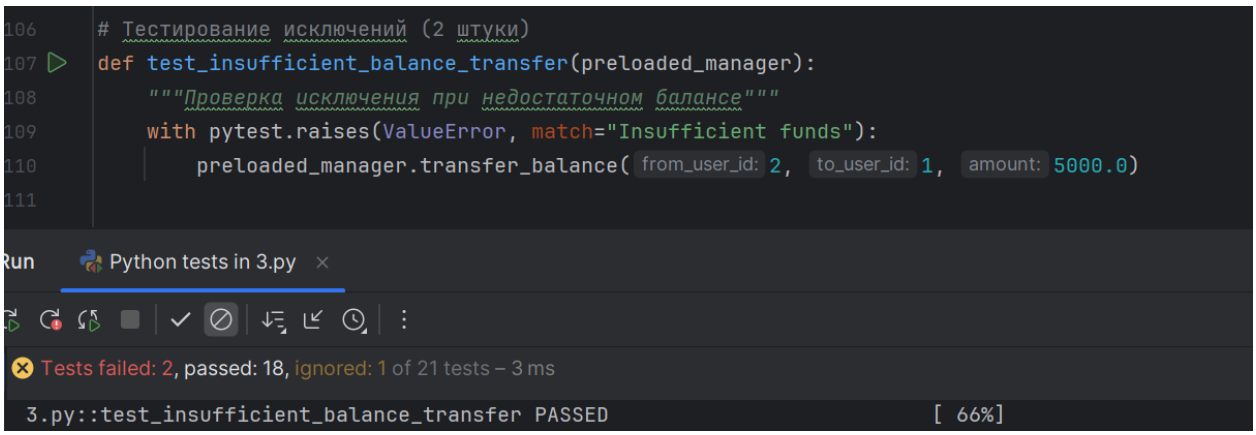
Run Python tests in 3.py

Tests failed: 2, passed: 18, ignored: 1 of 21 tests – 3 ms

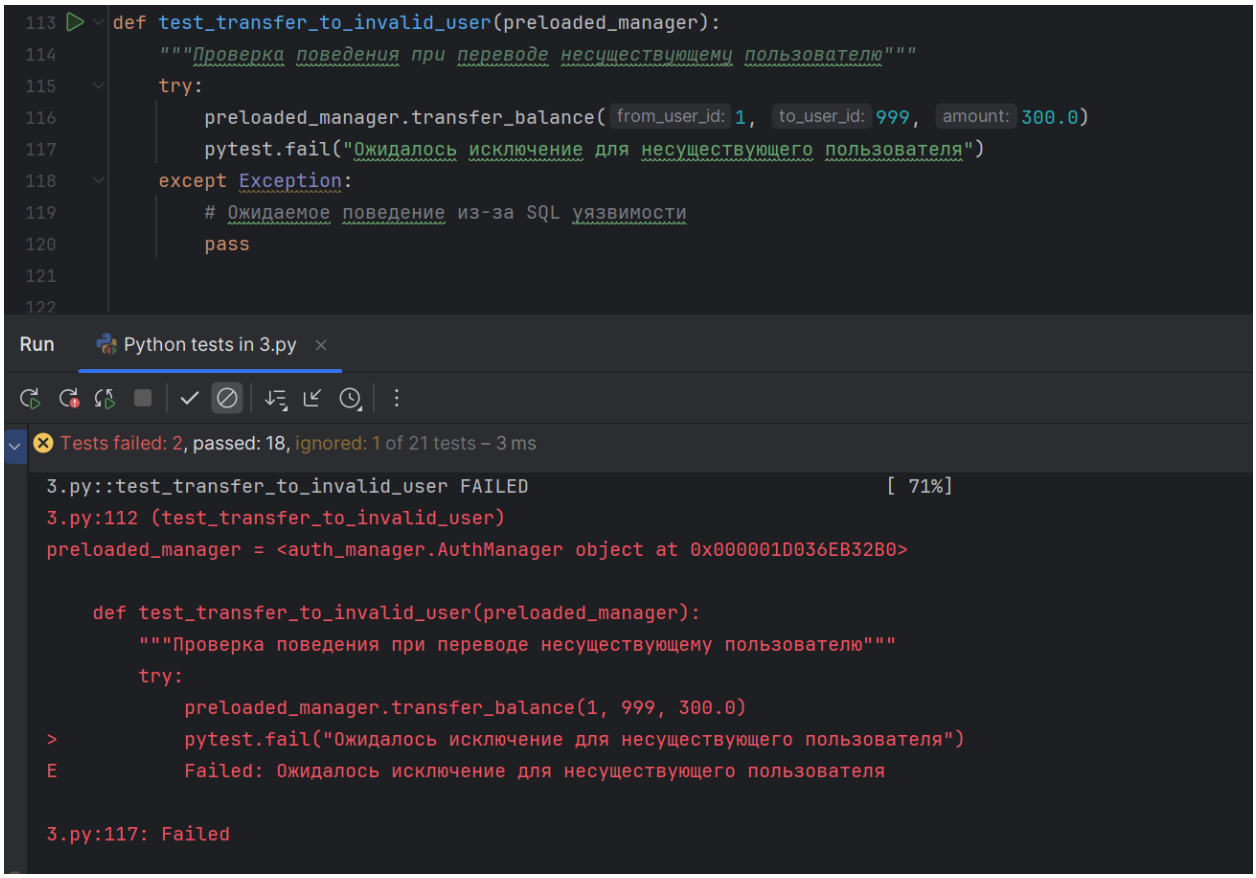
```
3.py::test_national_user_count[Canada-2] PASSED                      [ 47%]
3.py::test_national_user_count[Brazil-1] PASSED                      [ 52%]
3.py::test_national_user_count[Japan-1] PASSED                       [ 57%]
3.py::test_national_user_count[China-0] PASSED                       [ 61%]
```

3.3.ПАРАМЕТРИЗОВАННЫЙ-ТЕСТ

4.Тестирование исключений:

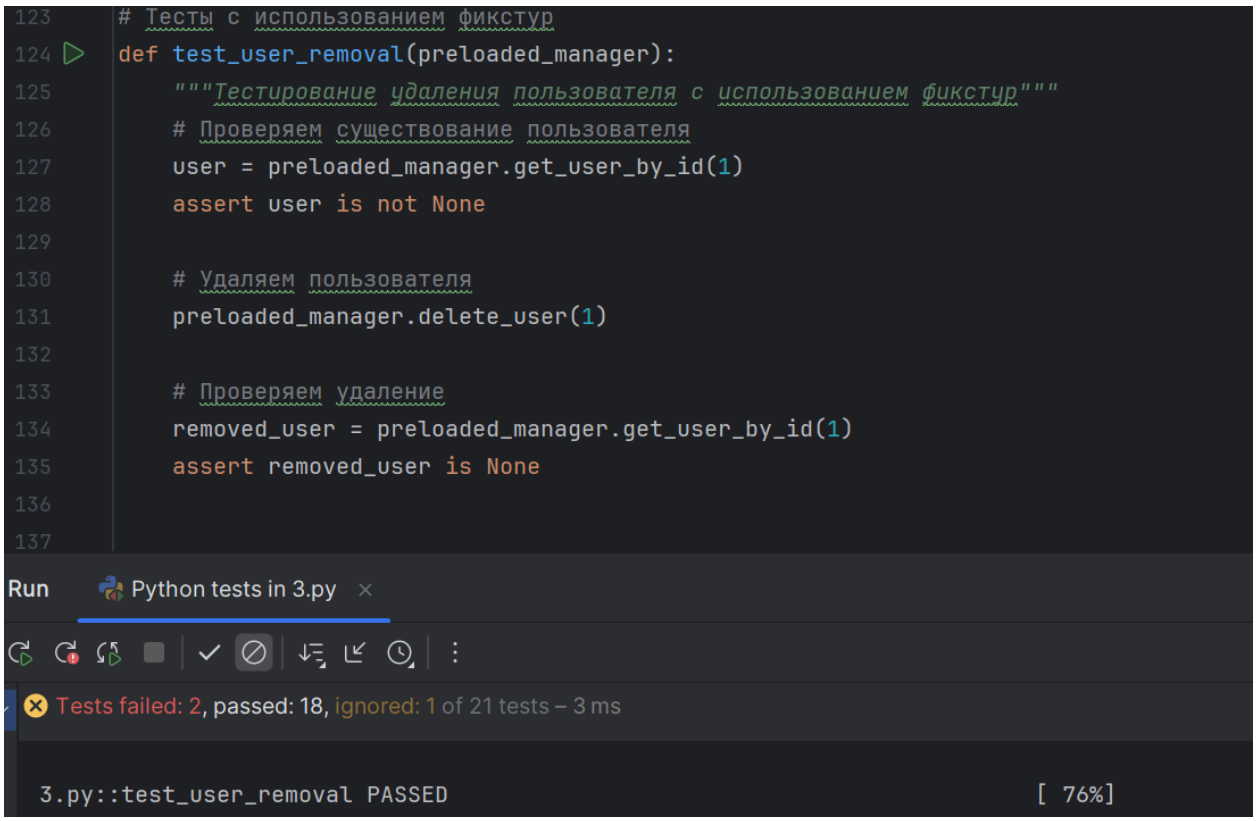


4.1-ТЕСТ-ИСКЛЮЧЕНИЕ



4.2-ТЕСТ-ИСКЛЮЧЕНИЕ

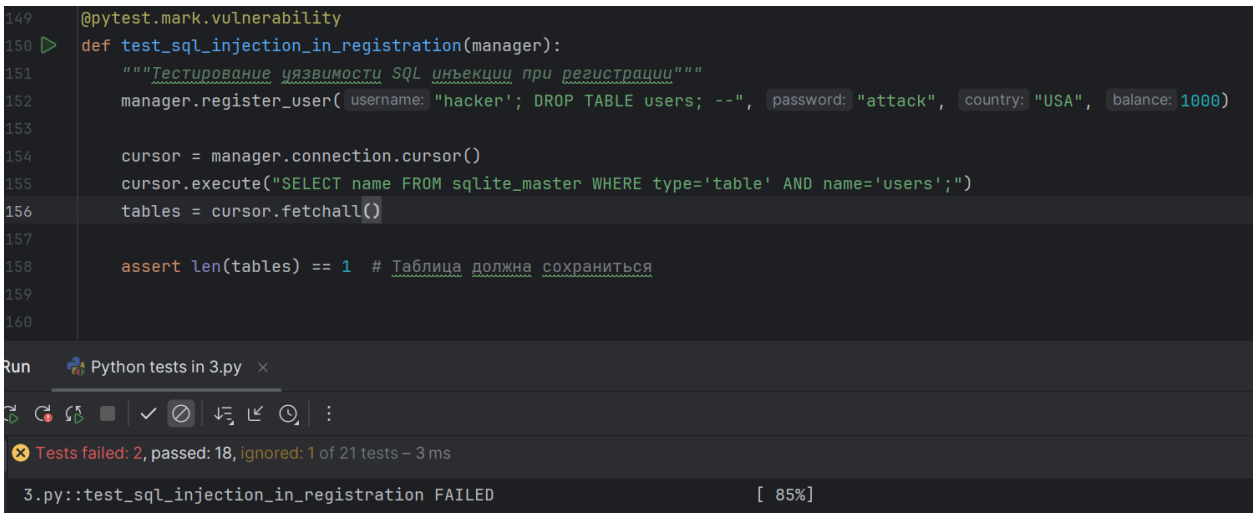
5.Тесты с использованием фикстур базы данных:

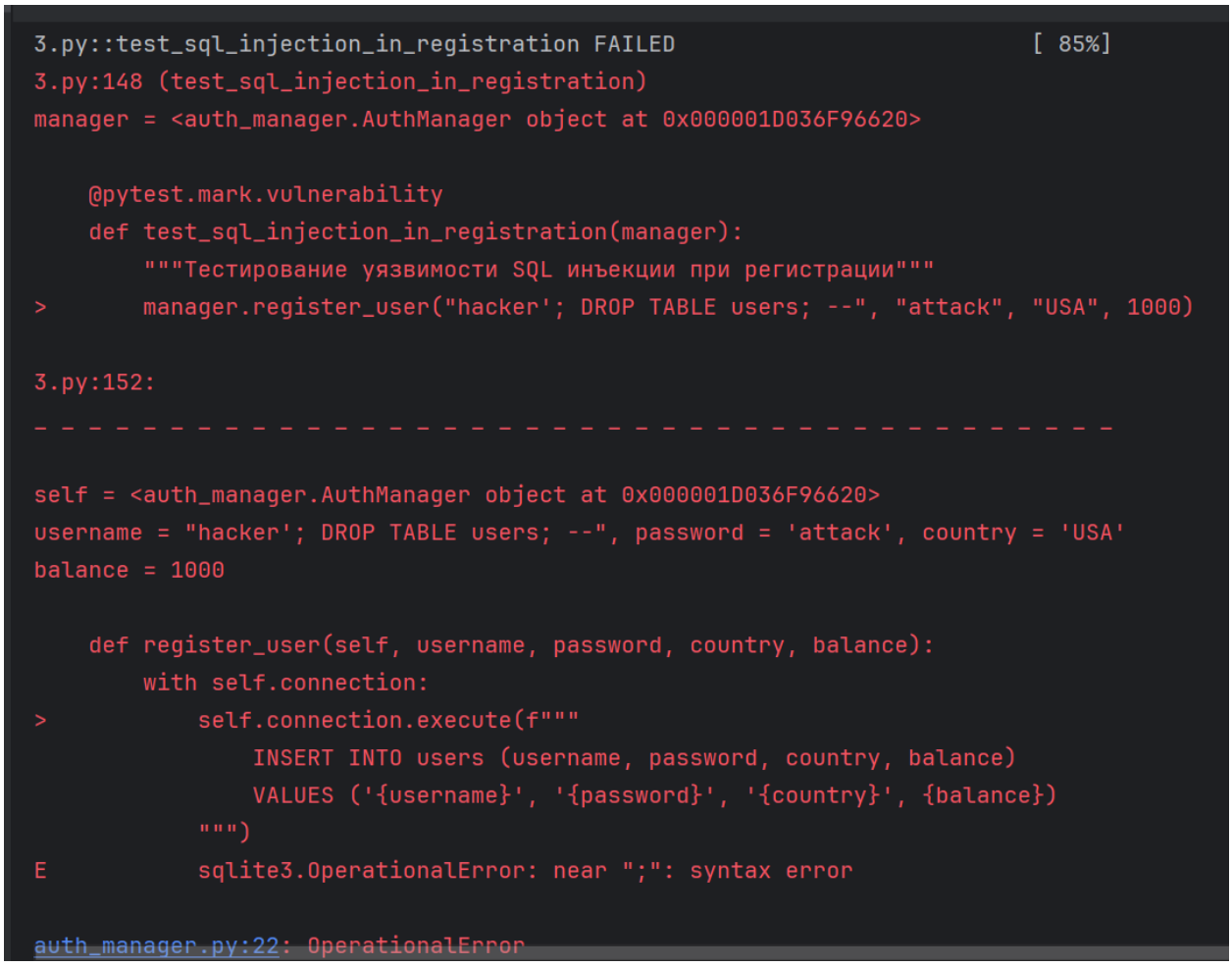


5.1-ТЕСТ-ФИКСТУРА

6.Тесты с метками:

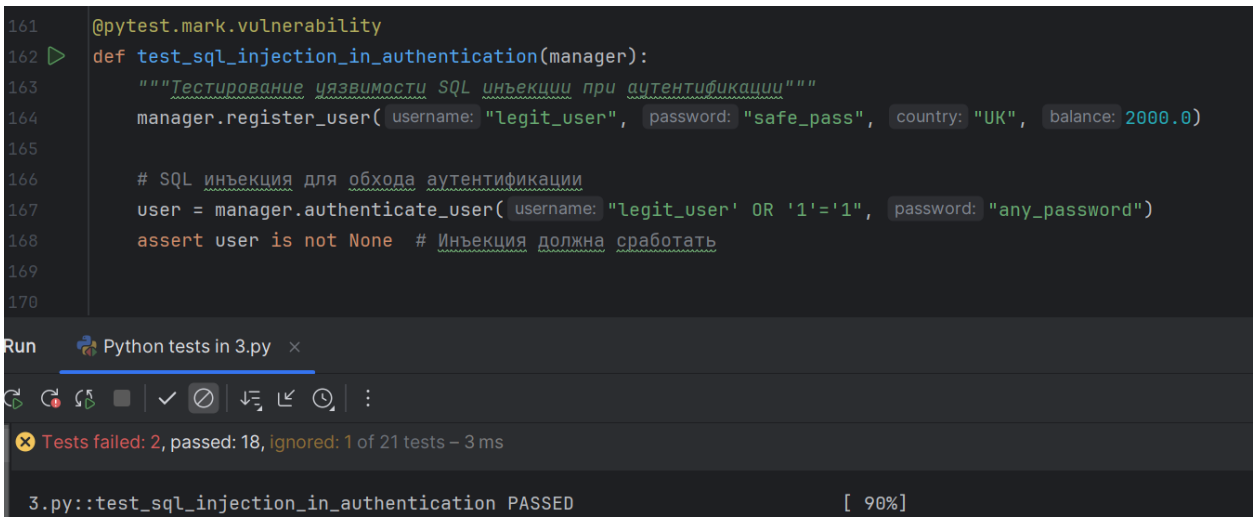
6.1-ТЕСТ-МЕТКА





6.2-ТЕСТ-МЕТКА

7.Тестирование различных векторов SQL-инъекций:



7.1-ТЕСТ-SQL-ИНЪЕКЦИЙ