Практическая работа №3

Код класса для тестирования:

```
import sqlite3

class AuthManager:
    def __init__(self, connection):
        self.connection = connection
        self.create_tables()

    def create_tables(self):
        with self.connection:
            self.connection.execute("""
                CREATE TABLE IF NOT EXISTS users (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    username TEXT NOT NULL UNIQUE,
                    password TEXT NOT NULL,
                    country TEXT NOT NULL,
                    balance REAL NOT NULL
                )
            """)

    def register_user(self, username, password, country, balance):
        with self.connection:
            self.connection.execute(f"""
                INSERT INTO users (username, password, country, balance)
                VALUES ('{username}', '{password}', '{country}', {balance})
            """)

    def authenticate_user(self, username, password):
        cursor = self.connection.cursor ()
        cursor.execute(f"""
            SELECT * FROM users
            WHERE username = '{username}' AND password = '{password}'
        """)
        return cursor.fetchone()

    def delete_user(self, user_id):
        with self.connection:
            self.connection.execute(f"""
                DELETE FROM users WHERE id = {user_id}
```

```
        """)

    def get_user_by_id(self, user_id):
        cursor = self.connection.cursor()
        cursor.execute(f"""
            SELECT * FROM users WHERE id = {user_id}
        """)
        return cursor.fetchone()

    def count_users_by_country(self, country):
        cursor = self.connection.cursor()
        cursor.execute(f"""
            SELECT COUNT(*) FROM users WHERE country = '{country}'
        """)
        return cursor.fetchone()[0]

    def transfer_balance(self, from_user_id, to_user_id, amount):
        with self.connection:
            # Проверяем, достаточно ли средств
            cursor = self.connection.cursor()
            cursor.execute(f"SELECT balance FROM users WHERE id =
{from_user_id}")
            from_balance = cursor.fetchone()[0]

            if from_balance < amount:
                raise ValueError("Insufficient funds")

            # Выполняем перевод
            self.connection.execute(f"""
                UPDATE users SET balance = balance - {amount} WHERE id =
{from_user_id}
            """)
            self.connection.execute(f"""
                UPDATE users SET balance = balance + {amount} WHERE id =
{to_user_id}
            """)
```

Базовые тесты:

```python
def test_register_user(auth_manager):
    """Тест регистрации пользователя"""
    auth_manager.register_user("testuser", "password", "Russia", 1000.0)

    cursor = auth_manager.connection.cursor()
    cursor.execute("SELECT * FROM users WHERE username = 'testuser'")
    user = cursor.fetchone()

    assert user is not None
    assert user[1] == "testuser"
    assert user[2] == "password"
    assert user[3] == "Russia"
    assert user[4] == 1000.0
```

Terminal Local

```
test_auth_manager.py::test_register_user PASSED
```

1-ТЕСТ

```python
def test_create_tables(auth_manager):
    """Тест создания таблиц"""
    cursor = auth_manager.connection.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users'")
    result = cursor.fetchone()
    assert result is not None
    assert result[0] == 'users'
```

Terminal Local

```
test_auth_manager.py::test_create_tables PASSED
```

2-ТЕСТ

```python
def test_authenticate_user_success(auth_manager):
    """Тест успешной аутентификации"""
    auth_manager.register_user("testuser", "password", "Russia", 1000.0)
    user = auth_manager.authenticate_user("testuser", "password")

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

Terminal Local

```
test_auth_manager.py::test_authenticate_user_success PASSED
```

3-ТЕСТ

Параметризованные тесты:

```python
# Параметризованные тесты (3 штуки)
@pytest.mark.parametrize("username, password, country, balance", [
    ("user1", "pass1", "Germany", 1200.0),
    ("user2", "pass2", "France", 800.0),
    ("user3", "pass3", "Spain", 1300.0),
])
def test_register_multiple_users(auth_manager, username, password, country, balance):
    """Параметризованный тест регистрации нескольких пользователей"""
    auth_manager.register_user(username, password, country, balance)
    user = auth_manager.authenticate_user(username, password)

    assert user is not None
    assert user[1] == username
    assert user[3] == country
    assert user[4] == balance
```

Terminal Local

```
test_auth_manager.py::test_register_multiple_users[user1-pass1-Germany-1200.0] PASSED
test_auth_manager.py::test_register_multiple_users[user2-pass2-France-800.0] PASSED
test_auth_manager.py::test_register_multiple_users[user3-pass3-Spain-1300.0] PASSED
```

4-ТЕСТ

```python
@pytest.mark.parametrize("username, password, expected", [
    ("nonexistent", "wrongpass", None),
    ("user1", "wrongpass", None),
    ("", "", None),
])
def test_authenticate_user_failure(populated_auth_manager, username, password, expected):
    """Параметризованный тес    т неуспешной аутентификации"""
    result = populated_auth_manager.authenticate_user(username, password)
    assert result == expected


@pytest.mark.parametrize("country, expected_count", [
    ("Russia", 2),
    ("USA", 1),
    ("Germany", 1),
    ("France", 0),
])
def test_count_users_by_country_parametrized(populated_auth_manager, country, expected_count):
    """Параметризованный тест подсчета пользователей по странам"""
    count = populated_auth_manager.count_users_by_country(country)
    assert count == expected_count

```

Terminal Local ×

```
test_auth_manager.py::test_authenticate_user_failure[nonexistent-wrongpass-None] PASSED
test_auth_manager.py::test_authenticate_user_failure[user1-wrongpass-None] PASSED
test_auth_manager.py::test_authenticate_user_failure[--None] PASSED
```

5-ТЕСТ

```python
@pytest.mark.parametrize("country, expected_count", [
    ("Russia", 2),
    ("USA", 1),
    ("Germany", 1),
    ("France", 0),
])
def test_count_users_by_country_parametrized(populated_auth_manager, country, expected_count):
    """Параметризованный тест подсчета пользователей по странам"""
    count = populated_auth_manager.count_users_by_country(country)
    assert count == expected_count

```

Terminal Local ×

```
test_auth_manager.py::test_count_users_by_country_parametrized[Russia-2] PASSED
test_auth_manager.py::test_count_users_by_country_parametrized[USA-1] PASSED
test_auth_manager.py::test_count_users_by_country_parametrized[Germany-1] PASSED
test_auth_manager.py::test_count_users_by_country_parametrized[France-0] PASSED
```

6-ТЕСТ

Тестирование исключений:

```python
# Тестирование исключений (2 штуки)
def test_transfer_balance_insufficient_funds(populated_auth_manager):
    """Тест перевода при недостаточных средствах"""
    with pytest.raises(ValueError, match="Insufficient funds"):
        populated_auth_manager.transfer_balance(2, 1, 2000.0)


def test_transfer_balance_nonexistent_user(populated_auth_manager):
    """Тест перевода несуществующему пользователю"""
    # Должно вызывать исключение из-за SQL-инъекционной уязвимости
    try:
        populated_auth_manager.transfer_balance(1, 999, 100.0)
        pytest.fail("Expected an exception for non-existent user")
    except Exception:
        # Ожидаем исключение из-за SQL-инъекции
        pass
```

```
Terminal  Local
test_auth_manager.py::test_transfer_balance_insufficient_funds PASSED
test_auth_manager.py::test_transfer_balance_nonexistent_user FAILED
```

7-ТЕСТ

```python
def test_transfer_balance_nonexistent_user(populated_auth_manager):
    """Тест перевода несуществующему пользователю"""
    # Должно вызывать исключение из-за SQL-инъекционной уязвимости
    try:
        populated_auth_manager.transfer_balance(1, 999, 100.0)
        pytest.fail("Expected an exception for non-existent user")
    except Exception:
        # Ожидаем исключение из-за SQL-инъекции
        pass
```

```
Terminal  Local
================================================================ FAILURES ================================================================
_____________________________________________ test_transfer_balance_nonexistent_user _____________________________________________

populated_auth_manager = <auth_manager.AuthManager object at 0x000001DBD4DD5300>

    def test_transfer_balance_nonexistent_user(populated_auth_manager):
        """Тест перевода несуществующему пользователю"""
        # Должно вызывать исключение из-за SQL-инъекционной уязвимости
        try:
            populated_auth_manager.transfer_balance(1, 999, 100.0)
>           pytest.fail("Expected an exception for non-existent user")
E           Failed: Expected an exception for non-existent user
```

8-ТЕСТ

Тесты с использованием фикстур базы данных:

```python
# Тесты с использованием фикстур базы данных
def test_delete_user(populated_auth_manager):
    """Тест удаления пользователя с использованием фикстур"""
    # Проверяем, что пользователь существует
    user_exists = populated_auth_manager.get_user_by_id(1)
    assert user_exists is not None

    # Удаляем пользователя
    populated_auth_manager.delete_user(1)

    # Проверяем, что пользователь удален
    user_deleted = populated_auth_manager.get_user_by_id(1)
    assert user_deleted is None
```

Terminal Local ×

```
test_auth_manager.py::test_delete_user PASSED
```

9-ТЕСТ

Тесты с метками:

```python
# Тесты с метками (минимум 2)
@pytest.mark.performance
def test_performance_large_number_of_users(auth_manager):
    """Тест производительности с большим количеством пользователей"""
    for i in range(100):
        auth_manager.register_user(f"user{i}", f"pass{i}", f"country{i % 10}", i + 100)

    count = auth_manager.count_users_by_country("country0")
```

Terminal Local × + ⌄

```
test_auth_manager.py::test_performance_large_number_of_users PASSED
```

10-ТЕСТ

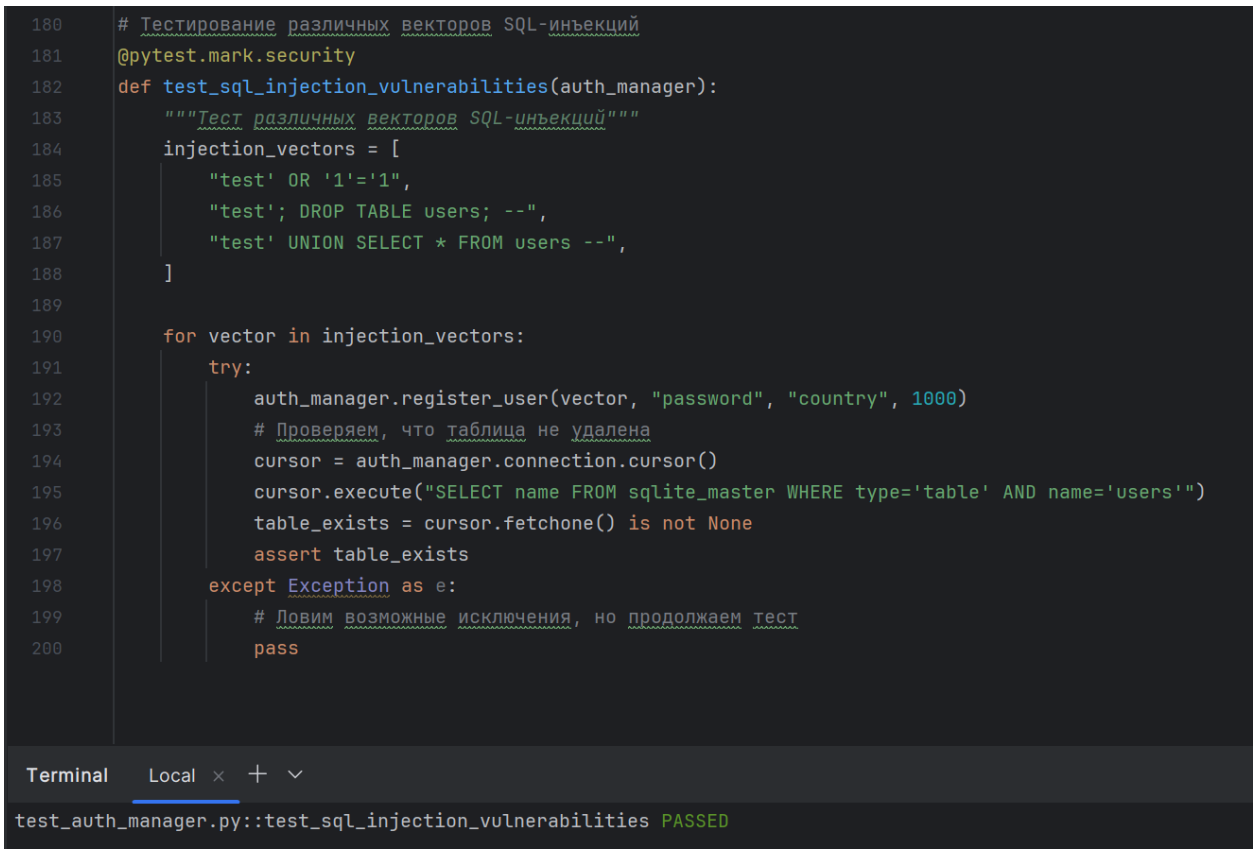
```
@pytest.mark.security
def test_sql_injection_register_user(auth_manager):
    """Тест SQL-инъекции при регистрации пользователя"""
    # Пробуем SQL-инъекцию через имя пользователя
    auth_manager.register_user("testuser'; DROP TABLE users; --", "password123", "Country", 1000)

    cursor = auth_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 остаться
```

```
Terminal    Local ×
test_auth_manager.py::test_sql_injection_register_user FAILED
```

11-ТЕСТ

```
________________________ test_sql_injection_register_user ________________________

auth_manager = <auth_manager.AuthManager object at 0x00000127637ED860>

    @pytest.mark.security
    def test_sql_injection_register_user(auth_manager):
        """Тест SQL-инъекции при регистрации пользователя"""
        # Пробуем SQL-инъекцию через имя пользователя
>       auth_manager.register_user("testuser'; DROP TABLE users; --", "password123", "Country", 1000)

test_auth_manager.py:153:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <auth_manager.AuthManager object at 0x00000127637ED860>, username = "testuser'; DROP TABLE users; --", password = 'password123'
country = 'Country', balance = 1000

    def register_user(self, username, password, country, balance):
        with self.connection:
>           self.connection.execute(f"""
                INSERT INTO users (username, password, country, balance)
                VALUES ('{username}', '{password}', '{country}', {balance})
            """)
E           sqlite3.OperationalError: near ";": syntax error
```

Тестирование различных векторов SQL-инъекций:


180 # Тестирование различных векторов SQL-инъекций
181 @pytest.mark.security
182 def test_sql_injection_vulnerabilities(auth_manager):
183 """Тест различных векторов SQL-инъекций"""
184 injection_vectors = [
185 "test' OR '1'='1",
186 "test'; DROP TABLE users; --",
187 "test' UNION SELECT * FROM users --",
188]
189
190 for vector in injection_vectors:
191 try:
192 auth_manager.register_user(vector, "password", "country", 1000)
193 # Проверяем, что таблица не удалена
194 cursor = auth_manager.connection.cursor()
195 cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users'")
196 table_exists = cursor.fetchone() is not None
197 assert table_exists
198 except Exception as e:
199 # Ловим возможные исключения, но продолжаем тест
200 pass
Terminal
Local
test_auth_manager.py::test_sql_injection_vulnerabilities PASSED

12-ТЕСТ