МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО

«Пермский государственный национальный исследовательский университет»

Механико-математический факультет

Кафедра информационной безопасности и систем связи

**Лабораторная работа №2 по дисциплине «Технологии разработки распределенных приложений»**

Специальность 02.03.02 Фундаментальная информатика и информационные технологии

|  |  |
| --- | --- |
| Выполнил:  Бродников Вадим Сергеевич, студент механико-математического факультета, группы ММ/З-2016НБ | Проверил:  Деменев Алексей Геннадьевич, доцент кафедры прикладной математики и информатики  механико-математического факультета, к.ф-м.н., доц. |

Пермь, 2021 г

Оглавление

[Термины и определения 3](#_Toc67076319)

[Введение 4](#_Toc67076320)

[Постановка задачи 5](#_Toc67076321)

[1 Проектирование 6](#_Toc67076322)

[2 Сценарии использования программы-клиента 14](#_Toc67076323)

[3 Тестирование 15](#_Toc67076324)

[Заключение 24](#_Toc67076325)

[Перечень использованных источников 25](#_Toc67076326)

**Термины и определения**

В настоящем отчете применяют следующие термины с соответствующими определениями:

**Серверное программное обеспечение (сервер)** - программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам.

**Клиент** - это программный компонент вычислительной системы, посылающий запросы серверу.

**Брокер** **сообщений** (очередь **сообщений**) – это отдельный сервис, который отвечает за хранение и доставку данных от сервисов-отправителей к сервисам-получателям с помощью модели Pub/Sub

**Введение**

**Цель:** изучение возможностей технологии передачи сообщений и сокетов для создания распределенных приложений. Сравнение методов реализации взаимодействия компонент распределенной системы.

**Задача:** реализовать распределенное приложение, в котором:

1. Веб-сервис и приложение, его использующее, должны быть разработаны на разных объектно-ориентированных языках программирования. Сервис обмена данными должен выполнять прием данных в нормализованную реляционную БД (например, спроектированную при выполнении входного контроля) из как минимум пять таблиц в 3-й нормальной форме.
2. Должно быть создано приложение, посылающее данные сервису при помощи сокетов и системы очередей сообщений, со свободной лицензией (Apache ActiveMQ, Apache Kafka или RabbitMQ), а при отсутствии такой возможности (соответствующих умений) допустимо использование импортных с бесплатной лицензией для университета в образовательных целях (например, MSMQ).
3. Данные перед передачей должны сжиматься и шифроваться при помощи ключа симметричного шифрования (DES).
4. Ключ симметричного шифрования должен передаваться сервису импорта для выполнения дешифрации данных.
5. При этом ключ симметричного шифрования должен в свою очередь шифроваться при помощи ключа асимметричного шифрования (RSA).
6. Ключ асимметричного шифрования должен генерироваться сервисом импорта и приложению должна передаваться открытая часть ключа.
7. Сервис импорта при получении данных должен импортировать их в БД при помощи механизма, реализованного при выполнении входного контроля.

**Постановка задачи**

Обеспечить проектирование, реализацию в виде программного кода, проверку и оценку созданного распределенного приложения по требованиям в следующей таблице:

***Распределение баллов за выполнение работы (***Максимальное количество баллов — 14)***:***

|  |  |
| --- | --- |
| **Требование к заданию** | **Максимальное количество баллов** |
| Приложение позволяет выполнять прием и передачу данных из ненормализованной БД в нормализованную без модификации данных. | 2 |
| Приложение позволяет передавать информацию с помощью очередей сообщений. | 3 |
| Приложение позволяет передавать информацию с помощью сокетов. | 3 |
| При передаче данных они шифруются с помощью симметричного ключа. | 2 |
| При передаче симметричного ключа шифрования данные шифруются с помощью ассиметричного ключа. | 2 |
| Приложение, написанное студентом, работает в сети без сбоев. | 2 |

1. **Проектирование**

Распределенное приложение, которое позволяет выполнить прием и передачу данных (базу данных), отправленных от клиента, использующее систему очереди сообщений RabbitMQ или с помощью сокетов и дальнейшая нормализация полученных данных и их запись в базу данных.

Очередь сообщений

RabbitMQ [1, 2, 3, 4] – брокер сообщений. Использует протокол AMQP [5] (Advanced Message Queuing Protocol), открытый протокол для передачи сообщений между компонентами системы. Для работы RabbitMQ Python3 [6] нужна библиотека pika [7, 8]. В терминологии RabbitMQ есть Producer(поставщик) – программа, отправляющая сообщения. В нашем случаем она реализована в client.py, написанной на языке программирования Python3 [6]. Queue (очередь) – имя, где хранятся сообщения. В нашем случае “data1”. Также есть Consumer(подписчик) – программа, принимающая сообщения. В нашем случае она реализована в server.py, написанной на языке программирования Python3 [6].

Сокеты

Сокет [9] — это программный интерфейс для обеспечения информационного обмена между процессами. Существует клиентские и серверные сокеты. Серверный сокет прослушивает определенный порт, в нашем случае, реализованном в server.py, написанном на языке программирования Python3. Клиентский сокет подключает к серверу, в нашем случае реализованном в client.py, написанном на языке программирования Python3. Сервер создает порт 9090(выбран самостоятельно). Данные(сокеты) передаются и принимаются по 16 байт

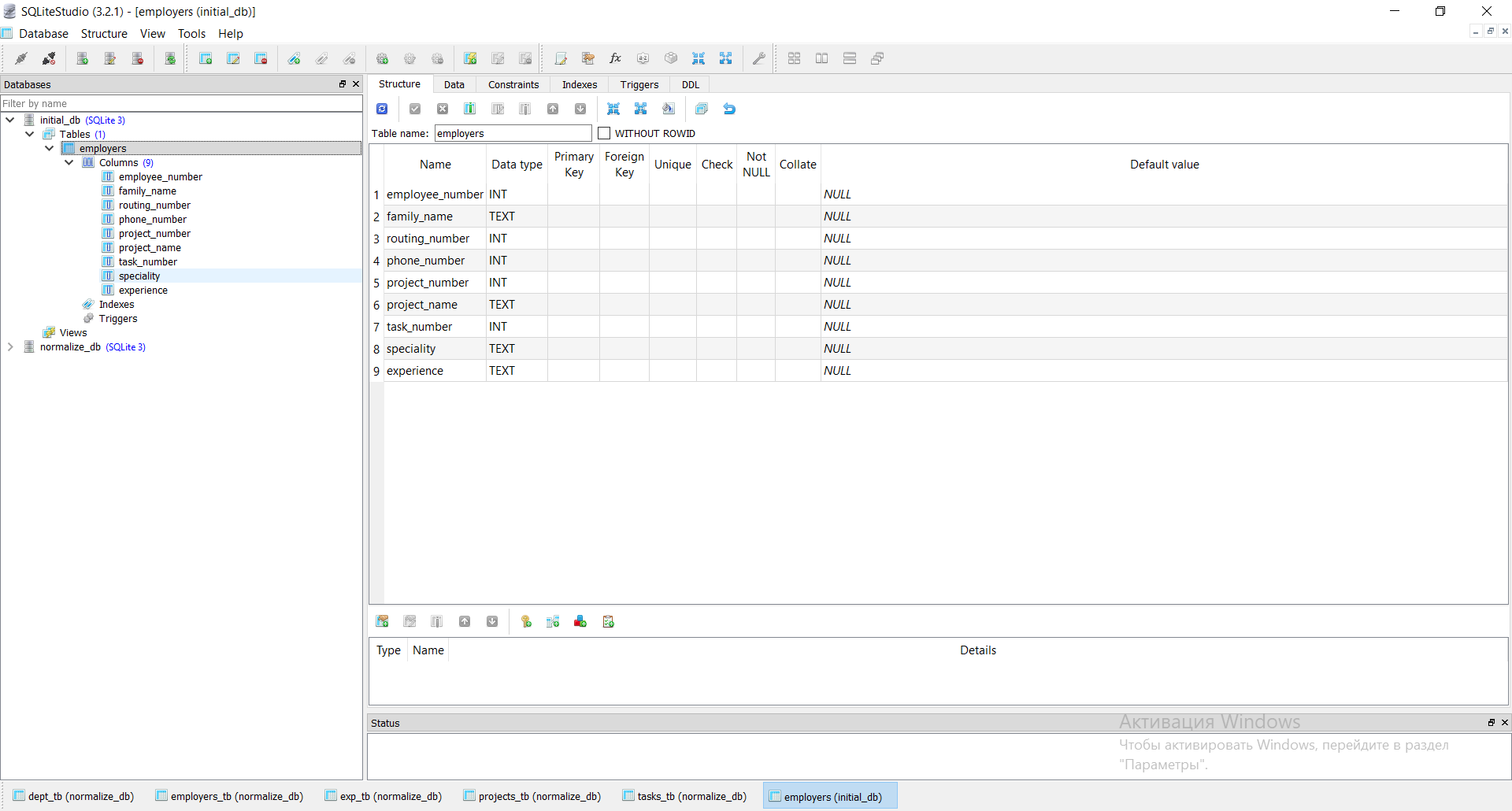
Шифрование DES и RSA

RSA [10] –криптографический алгоритм с открытым ключом. В приложении генерируется два ключа: публичный(открытый) и приватный(закрытый). DES [11]– это алгоритм симметричного ключа, основанный на сети Фейстеля. В качестве шифра с симметричным ключом он использует один и тот же ключ как для процессов шифрования, так и для дешифрования.

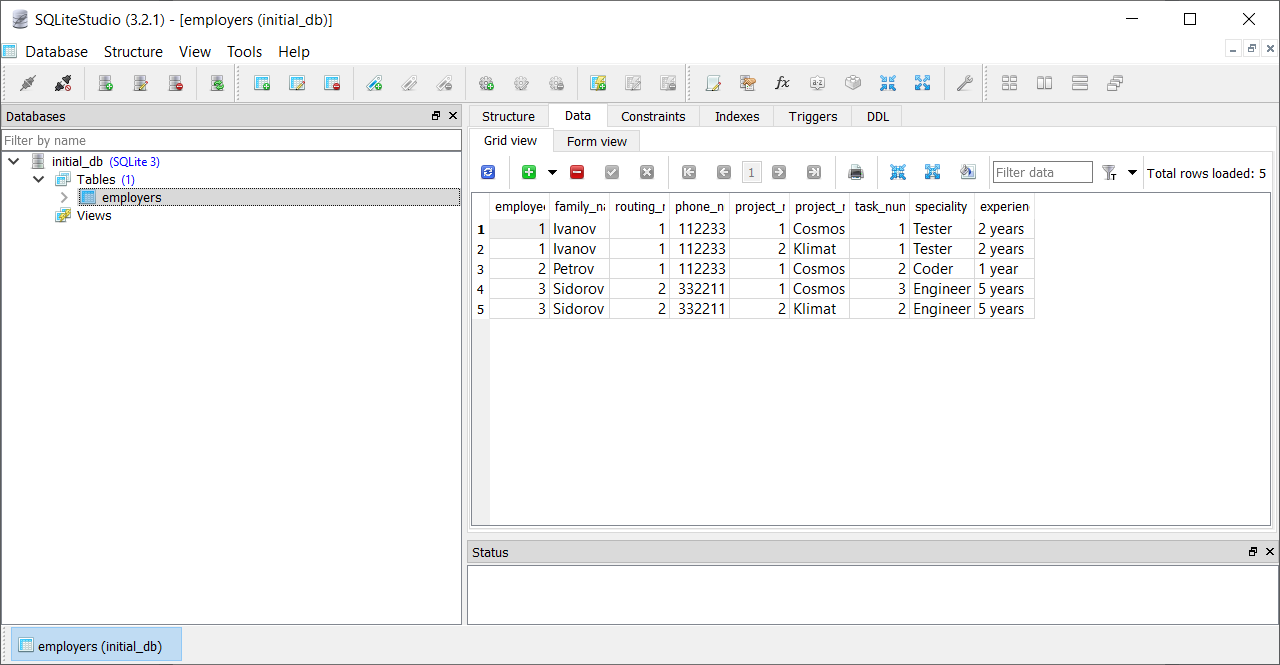
В приложении DES-ключ шифруется RSA-ключом. Реализовано в файле keys.py, написанном на языке программирования Python3. А DES шифрует данные, которые посылает клиент RabbitMQ и расшифровывает на сервере, полученные данные. С сокетами аналогично.

Ненормализованная БД и ее нормализация

Ненормализованная БД, созданная на языке SQLite3 [12,13] с названием “initial\_db.db” содержит таблицу employers. Приведена ниже:



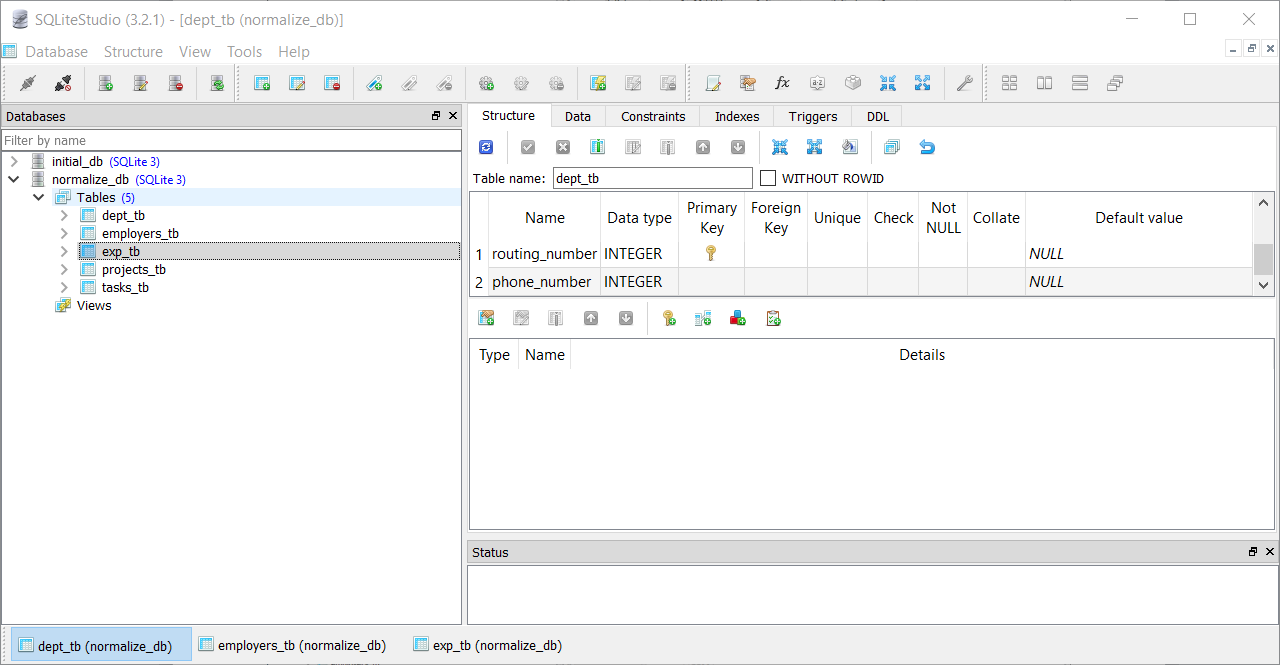
*Рис. 1 структура таблицы данных “employers” базы данных “initial\_db.db”.*



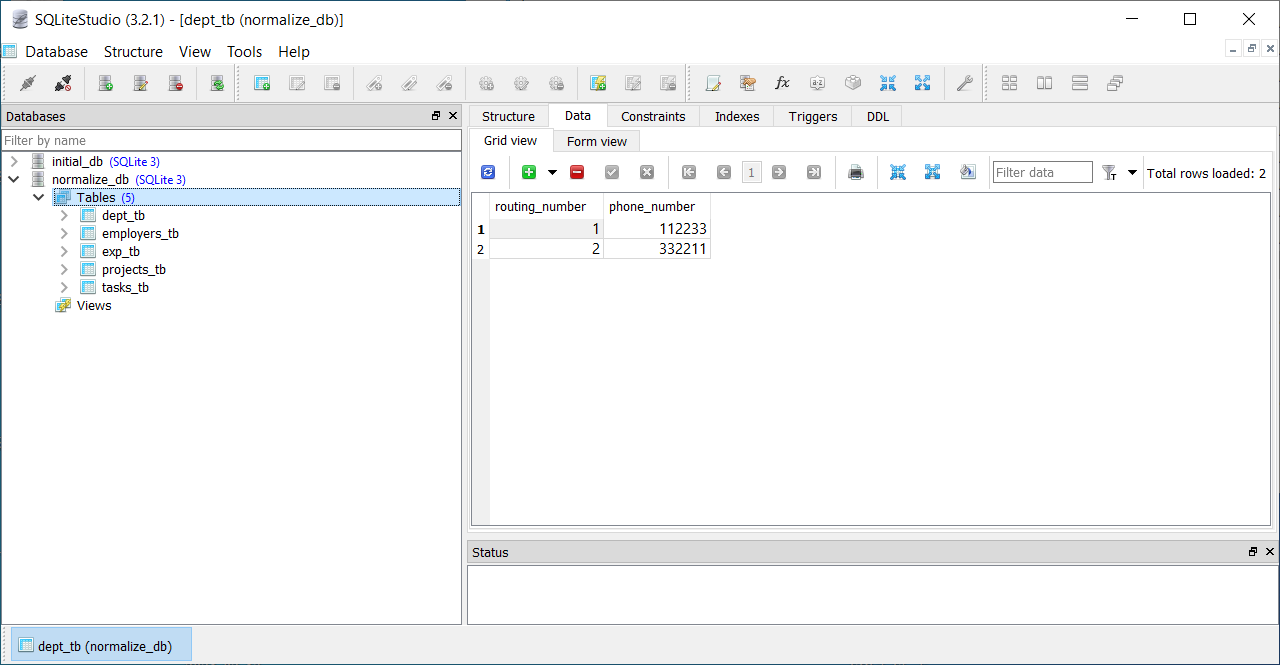
*Рис. 2 данные таблицы данных “employers” базы данных “initial\_db.db”.*

То есть у нас есть таблица из 9 столбцов: номер сотрудника(int значение), фамилия сотрудника(text значение), номер отдела, в котором работает сотрудник(int значение), номер телефона сотрудника(int значение), номер проекта(int значение), название проекта(text значение), номер задачи(int значение), должность сотрудника(text значение) и опыт работа сотрудника(text значение). Ключей нету.

В результате нормализации должны получится следующая база данных с 5 таблицами:

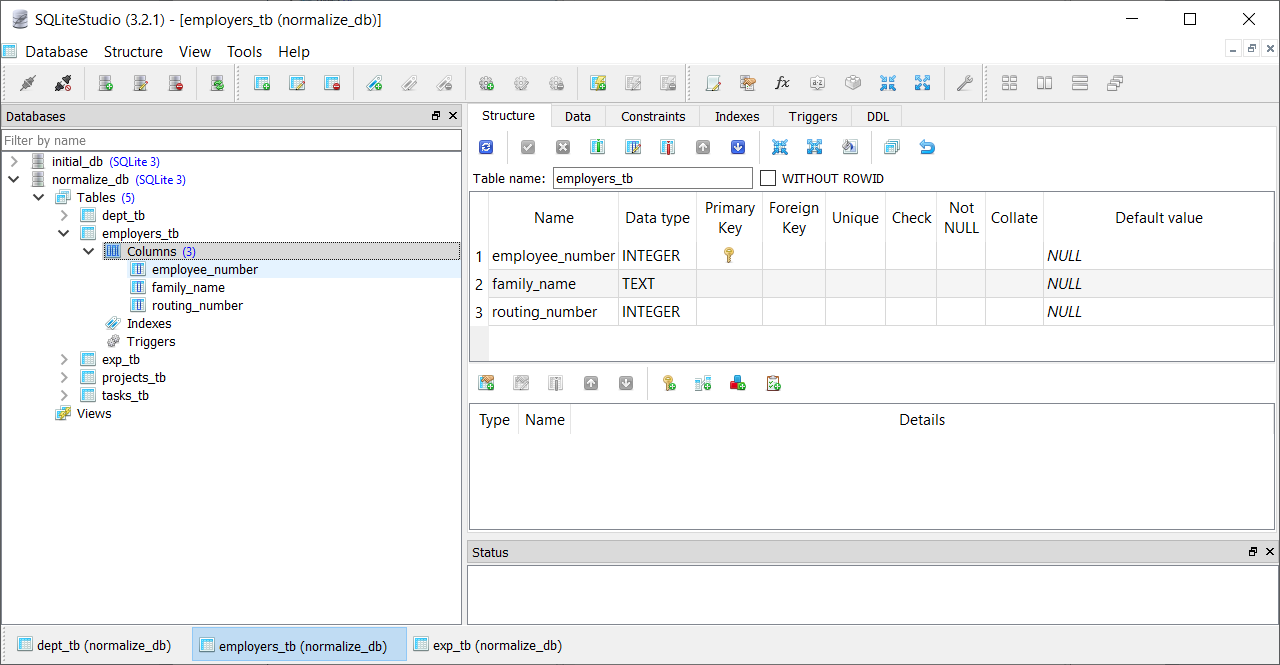


*Рис. 3 структура таблицы данных “dept\_tb” базы данных “normalize\_db.db”.*

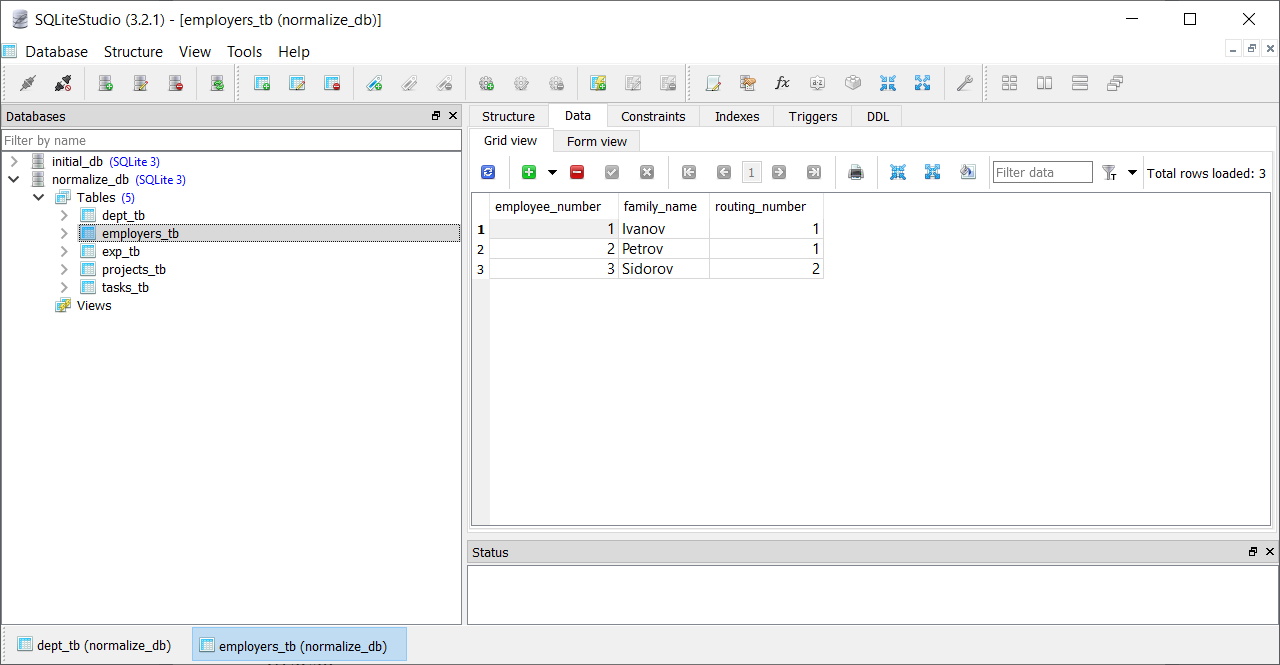


*Рис. 4 данные таблицы данных “dept\_tb” базы данных “normalize\_db.db”.*

Таблица Отделы. То есть номер отдела (int значение, привязан ключ) и номер телефона(int значение). Приведена ниже:

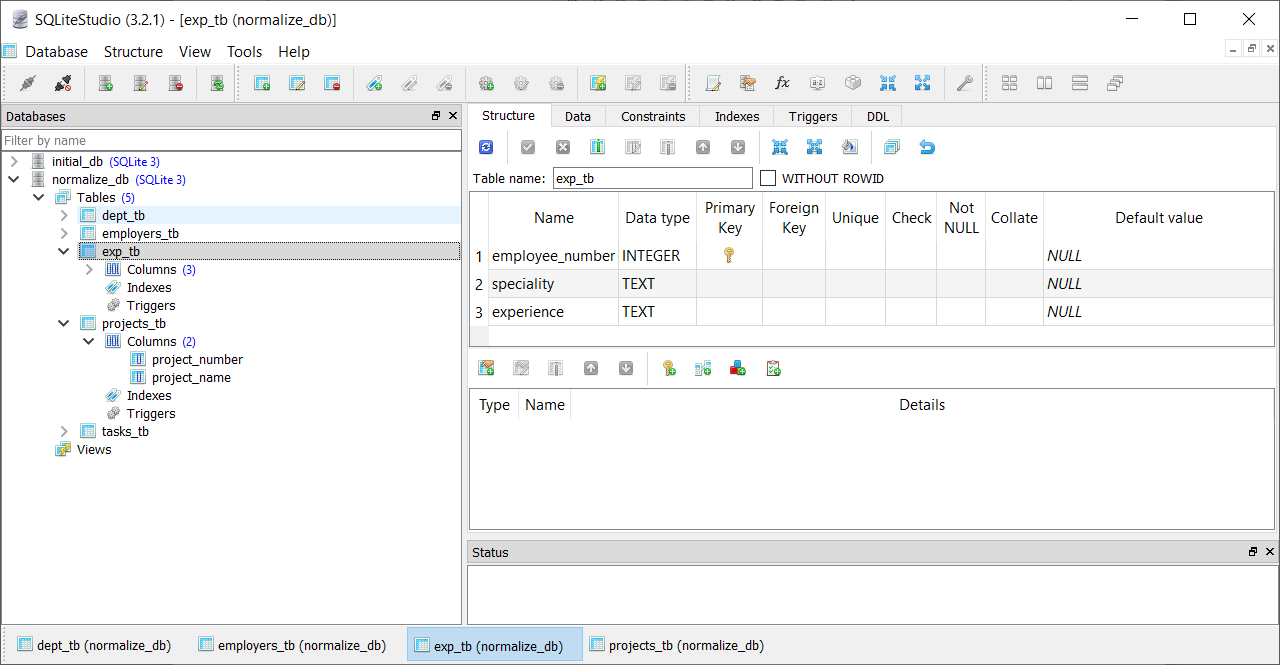


*Рис. 5 структура таблицы данных “ employers\_tb ” базы данных “normalize\_db.db”.*

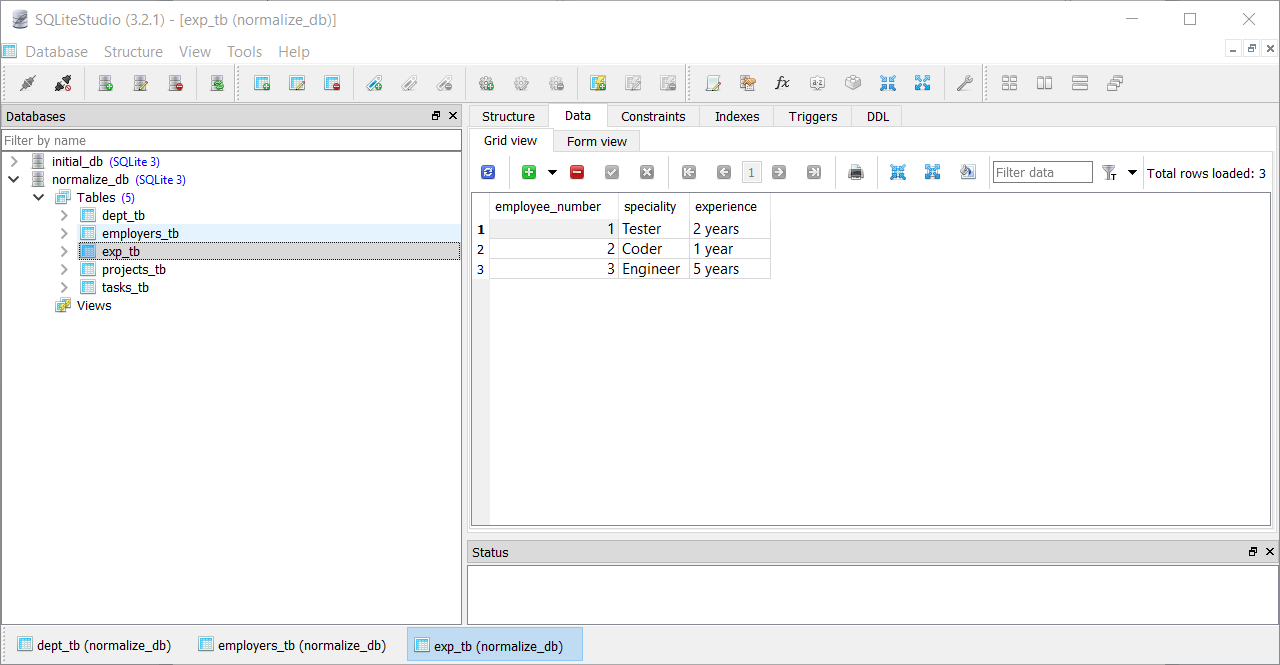


*Рис. 6 данные таблицы данных “employers\_tb” базы данных “normalize\_db.db”.*

Таблица Отделы-Сотрудники. То есть номер сотрудника (int значение, привязан ключ), фамилия сотрудника(text значение) и номер отдела(int значение). Приведена ниже:

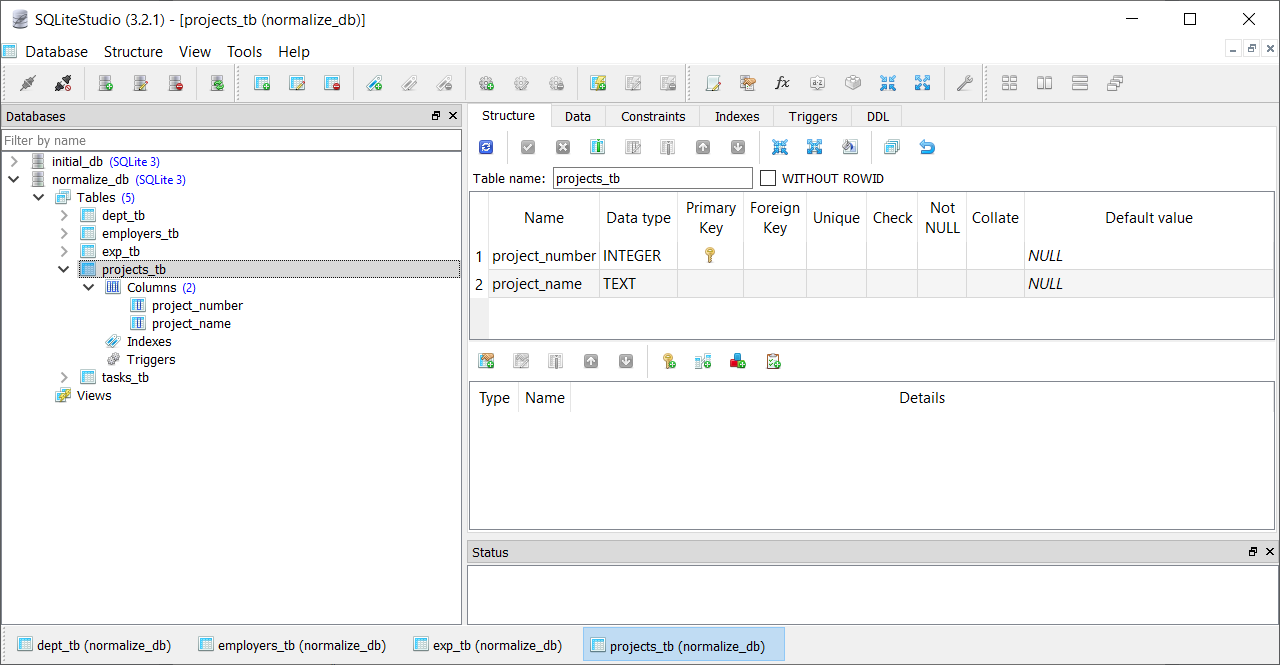


*Рис. 7 структура таблицы данных “exp\_tb” базы данных “normalize\_db.db”.*

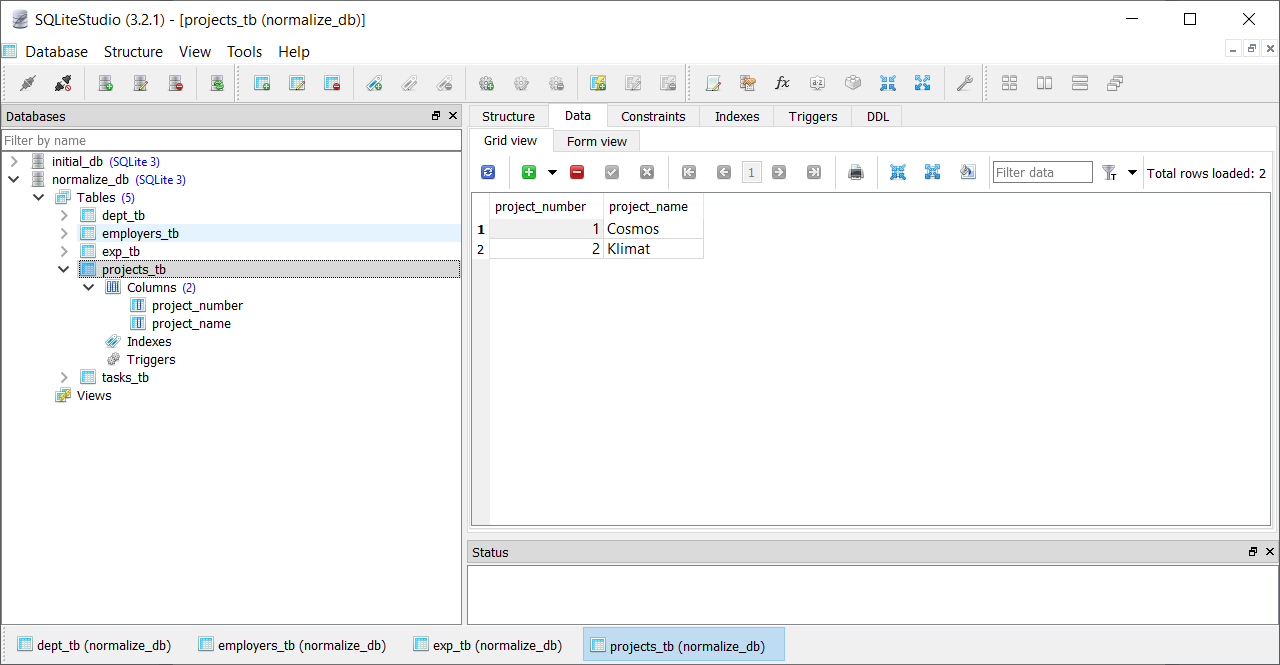


*Рис. 8 данные таблицы данных “exp\_tb” базы данных “normalize\_db.db”.*

Таблица Опыт сотрудников. То есть номер сотрудника (int значение, привязан ключ), специальность сотрудника(text значение) и опыт работы(text значение). Приведена ниже:

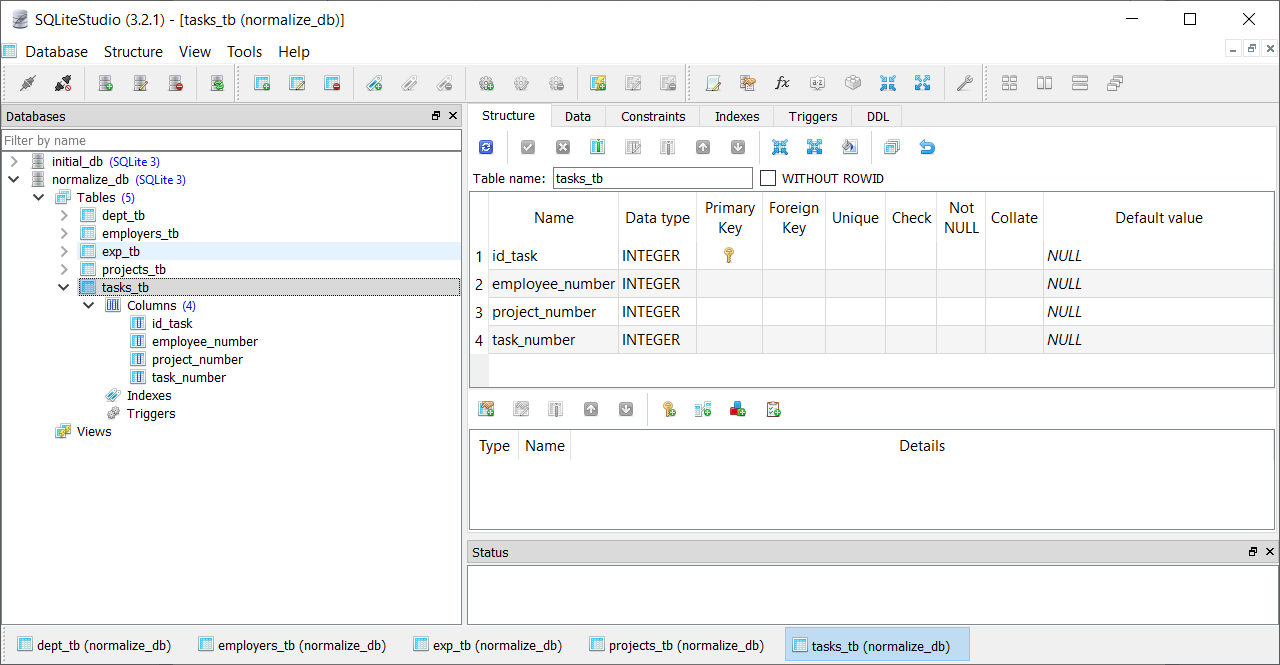


*Рис. 9 структура таблицы данных “projects\_tb” базы данных “normalize\_db.db”.*

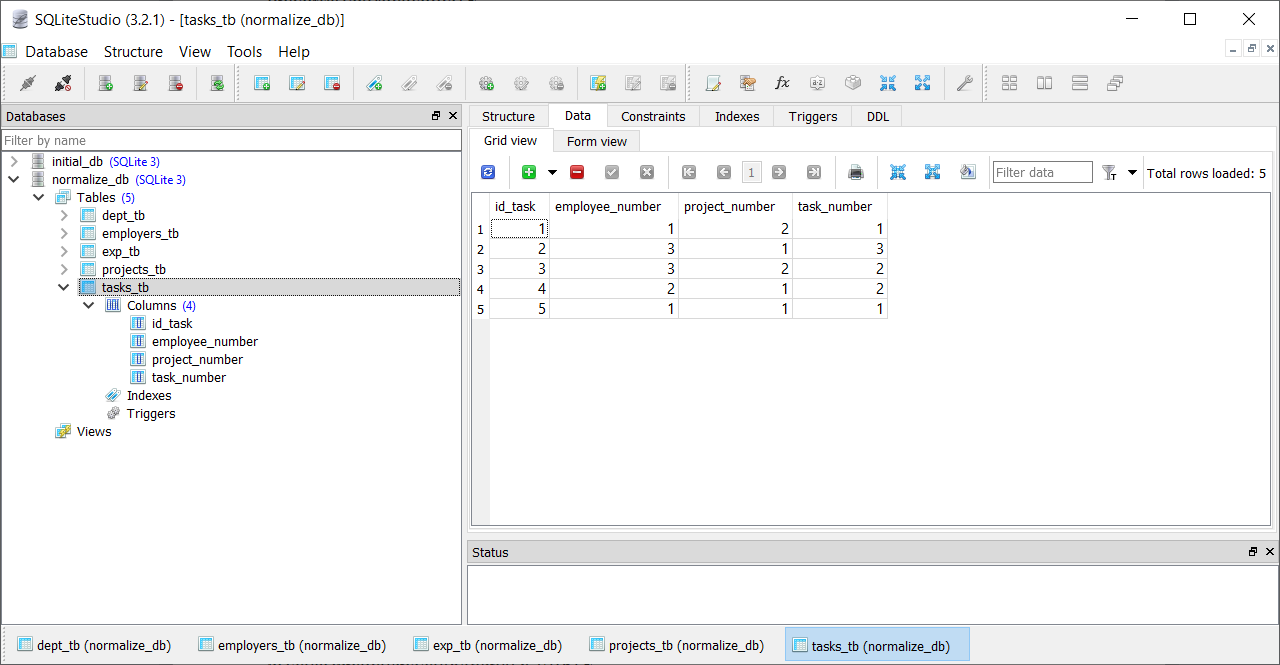


*Рис. 10 данные таблицы данных “projects\_tb” базы данных “normalize\_db.db”.*

Таблица Проекты. То есть номер проекта (int значение, привязан ключ) и название проекта(text значение).



*Рис. 11 структура таблицы данных “tasks\_tb” базы данных “normalize\_db.db”.*



*Рис. 12 – данные таблицы данных “tasks\_tb” базы данных “normalize\_db.db”.*

Таблица Задачи. То есть номер задачи (int значение, привязан ключ) и номер сотрудника (text значение), номер проекта (int значение) и номер подзадачи (int значение)

Для разработки клиентского и серверного приложения был выбран объектно-ориентированный язык программирования Python3 3.9.1 [6, 14]. В качестве среды разработки была выбрана среда Python IDLE [14].

**2 Сценарии использования программы-клиента**

Сценарий использования №1 – отправка данных через RabbitMQ

Предусловия: Сервер запущен

1. Пользователь вводит в командную строку через клавиатуру “1”, чтобы отправить ненормализованную базу данных через RabbitMQ
2. Ожидание получения результат

Ожидаемый результат: на серверном приложении распечатались нормализованные 5 таблиц. В текущей папке создалась база данных “normalize\_db.db”.

Сценарий использования №2 – отправка данных через сокеты

Предусловия: Сервер запущен

1. Пользователь вводит в командную строку через клавиатуру “2”, чтобы отправить ненормализованную базу данных через сокеты
2. Ожидание получения результат

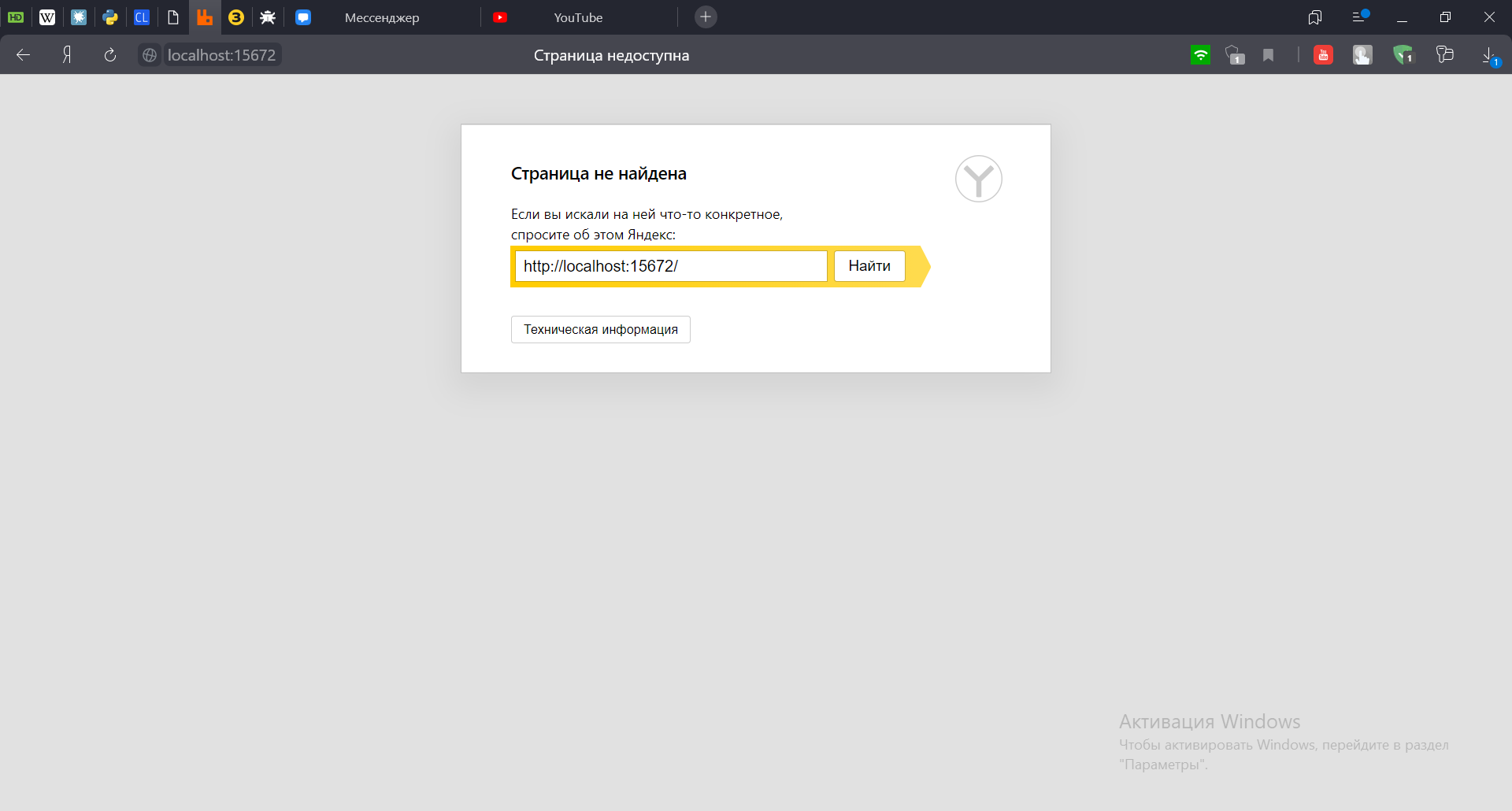
Ожидаемый результат: на серверном приложении распечатались нормализованные 5 таблиц. В текущей папке создалась база данных “normalize\_db.db”.

**3 Тестирование**

В результате разработки были созданы client.py – клиентское приложение, server.php – серверное приложение, normalize.py – вспомогательный файл для нормализации базы данных, keys.py – вспомогательный файл для генерации ключей, initial\_db.db – ненормализованная база данных.

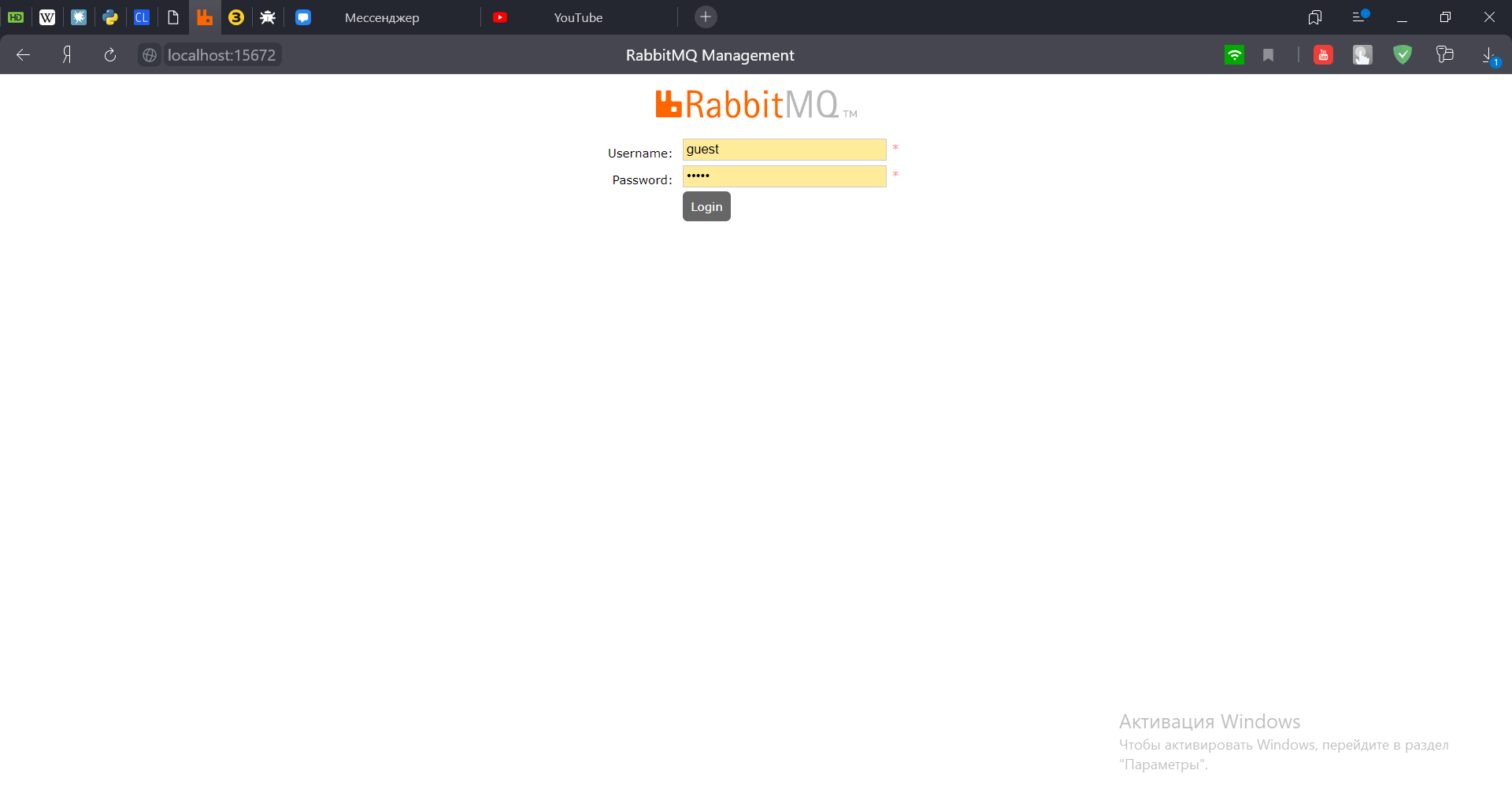
Сценарий №1

Перед началом тестирования выполним предусловие для сценария – запустим сервер RabbitMQ. Поэтому сначала остановим сервер RabbitMQ. Пуск – > RabbitMQ Server –> RabbitMQ Service – stop. Подтверждаем действие. Проверяем по адресу <http://localhost:15672> (визуальное отображение работы RabbitMQ Server):



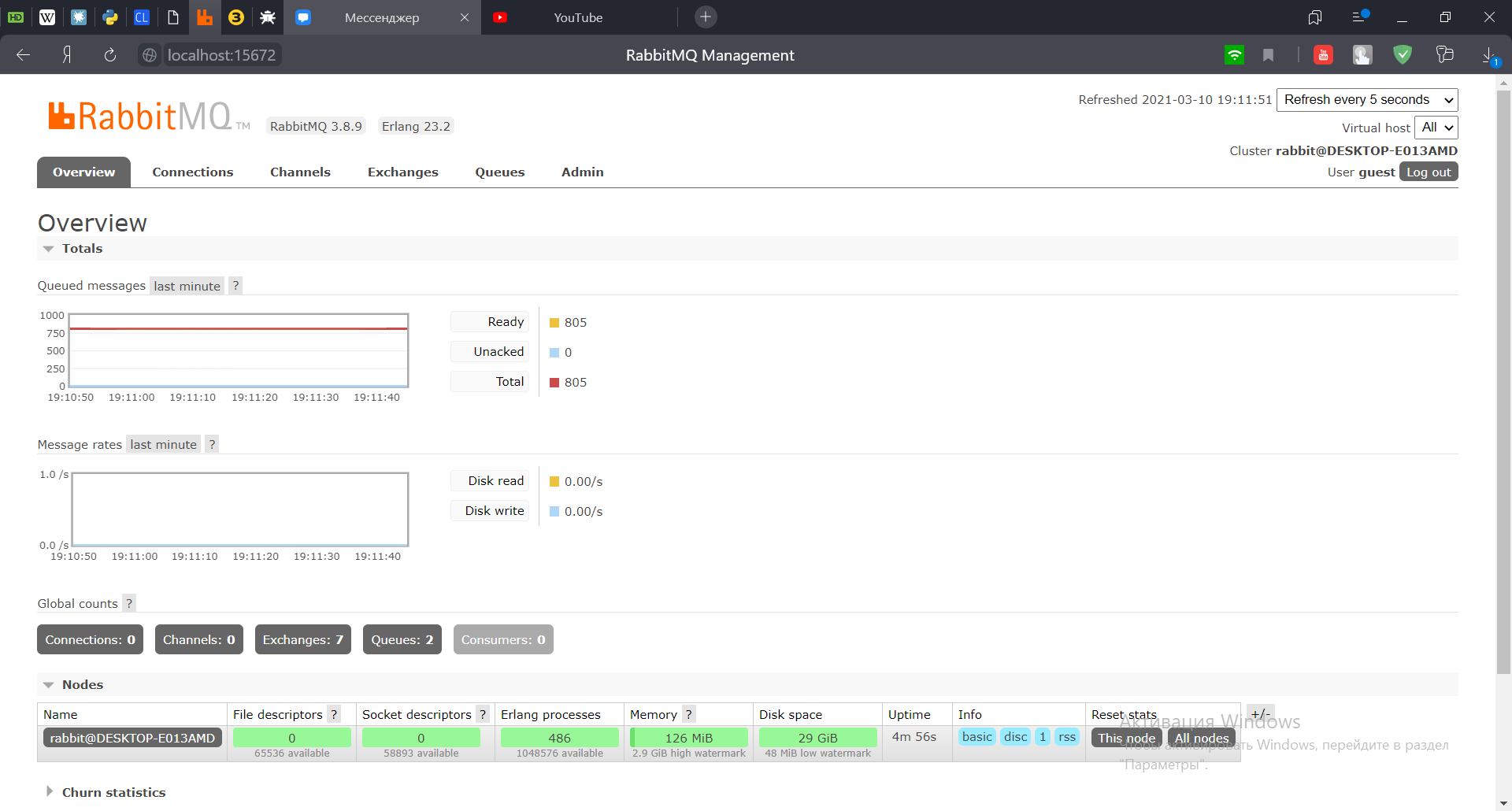
*Рис. 13 проверка сервера RabbitMQ на отключение*

Теперь запускаем сервер. Пуск –> RabbitMQ Server –> RabbitMQ Service – start. Подтверждаем действие. Проверяем по адресу http://localhost:15672 (визуальное отображение работы RabbitMQ Server):



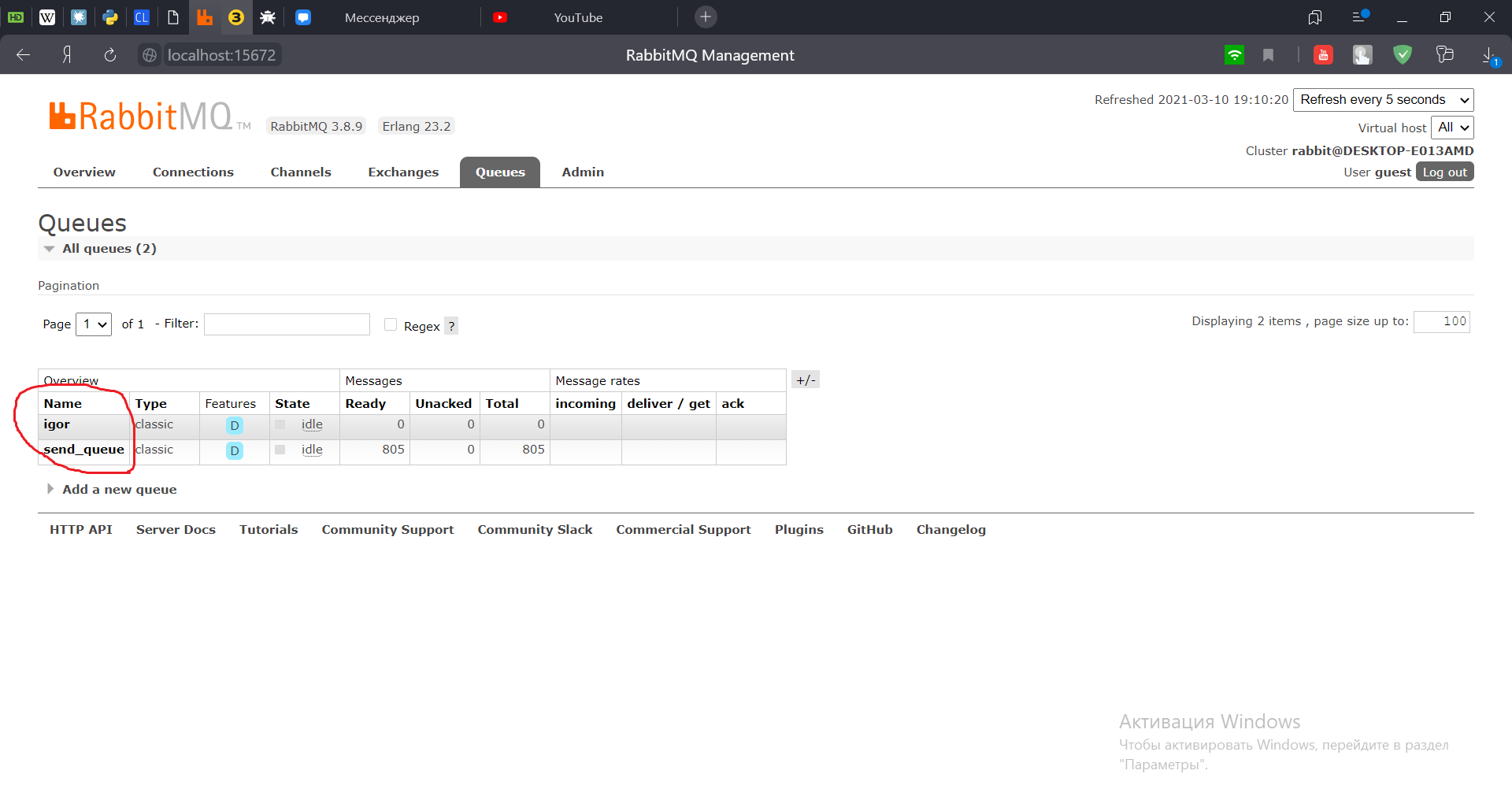
*Рис. 14 авторизация на сервере RabbitMQ*

Запрашивают авторизацию. Вводим логин и пароль по умолчанию: Username: guest, Password: guest.



*Рис. 15 управление RabbitMQ*

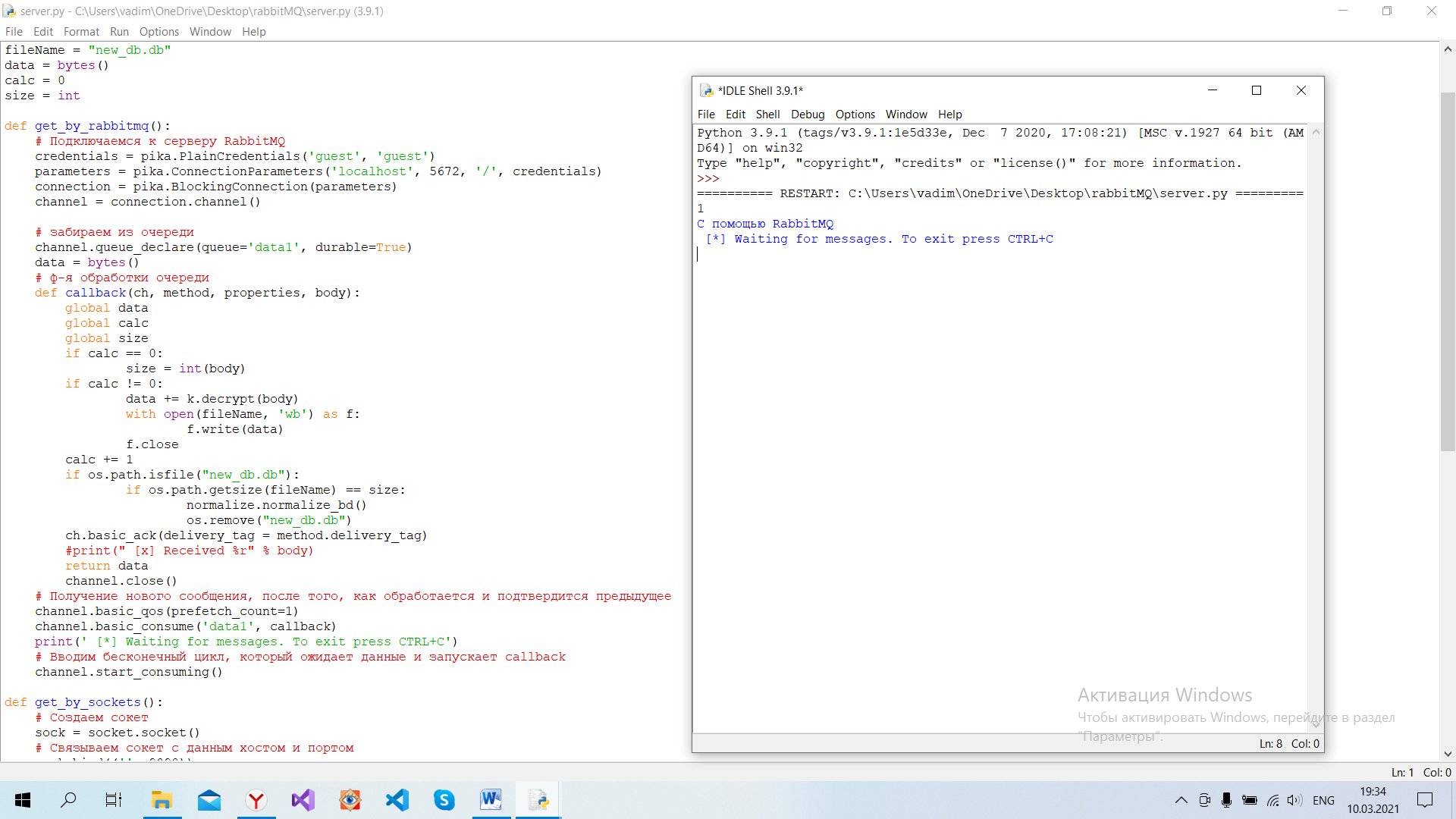
Переходим в раздел Queues для проверки существующей на данный момент очереди. Приведено ниже:



*Рис. 16 RabbitMQ раздел очереди*

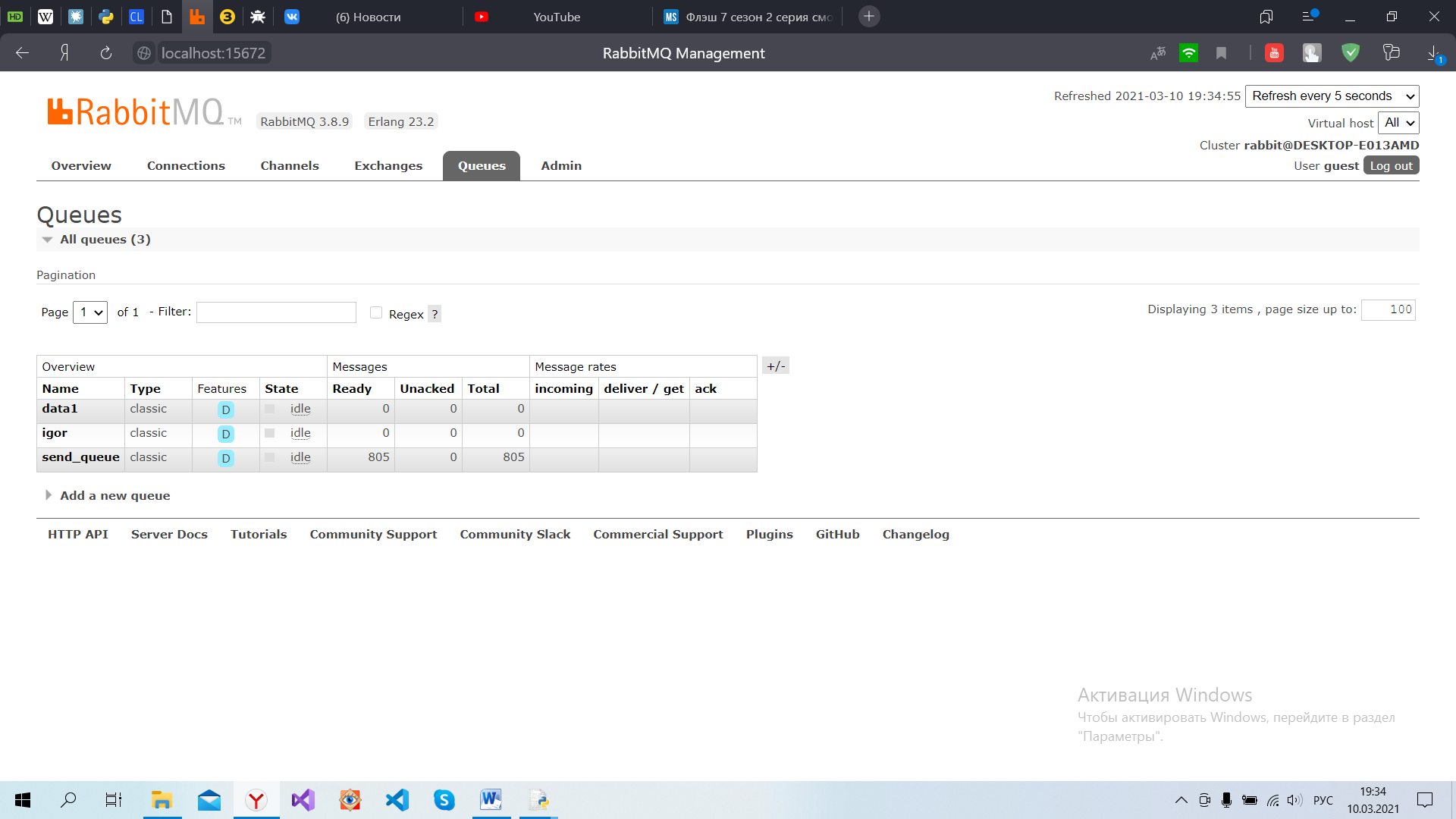
Проверяем, что нету очереди с названием “data1”. Мы будем использовать ее в приложении.

Запускаем server.py. Вводим через клавиатуру «1» в интерфейсе приложения



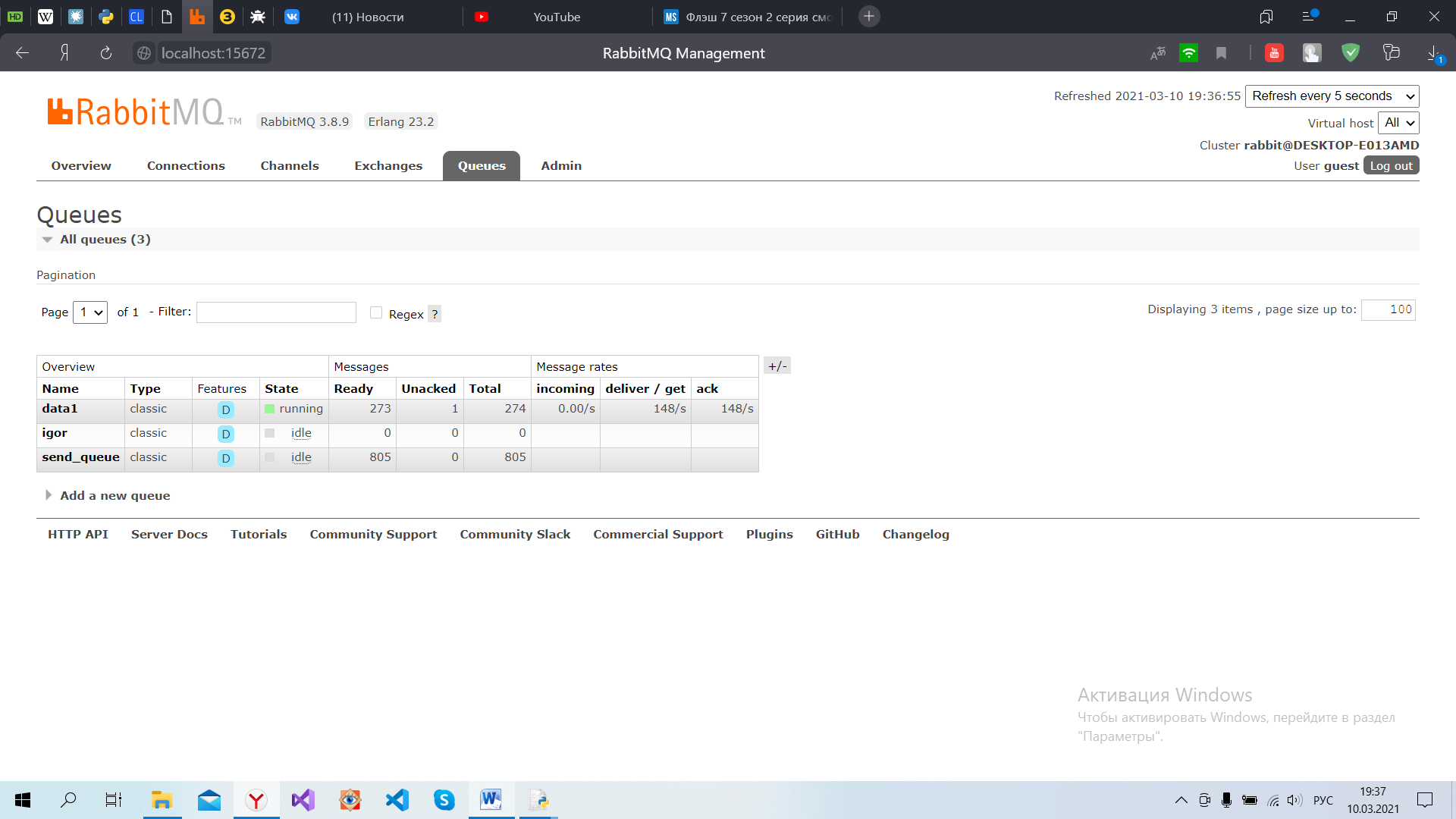
*Рис 17 тестирование приложения*

Теперь заходим на <http://localhost:15672>. Теперь у нас создалась очередь.



*Рис. 18 тестирование приложения*

Запускаем client.py. Вводим через клавиатуру «1» в интерфейсе



*Рис. 19 тестирование приложения*

На сервере статус очереди “data1” работающий. Значит, очередь используется приложением.

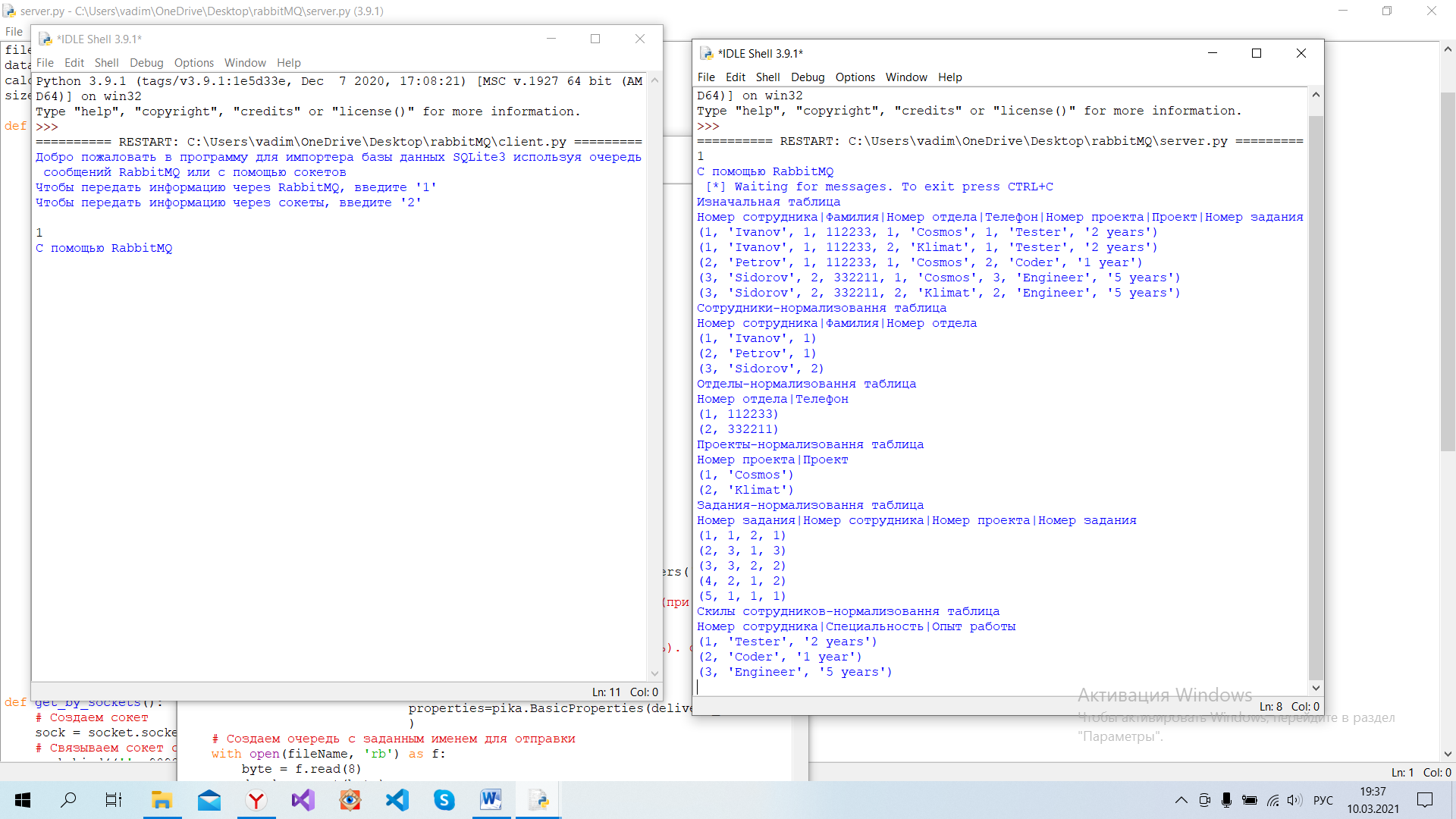


Рис. 20 тестирование приложения

Сервер выдает результаты в виде 5 таблиц, прочитанных из созданной базы данных “normalize\_db.db”.

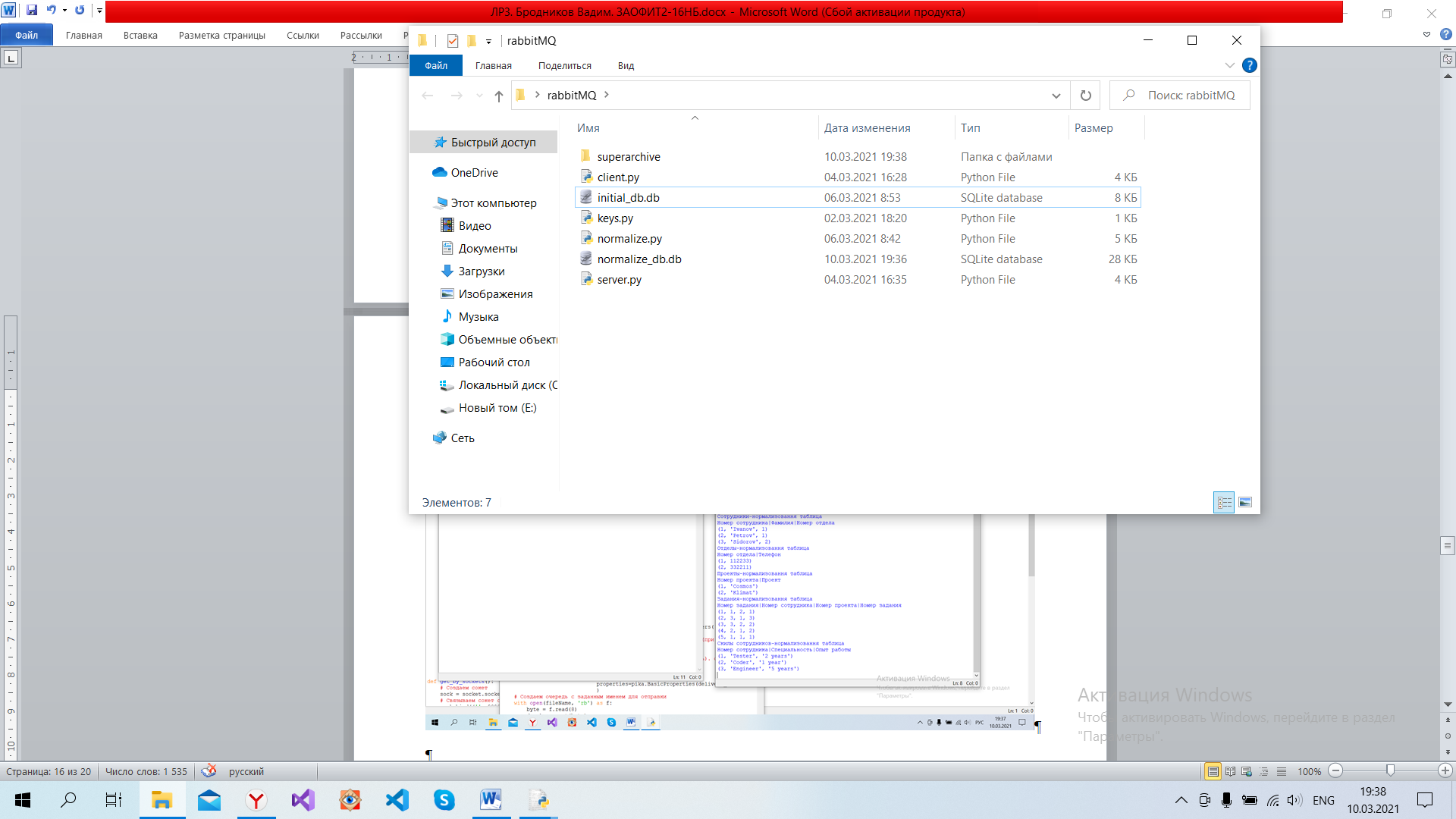


Рис. 21 тестирование приложения

В текущей папке появился файл с базой данной “normalize\_db.db”.

Тестирование прошло успешно.

Сценарий №2

Удаляем созданный после предыдущего сценария базу данных “normalize\_db.db”.

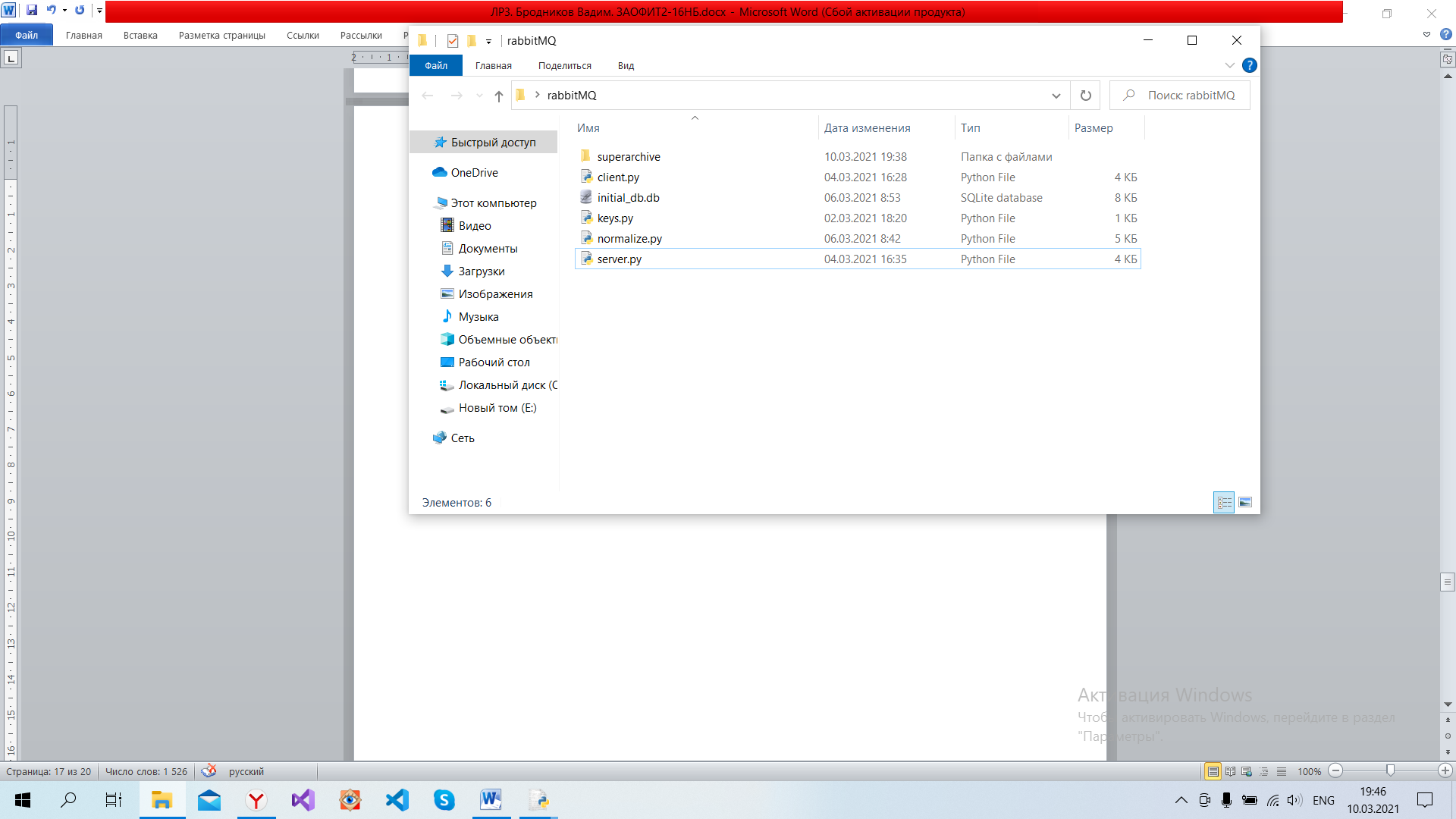


Рис. 22 тестирование приложения

Запускаем серверный сокет через server.py. Вводим через клавиатуру «2» в интерфейсе приложения.

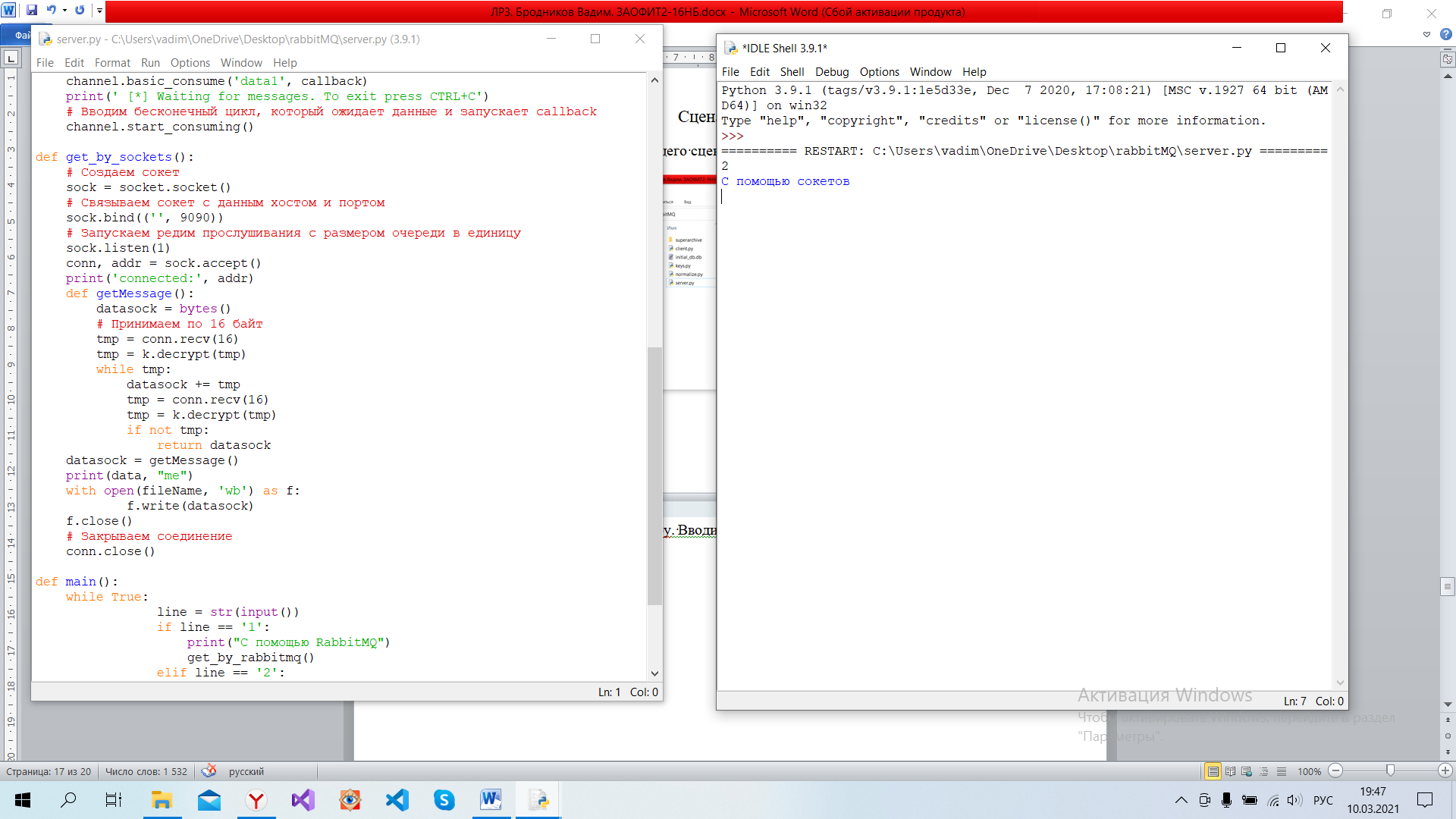
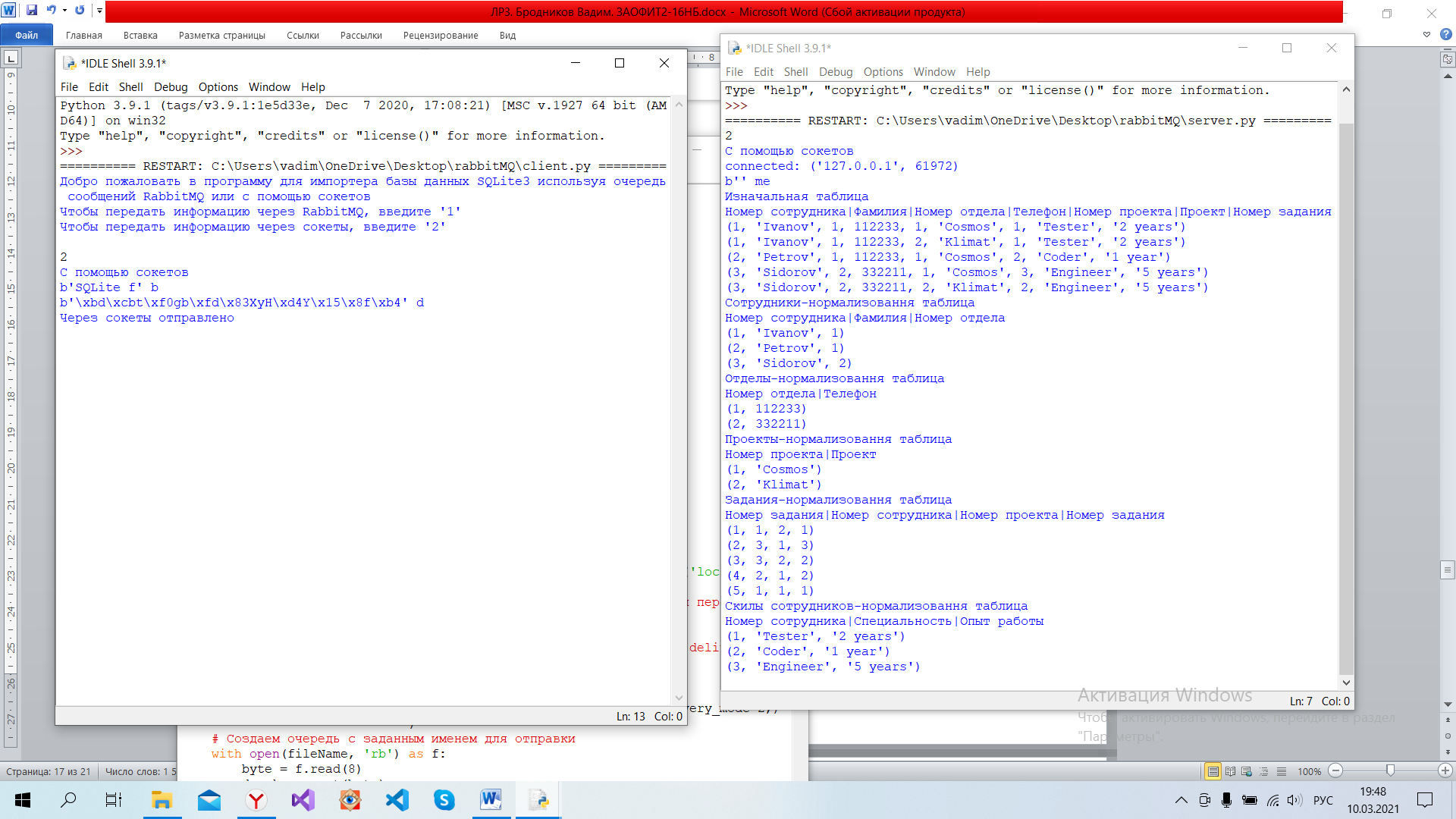


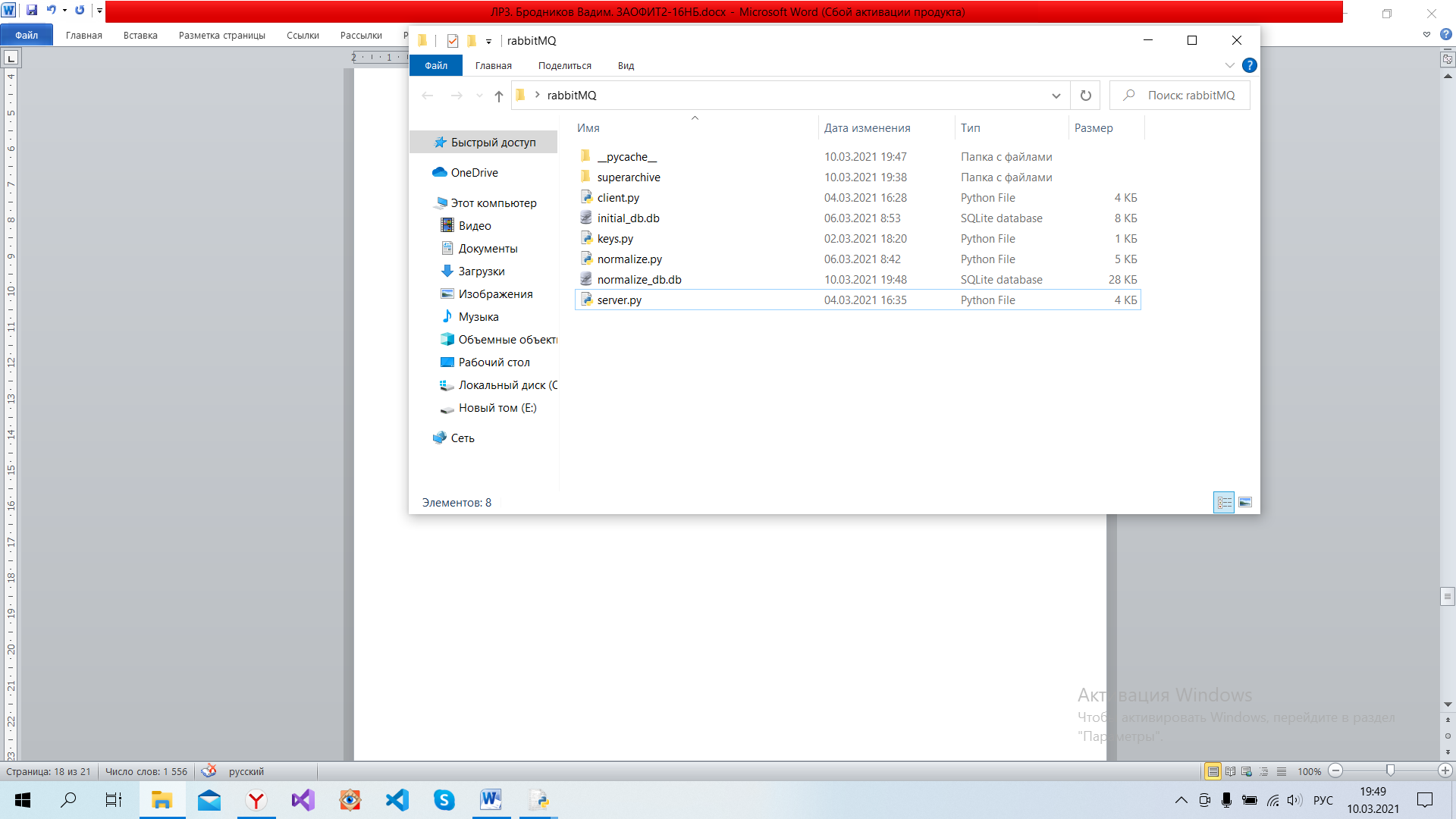
Рис. 23 тестирование приложения

Запускаем клиентский сокет через client.py. Вводим через клавиатуру «2».



*Рис. 24 тестирование приложения*

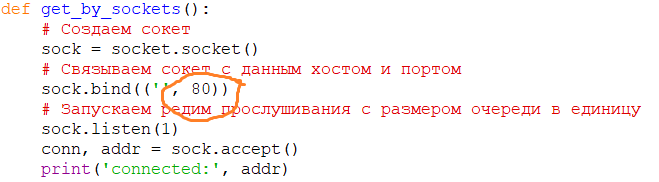
Сервер выдает результаты в виде 5 таблиц, прочитанных из созданной базы данных “normalize\_db.db”.



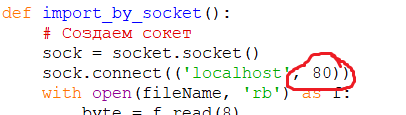
*Рис. 25 тестирование приложения*

В текущей папке появился “normalize\_db.db”.

Проверяем через приложение Wireshark [14] [15], что данные передались через сокеты. Для этого нам нужно указать в приложении Wireshark порт для захвата пакетов. Порт, в моем случае, то есть в реализации (client.py и server.py) указан порт 80:

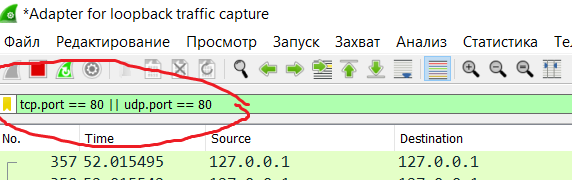


*Рис. 26 демонстрация установленного порта в server.py*



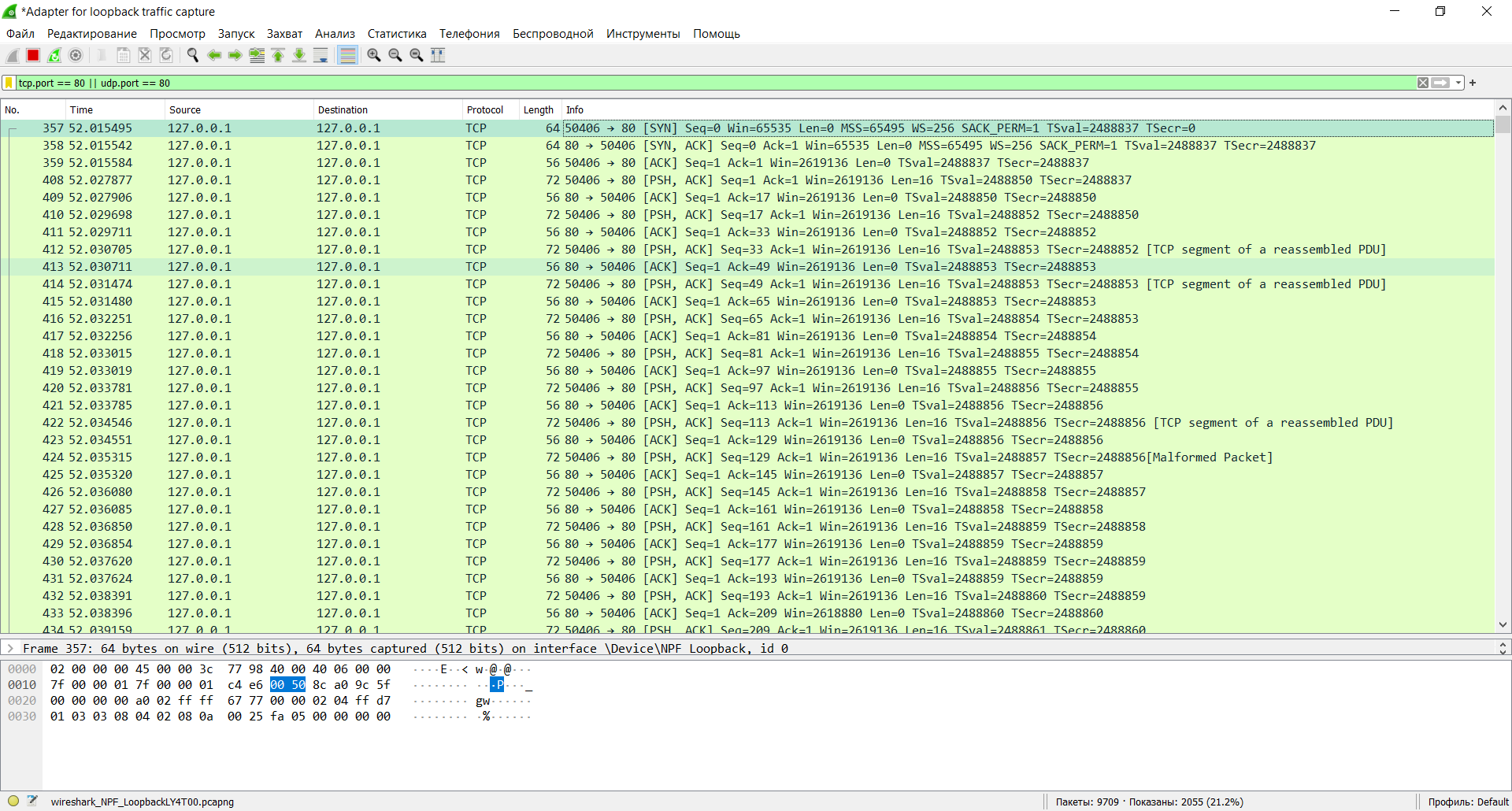
*Рис. 27 демонстрация установленного порта в client.py*

Значит, фильтруем под соответствующий порт:

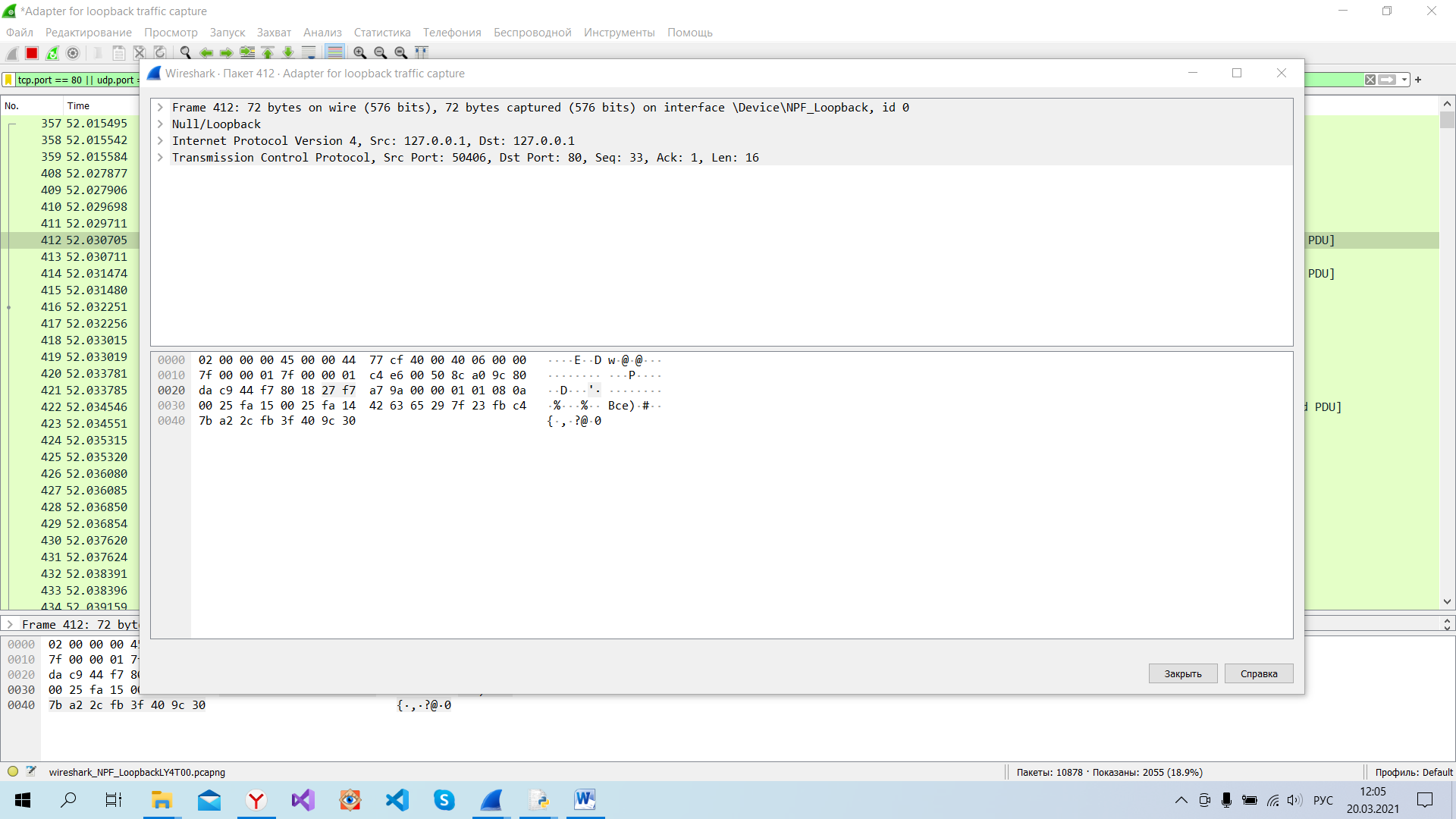


*Рис. 28 тестирование приложения*

В итоге программа зафиксировала пакеты по протоколу TCP:

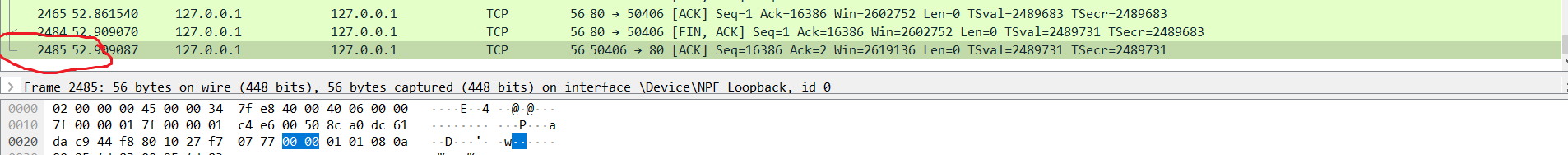


*Рис. 29 тестирование приложения*



*Рис. 30 демонстрация одного пакета под номером 412*

Всего пакетов получается 2485:



*Рис. 31 демонстрация приложения Wireshark*

Тестирование прошло успешно.

**Заключение**

В результате работы была достигнута цель и выполнены следующие требования:

1. Требование «Приложение позволяет выполнять прием и передачу данных из ненормализованной БД в нормализованную без модификации данных» выполнено полностью. Ожидаемое количество баллов: 2.
2. Требование «Приложение позволяет передавать информацию с помощью очередей сообщений» выполнено полностью через RabbitMQ. Ожидаемое количество баллов: 3.
3. Требование «Приложение позволяет передавать информацию с помощью сокетов» выполнено полностью. Ожидаемое количество баллов: 3.
4. Требование «При передаче данных они шифруются с помощью симметричного ключа» выполнено полностью. Ожидаемое количество баллов: 2.
5. Требование «При передаче симметричного ключа шифрования данные шифруются с помощью ассиметричного ключа» выполнено некорректно. Ожидаемое количество баллов: 0.
6. Требование «Приложение, написанное студентом, работает в сети без сбоев» выполнено полностью. Ожидаемое количество баллов: 2.

**Перечень использованных источников**

1. RabbitMQ tutorial 1 — Hello World [Электронный ресурс]. URL: <https://habr.com/ru/post/149694/> (Дата обращения: 07.03.2021)
2. Лицензия RabbitMQ [Электронный ресурс]. URL: <https://www.rabbitmq.com/mpl.html> (Дата обращения 07.03.2021)
3. Downloading and Installing RabbitMQ [Электронный ресурс]. URL: <https://www.rabbitmq.com/download.html> (Дата обращения 07.03.2021)
4. Erlang Programming Languege [Электронный ресурс]. URL: <https://www.erlang.org/downloads> (Дата обращения 07.03.2021)
5. AMQP [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/AMQP> (Дата обращения 07.03.2021)
6. Download Python [Электронный ресурс]. URL: <https://www.python.org/downloads/> (Дата обращения 07.03.2021)
7. Connecting to RabbitMQ – pika 0.9.6 documentation [Электронный ресурс]. URL: <https://pika.readthedocs.io/en/0.9.6/connecting.html> (Дата обращения 07.03.2021)
8. Лицензия Pika [Электронный ресурс]. URL: <https://github.com/pika/pika/blob/master/LICENSE> (Дата обращения 07.03.2021)
9. Сокеты в Python для начинающих [Электронный ресурс]. URL: <https://habr.com/ru/post/149077/> (Дата обращения 07.03.2021)
10. RSA шифрование в Python [Электронный ресурс]. URL: <https://blablacode.ru/programmirovanie/547> (Дата обращения 07.03.2021)
11. Что такое шифрование 3DES и как работает DES [Электронный ресурс]. URL: <https://heritage-offshore.com/informacionnoj-bezopasnosti/chto-takoe-shifrovanie-3des-i-kak-rabotaet-des/> (Дата обращения 07.03.2021)
12. Руководство по SQLite в Python [Электронный ресурс]. URL: <https://pythonru.com/osnovy/sqlite-v-python> (Дата обращения 07.03.2021)
13. Лицензия SQLite [Электронный ресурс]. URL: <https://www.sqlite.org/copyright.html> (Дата обращения 07.03.2021)
14. Спецификация Python IDLE 3.9.1 [Электронный ресурс]. URL: <https://docs.python.org/3.9/library/idle.html> (Дата обращения: 07.03.2021)
15. Wireshark. Go Deep [Электронный ресурс] URL: <https://www.wireshark.org/> (Дата обращения: 20.03.2021)
16. License Wireshark [Электронный ресурс] URL: <https://wiki.wireshark.org/License> (Дата обращения: 20.03.2021)