|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ | | | | | | | | | |
|  |  | | | |  | |  | |
| Пермский государственный национальный  исследовательский университет | | | | | | | | | |
|  | |  | | | |  | |
| ОТЧЕТ  по лабораторной работе №2  «Разработка распределенного приложения, использующего технологию передачи сообщений и сокеты»  по дисциплине  «Технологии разработки распределенных приложений» | | | | | | | |
|  | | |  |  | | | |
| Работу выполнил студент гр. КМБ-16  Дровосеков Даниил Алексеевич  01.12.2020 | | |  | Проверил  доцент кафедры прикладной математики и информатики,  к. ф.-м.н., доц.  Деменев Алексей Геннадьевич | | | |
|  | | |  |  | | | |
| Пермь, 2020 г. | | | | | | | | | |

**Введение**

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

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

1. 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, проверку созданного приложения требованиям в таблице:

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

Таблица 1. Требования к выполнению работы

|  |  |
| --- | --- |
| **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 асимметричного ключа. | 2 |
| Приложение, написанное студентом, работает в сети без сбоев. | 2 |

**Проектирование**

Предметная область баз данных – БД банка. Главная сущность – кредиты.

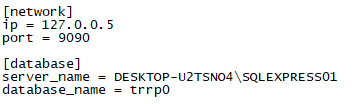


**Рис. 1.** Данные в ненормализованном виде

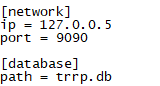
Ненормализованная таблица содержится в базе данных SQLite 3.33.0 [1,2]. Приложения для экспорта и импорта данных разрабатываются на языке программирования Python 3.7 [3,4]. Для создания приложений применяется среда разработки Jupyter Notebook 7.2 [5,6]. Таблицы в 3-й нормальной форме располагаются в базе данных MS SQL Server 2012 Express [7]. Для ассиметричного шифрования применяется модуль «RSA» в составе python-библиотеки «Crypto» [8]. Для симметричного шифрования применяется модуль «DES» в составе python-библиотеки «Crypto» [8].

**Реализация и проверка**

Приложение для импорта данных выступает в роли сервера, открывая сокет и ожидая подключения клиента. Приложение для экспорта данных выступает в роли клиента, подключаясь посредством сокета к серверу. Данные о ip-адресе и номере порта содержатся в файлах «imp\_config.ini» и «exp\_config.ini» для сервера и клиента соответственно.



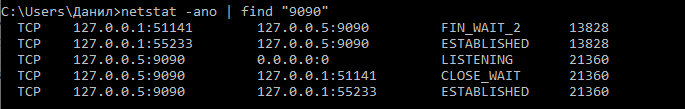
**Рис. 2.** Содержимое файла «imp\_config.ini»



**Рис. 3.** Содержимое файла «exp\_config.ini»

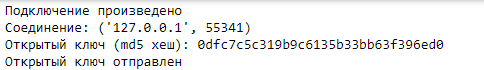
Чтобы убедиться в том, что программы используют для подключения сокеты, воспользуемся утилитой командной строки Windows «nestat» со следующими ключами:

* -a: Отображает все активные TCP-подключения и порты TCP и UDP, прослушиваемые компьютером
* -n: Отображает активные TCP-подключения, однако адреса и номера портов выражаются в числовом виде, и для определения имен не выполняется никаких попыток
* -o: Отображает активные TCP-подключения и включает идентификатор процесса (PID) для каждого подключения. Приложение, основанное на PID, можно найти на вкладке процессы в диспетчере задач Windows



**Рис. 4.** Вывод команды «netstat»

При установке соединения сервер генерирует пару открытый-закрытый ключ RSA и отравляет открытый ключ клиенту.

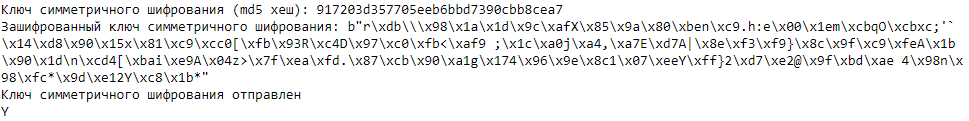


**Рис. 5.** Генерация и отправка открытого ключа RSA сервером

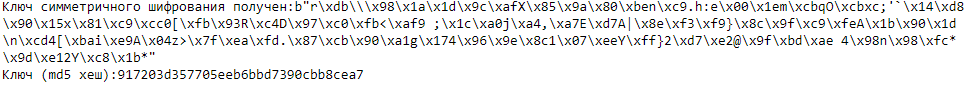


**Рис. 6.** Получение открытого ключа RSA клиентом

Затем клиент генерирует ключ шифрования DES, шифрует его открытым ключом RSA и посылает серверу. Сервер получает зашифрованный ключ и дешифрует его своим закрытым ключом.

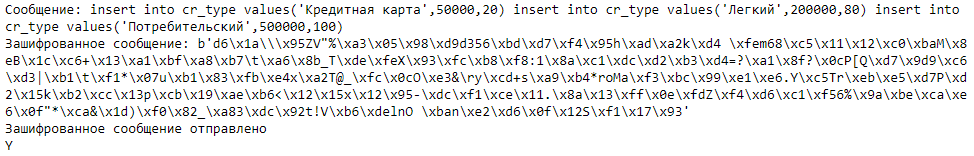


**Рис. 7.** Генерация, шифрование и отправка ключа DES клиентом

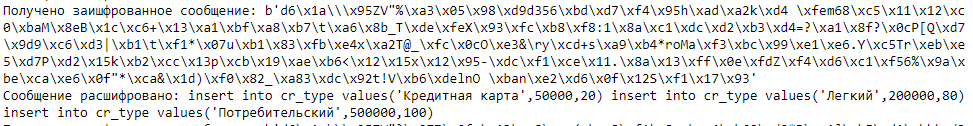


**Рис. 8.** Получение и дешифровка ключа DES сервером

Далее данные пересылаются от клиента серверу в зашифрованном виде.

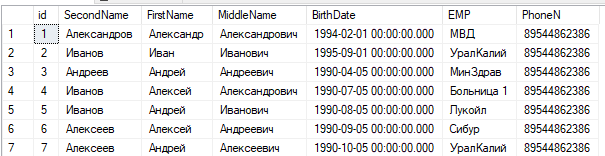


**Рис. 9.** Шифрование и отправка сообщения серверу клиентом

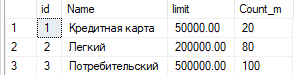


**Рис. 10.** Получение и расшифровка зашифрованного сообщения сервером

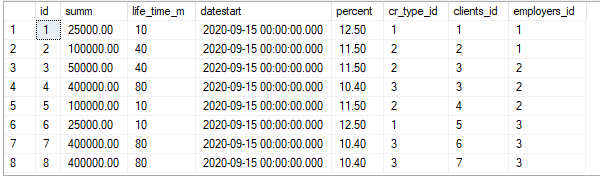
После окончания работы программ данные будут находиться в пяти таблицах в 3-й нормальной форме в базе данных MS SQL Server.



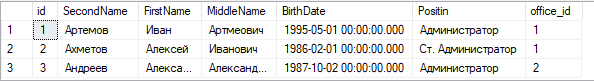
**Рис. 11.** Таблица «clients» (Клиенты) в MS SQL Server



**Рис. 12.** Таблица «cr\_type» (Типы кредитов) в MS SQL Server



**Рис. 13.** Таблица «credits» (Кредиты) в MS SQL Server

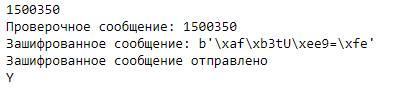


**Рис. 14.** Таблица «employers» (Сотрудники) в MS SQL Server

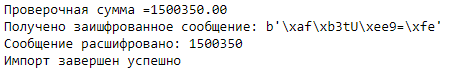


**Рис. 15.** Таблица « office» (Офисы) в MS SQL Server

После передачи сообщений с данными, клиент и сервер считают общую сумма кредитов плюс сумма срока кредитов, клиент отправляет эту сумму серверу, сервер на основании ее говорит и том корректно прошел импорт или нет.



**Рис. 16.** Отправка проверочного сообщения



**Рис. 17.** Получение проверочного сообщения

**Заключение**

В результате работы была достигнута цель и выполнены следующие требования:

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

**Перечень использованных источников**

1. SQLite Download Page. url: https://www.sqlite.org/download.html (дата обращения: 01.12.2020)
2. SQLite Is Public Domain. url: https://www.sqlite.org/copyright.html (дата обращения: 01.12.2020)
3. Python 3.7.0. url: <https://www.python.org/downloads/release/python-370/> (дата обращения: 01.12.2020)
4. History and License. url: <https://docs.python.org/3.7/license.html> (дата обращения: 01.12.2020)
5. Installation. url: <https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html> (дата обращения: 01.12.2020)
6. Documentation. url: <https://jupyter.org/documentation> (дата обращения: 01.12.2020)
7. Центр загрузки. url: <https://www.microsoft.com/ru-ru/Download> (дата обращения: 01.12.2020)
8. Project description. url: <https://pypi.org/project/pycrypto/> (дата обращения: 01.12.2020)

**Приложение**

Код программы для импорта данных (trrp2.ipynb):

|  |
| --- |
| #import  from Crypto.Cipher import PKCS1\_v1\_5  from socket import\*  from Crypto.PublicKey import RSA  from Crypto.Cipher import PKCS1\_OAEP  import datetime  from Crypto.Cipher import DES  import pickle  import pyodbc  from configparser import ConfigParser  import hashlib        def check\_data():  configur = ConfigParser()  configur.read('imp\_config.ini')  server\_name = configur.get('database', 'server\_name')  database\_name = configur.get('database', 'database\_name')  connection\_string = 'DRIVER={SQL Server};SERVER=' + server\_name + ';DATABASE=' + database\_name  connection = pyodbc.connect(connection\_string)  cursor = connection.cursor()  response = """  select sum(summ)+sum(life\_time\_m) from credits as a1  left join clients as a2 on a1.clients\_id=a2.id  left join cr\_type as a3 on a1.cr\_type\_id=a3.id  left join employers as a4 on a1.employers\_id=a4.id  left join office as a5 on a4.office\_id=a5.id  """  chek=0  cursor.execute(response)  for r in cursor:  chek=r[0]  print('Проверочная сумма =' + str(r[0]))  connection.commit()  connection.close()  return chek    def connect():    serv\_sock.bind((ip, int(port)))  print('Подключение произведено')  serv\_sock.listen(1)  client\_conn, client\_addr = serv\_sock.accept()  print('Соединение:', client\_addr)  serv\_rsa\_key = RSA.generate(1024)  exported\_key = serv\_rsa\_key.publickey()  exported\_key=exported\_key.export\_key()  print('Открытый ключ (md5 хеш): ' + str(hashlib.md5(exported\_key).hexdigest()))  data\_msg = pickle.dumps(exported\_key)  client\_conn.send(data\_msg)  print('Открытый ключ отправлен')  return client\_conn,serv\_rsa\_key  def recv\_key(client\_conn,serv\_rsa\_key):    data\_msg = client\_conn.recv(100000)  enc\_msg = pickle.loads(data\_msg)  print('Ключ симметричного шифрования получен:' + str(enc\_msg))  serv\_rsa\_cipher = PKCS1\_OAEP.new(serv\_rsa\_key)  des\_key = serv\_rsa\_cipher.decrypt(enc\_msg)  print('Ключ (md5 хеш):'+ str(hashlib.md5(des\_key).hexdigest()))  msg = 'Y'  data\_msg = pickle.dumps(msg.encode('utf-8'))  client\_conn.send(data\_msg)  serv\_des\_cipher = DES.new(des\_key, DES.MODE\_ECB)  return serv\_des\_cipher      def load\_data(serv\_des\_cipher,client\_conn):    server\_name = configur.get('database','server\_name')  database\_name = configur.get('database','database\_name')  connection\_string = 'DRIVER={SQL Server};SERVER=' + server\_name + ';DATABASE=' + database\_name  connection = pyodbc.connect(connection\_string)  cursor = connection.cursor()  load\_directory(cursor,client\_conn,serv\_des\_cipher)  load\_directory(cursor,client\_conn,serv\_des\_cipher)  load\_directory(cursor,client\_conn,serv\_des\_cipher)  load\_directory\_2(cursor,client\_conn,serv\_des\_cipher)  load\_directory\_3(cursor,client\_conn,serv\_des\_cipher)    connection.commit()  connection.close()    chek\_v=check\_data()  data\_msg = client\_conn.recv(100000)  enc\_msg = pickle.loads(data\_msg)  print('Получено заишфрованное сообщение: ' + str(enc\_msg))  dec\_msg = serv\_des\_cipher.decrypt(enc\_msg)  data\_ = str(dec\_msg.decode('utf-8')).strip()  print('Сообщение расшифровано: ' + str(dec\_msg.decode('utf-8')))  if (int(data\_)==int(chek\_v)):  print('Импорт завершен успешно')  else:  print('Возникла ошибка')  msg = 'Y'  data\_msg = pickle.dumps(msg.encode('utf-8'))  client\_conn.send(data\_msg)    def load\_directory(cursor,client\_conn,serv\_des\_cipher):  data\_msg = client\_conn.recv(100000)  enc\_msg = pickle.loads(data\_msg)  print('Получено заишфрованное сообщение: ' + str(enc\_msg))  dec\_msg = serv\_des\_cipher.decrypt(enc\_msg)  data\_ = str(dec\_msg.decode('utf-8')).strip()  print('Сообщение расшифровано: ' + str(dec\_msg.decode('utf-8')))  cursor.execute(data\_)  msg = 'Y'  data\_msg = pickle.dumps(msg.encode('utf-8'))  client\_conn.send(data\_msg)    def load\_directory\_2(cursor,client\_conn,serv\_des\_cipher):  data\_msg = client\_conn.recv(100000)  enc\_msg = pickle.loads(data\_msg)  print('Получено заишфрованное сообщение: ' + str(enc\_msg))  dec\_msg = serv\_des\_cipher.decrypt(enc\_msg)  data\_ = str(dec\_msg.decode('utf-8')).strip().split("|")  print('Сообщение расшифровано: ' + str(dec\_msg.decode('utf-8')))  index=""  for i in range(len(data\_)-1):  if (i%2==0):  print(data\_[i])  cursor.execute(str(data\_[i]))  index=cursor.fetchone()  if (i%2==1):  cursor.execute(data\_[i]+" , "+str(index[0])+")")  msg = 'Countries has been caugth'  data\_msg = pickle.dumps(msg.encode('utf-8'))  client\_conn.send(data\_msg)      def load\_directory\_3(cursor,client\_conn,serv\_des\_cipher):  data\_msg = client\_conn.recv(100000)  enc\_msg = pickle.loads(data\_msg)  print('Получено заишфрованное сообщение: ' + str(enc\_msg))  dec\_msg = serv\_des\_cipher.decrypt(enc\_msg)  data\_ = str(dec\_msg.decode('utf-8')).strip().split("|")  print('Сообщение расшифровано: ' + str(dec\_msg.decode('utf-8')))  index=""  index1=""  index2=""  for i in range(len(data\_)-1):  if (i%4==0):  cursor.execute(str(data\_[i]))  index=cursor.fetchone()  if (i%4==1):  cursor.execute(str(data\_[i]))  index1=cursor.fetchone()  if (i%4==2):  cursor.execute(str(data\_[i]))  index2=cursor.fetchone()  if (i%4==3):  cursor.execute(data\_[i]+" , "+str(index[0])+" , "+str(index1[0])+" , "+str(index2[0])+")")  msg = 'Y'  data\_msg = pickle.dumps(msg.encode('utf-8'))  client\_conn.send(data\_msg)    configur = ConfigParser()  configur.read('imp\_config.ini')  ip = configur.get('network','ip')  port = configur.get('network','port')  serv\_sock = socket(AF\_INET, SOCK\_STREAM)  client\_conn,serv\_rsa\_key=connect()  serv\_des\_cipher=recv\_key(client\_conn,serv\_rsa\_key)  load\_data(serv\_des\_cipher,client\_conn)  print('End') |

Код программы для экспорта (trrp2\_1.ipynb):

|  |
| --- |
| #export  from socket import\*  import time  import datetime  from Crypto.PublicKey import RSA  from Crypto.Cipher import PKCS1\_OAEP  import random  from Crypto.Cipher import DES  import pickle  import sqlite3  from configparser import ConfigParser  import hashlib    def pad(text):  while len(text) % 8 != 0:  text += b' '  return text  def check\_data():  configur = ConfigParser()  configur.read('exp\_config.ini')  path = configur.get('database', 'path')  connection = sqlite3.connect(path)  cursor = connection.cursor()  response = 'SELECT sum(Summ)+sum(life\_time) FROM cred\_inf;'  rows = cursor.execute(response)  chek=0  for r in rows:  print(r[0])  chek=r[0]  connection.close()  return chek    def connect():  client\_sock.connect((ip, int(port)))  print('Подключение произведено')  data\_msg = client\_sock.recv(100000)  exported\_key = pickle.loads(data\_msg)  rsa\_public\_key = RSA.import\_key(exported\_key)  client\_rsa\_cipher = PKCS1\_OAEP.new(rsa\_public\_key)  print('Получен открытый ключ (md5 хеш): ' + str(hashlib.md5(rsa\_public\_key.export\_key()).hexdigest()))  return client\_rsa\_cipher      def send\_key(client\_rsa\_cipher):    des\_key = ''  for i in range(8):  letter\_num = random.randint(0,25) + ord('a')  des\_key += chr(letter\_num)  print('Ключ симметричного шифрования (md5 хеш): '+ hashlib.md5(des\_key.encode('utf-8')).hexdigest())  enc\_msg = client\_rsa\_cipher.encrypt(des\_key.encode('utf-8'))  print('Зашифрованный ключ симметричного шифрования: ' + str(enc\_msg))  data\_msg = pickle.dumps(enc\_msg)  client\_sock.send(data\_msg)  print('Ключ симметричного шифрования отправлен')  data\_msg = client\_sock.recv(100000)  print('Y')  client\_des\_cipher = DES.new(des\_key.encode('utf-8'), DES.MODE\_ECB)  return client\_des\_cipher        def send\_data(client\_des\_cipher):    path = configur.get('database', 'path')  connection = sqlite3.connect(path)  cursor = connection.cursor()  cursor.execute("select distinct cr\_type\_Name , [limit] , Count\_m from cred\_inf")  data=cursor.fetchall()  msg = ''  for row in data:  msg += ("insert into cr\_type values('"+str(row[0])+"',"+str(row[1])+","+str(row[2])+")" + " ")  print('Сообщение: ' + msg)  enc\_msg = client\_des\_cipher.encrypt(pad(msg.encode('utf-8')))  print('Зашифрованное сообщение: ' + str(enc\_msg))  data\_msg = pickle.dumps(enc\_msg)  client\_sock.send(data\_msg)  print('Зашифрованное сообщение отправлено')  data\_msg = client\_sock.recv(100000)  print('Y')  #####  cursor.execute("select distinct office\_Name from cred\_inf")  data=cursor.fetchall()  msg = ''  for row in data:  msg += ("insert into office values('"+str(row[0])+"')" + " ")  print('Сообщение: ' + msg)  enc\_msg = client\_des\_cipher.encrypt(pad(msg.encode('utf-8')))  print('Зашифрованное сообщение: ' + str(enc\_msg))  data\_msg = pickle.dumps(enc\_msg)  client\_sock.send(data\_msg)  print('Зашифрованное сообщение отправлено')  data\_msg = client\_sock.recv(100000)  print('Y')  #####  cursor.execute("select distinct client\_SecondName,client\_FirstName,client\_MiddleName, client\_BDate, EMP , client\_Phone from cred\_inf")  data=cursor.fetchall()  msg = ''  for row in data:  bdate=str(row[3]).split(".")[2]+str(row[3]).split(".")[1]+str(row[3]).split(".")[0]  msg += ("insert into clients values('"+str(row[0])+"','"+str(row[1])+"','"+str(row[2])+"','"+str(bdate)+"','"+str(row[4])+"','"+str(row[5])+"')" + " ")  print('Сообщение: ' + msg)  enc\_msg = client\_des\_cipher.encrypt(pad(msg.encode('utf-8')))  print('Зашифрованное сообщение: ' + str(enc\_msg))  data\_msg = pickle.dumps(enc\_msg)  client\_sock.send(data\_msg)  print('Зашифрованное сообщение отправлено')  data\_msg = client\_sock.recv(100000)  print('Y')  ####  cursor.execute("select distinct employ\_SecondName,employ\_FirstName,employ\_MiddleName,employ\_BDate,Positin,office\_Name from cred\_inf")  data=cursor.fetchall()  msg = ''  for row in data:  bdate=str(row[3]).split(".")[2]+str(row[3]).split(".")[1]+str(row[3]).split(".")[0]  msg += "select top 1 id from office where Name='"+str(row[5])+"'| "  msg += "insert into employers values('"+str(row[0])+"','"+str(row[1])+"','"+str(row[2])+"','"+str(bdate)+"','"+str(row[4])+"' |"    print('Сообщение: ' + msg)  enc\_msg = client\_des\_cipher.encrypt(pad(msg.encode('utf-8')))  print('Зашифрованное сообщение: ' + str(enc\_msg))  data\_msg = pickle.dumps(enc\_msg)  client\_sock.send(data\_msg)  print('Зашифрованное сообщение отправлено')  data\_msg = client\_sock.recv(100000)  print('Y')  ####  cursor.execute("select Summ , life\_time, percent,datest,cr\_type\_Name , [limit] , Count\_m , client\_SecondName,client\_FirstName,client\_MiddleName, client\_BDate, employ\_SecondName,employ\_FirstName,employ\_MiddleName,employ\_BDate from cred\_inf")  data=cursor.fetchall()  msg = ''  for row in data:  bdate=str(row[3]).split(".")[2]+str(row[3]).split(".")[1]+str(row[3]).split(".")[0]  bdate2=str(row[10]).split(".")[2]+str(row[10]).split(".")[1]+str(row[10]).split(".")[0]  bdate3=str(row[14]).split(".")[2]+str(row[14]).split(".")[1]+str(row[14]).split(".")[0]  datest=str(row[3]).split(".")[2]+str(row[3]).split(".")[1]+str(row[3]).split(".")[0]  msg += "select top 1 id from cr\_type where Name='"+str(row[4])+"' and limit="+str(row[5])+" and Count\_m="+str(row[6])+"|"  msg += "select top 1 id from clients where SecondName='"+str(row[7])+"' and FirstName='"+str(row[8])+"' and MiddleName='"+str(row[9])+"' and BirthDate='"+bdate2+"'|"  msg += "select top 1 id from employers where SecondName='"+str(row[11])+"' and FirstName='"+str(row[12])+"' and MiddleName='"+str(row[13])+"' and BirthDate='"+bdate3+"'|"  msg += "insert into credits values("+str(row[0])+","+str(row[1])+",'"+datest+"',"+str(row[2])+"|"    print('Сообщение: ' + msg)  enc\_msg = client\_des\_cipher.encrypt(pad(msg.encode('utf-8')))  print('Зашифрованное сообщение: ' + str(enc\_msg))  data\_msg = pickle.dumps(enc\_msg)  client\_sock.send(data\_msg)  print('Зашифрованное сообщение отправлено')  data\_msg = client\_sock.recv(100000)  print('Y')  connection.close()  ####  chek\_v=check\_data()  msg=str(chek\_v)  print('Проверочное сообщение: ' + msg)  enc\_msg = client\_des\_cipher.encrypt(pad(msg.encode('utf-8')))  print('Зашифрованное сообщение: ' + str(enc\_msg))  data\_msg = pickle.dumps(enc\_msg)  client\_sock.send(data\_msg)  print('Зашифрованное сообщение отправлено')  data\_msg = client\_sock.recv(100000)  print('Y')    configur = ConfigParser()  configur.read('exp\_config.ini')  ip = configur.get('network', 'ip')  port = configur.get('network', 'port')  client\_sock = socket(AF\_INET, SOCK\_STREAM)  client\_rsa\_cipher=connect()  client\_des\_cipher=send\_key(client\_rsa\_cipher)  send\_data(client\_des\_cipher)  time.sleep(3)  print('End') |