| create table arp\_diagnostics (  id serial primary key,  abonent\_id serial,  has\_arp boolean not null,  arp\_id serial,  time timestamp );  create table district (  id serial primary key,  town text,  district text );  create table employee (  id serial primary key,  name text not null,  surname text not null,  role text not null,  employment\_date date );  create table cable\_diagnostics (  id serial primary key,  employee\_id serial not null,  port\_id serial not null,  link\_state text,  diagnostics text,  constraint fk\_cable\_diagnostics\_employee\_id  foreign key (employee\_id)  references employee(id) );  create table executor (  request\_id serial primary key,  department text );  create table provider (  id serial primary key,  name text not null,  type\_of\_connection text );  create table abonent (  id serial primary key,  login text not null unique,  account\_number integer not null unique,  name text,  surname text,  phone\_number text unique,  email text unique,  status text,  address\_id serial not null,  note text,  provider\_id serial not null,  constraint fk\_abonent\_provider\_id  foreign key (provider\_id)  references provider(id) );  create table vlan (  id serial primary key,  name text not null );  create table switch (  id serial primary key,  ip varchar(16) not null unique,  model text,  port\_count integer,  house\_id serial,  provider\_id serial,  uplink\_port integer );  create table commutation (  abonent\_id serial,  switch\_id serial,  port\_id serial,  cable\_type text,  primary key (abonent\_id, switch\_id, port\_id) );  create table equipment\_connections (  equipment1\_id serial,  equipment2\_id serial,  speed integer not null check (speed = 100 or speed = 1000 or speed = 10000),  primary key (equipment1\_id, equipment2\_id) );  create table payment\_history (  abonent\_id serial primary key,  date timestamp not null,  balance\_change integer not null );  create table address (  id serial primary key,  house\_id serial not null,  porch integer,  floor integer,  flat integer not null );  create table request (  id serial primary key,  abonent\_id serial not null,  author\_id serial not null,  create\_date timestamp not null,  status text not null,  status\_change\_date timestamp,  close\_date timestamp,  type text not null,  text text,  constraint fk\_request\_author\_id  foreign key (author\_id)  references employee(id) );  create table node (  id serial primary key,  model text,  district\_id serial,  provider\_id serial,  constraint fk\_node\_district\_id  foreign key (district\_id)  references district(id)  constraint fk\_node\_provider\_id  foreign key (provider\_id)  references provider(id) );  create table mac\_address (  port\_id serial,  mac\_address text,  primary key (port\_id, mac\_address) );  create table connection\_diagnostics (  id serial primary key,  employee\_id serial not null,  switch text not null,  port text not null,  has\_link boolean,  link\_type text,  has\_mac boolean,  errors integer,  errors\_rise boolean,  correct\_vlan boolean,  switch\_loss integer );  create table billing (  abonent\_id serial primary key,  balance integer not null,  status text );  create table house (  id serial primary key,  district\_id serial not null,  street text not null,  number integer not null );  create table router\_settings (  abonent\_id serial primary key,  ip varchar(16) not null,  mask varchar(16) not null,  gateway varchar(16) not null );  create table arp (  id serial primary key,  create\_date timestamp not null,  mac\_address\_id serial not null,  node\_id serial not null,  vlan\_id serial not null );  create table comment (  id serial primary key,  request\_id serial not null,  date timestamp not null,  text text not null,  employee\_id serial not null,  constraint fk\_comment\_request\_id  foreign key (request\_id)  references request(id),  constraint fk\_comment\_employee\_id  foreign key (employee\_id)  references employee(id) );  create table port (  id serial primary key,  switch\_id serial not null,  port\_number integer not null,  address\_id serial,  vlan\_id serial,  port\_status text not null,  connection\_speed integer check (connection\_speed = 0 or connection\_speed = 10 or connection\_speed = 100 or connection\_speed = 1000),  description text,  connection\_status text,  constraint fk\_port\_switch\_id  foreign key (switch\_id)  references switch(id),  constraint fk\_port\_vlan\_id  foreign key (vlan\_id)  references vlan(id) ); |
| --- |

Заполнение

import psycopg2

from sshtunnel import SSHTunnelForwarder

import io

import hashlib

import string

import random

from datetime import datetime

PORT = 5432

REMOTE\_USERNAME = 's312637'

REMOTE\_HOST = 'helios.cs.ifmo.ru'

REMOTE\_SSH\_PORT = 2222

REMOTE\_PASSWORD = 'XVQI.2552'

server = SSHTunnelForwarder((REMOTE\_HOST, REMOTE\_SSH\_PORT),

ssh\_username=REMOTE\_USERNAME,

ssh\_password=REMOTE\_PASSWORD,

remote\_bind\_address=('192.168.10.80', PORT),

local\_bind\_address=('localhost', 10022))

def add\_address(cursor, town, district, street, house\_number, porch, floor, flat):

query = """SELECT id FROM district WHERE town=%s AND district=%s;"""

cursor.execute(query, (town, district))

id = cursor.fetchone()

if not(id):

query = """INSERT INTO district (town, district) VALUES (%s, %s) RETURNING id;"""

cursor.execute(query, (town, district))

district\_id = cursor.fetchone()[0]

else:

district\_id = id[0]

query = """SELECT id FROM house WHERE district\_id=%s AND street=%s AND number=%s;"""

cursor.execute(query, (district\_id, street, house\_number))

id = cursor.fetchone()

if not(id):

query = """INSERT INTO house (district\_id, street, number) VALUES (%s, %s, %s) RETURNING id;"""

cursor.execute(query, (district\_id, street, house\_number))

house\_id = cursor.fetchone()[0]

else:

house\_id = id[0]

query = """SELECT id FROM address WHERE house\_id=%s AND flat=%s;"""

cursor.execute(query, (house\_id, flat))

id = cursor.fetchone()

if not(id):

query = """INSERT INTO address (house\_id, porch, floor, flat) VALUES (%s, %s, %s, %s) RETURNING id;"""

cursor.execute(query, (house\_id, porch, floor, flat))

return cursor.fetchone()[0]

else:

return id[0]

def get\_address(cursor, town, district, street, house\_number, porch, floor, flat):

query = """SELECT id FROM district WHERE town=%s AND district=%s;"""

cursor.execute(query, (town, district))

id = cursor.fetchone()

if not(id):

return None

else:

district\_id = id[0]

query = """SELECT id FROM house WHERE district\_id=%s AND street=%s AND number=%s;"""

cursor.execute(query, (district\_id, street, house\_number))

id = cursor.fetchone()

if not(id):

return None

else:

house\_id = id[0]

query = """SELECT id FROM address WHERE house\_id=%s AND flat=%s;"""

cursor.execute(query, (house\_id, flat))

id = cursor.fetchone()

if not(id):

return None

else:

return int(id[0])

def add\_abonent(cursor, login, account\_number, name, surname, phone\_number, email, status, address\_id, note):

print("Choose provider id (1 - Ростелеком, 2 - Етелеком, 3 - Тиера, 4 - Скайнет):")

provider\_id = int(input())

query = """INSERT INTO abonent (login, account\_number, name, surname, phone\_number, email, status, address\_id, note, provider\_id)

VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s) RETURNING id;"""

cursor.execute(query, (login, account\_number, name, surname, phone\_number, email, status, address\_id, note, provider\_id))

if \_\_name\_\_ == "\_\_main\_\_":

server.start()

conn = psycopg2.connect(user=REMOTE\_USERNAME, database='studs', password='IUkanShc6DhfHo7x', host='localhost', port=10022)

cursor = conn.cursor()

add\_abonent(cursor, "fr415", "55566", "Игорь", "Вихорьков", "89123339512", "igor@gmail.com", "действующий",

add\_address(cursor, "Санкт-Петербург", "Невский", "Дыбенко", 26, 3, 1, 5),

None)

add\_abonent(cursor, "ps466", "11111", "Анна", "Павловна", "89123316323", "anna@gmail.com", "действующий",

add\_address(cursor, "Санкт-Петербург", "Шушары", "Школьный", 67, 2, 5, 77),

None)

add\_abonent(cursor, "fr108", "374748", "Дмитрий", "Борисов", "89124454635", "dmitry@gmail.com", "действующий",

add\_address(cursor, "Санкт-Петербург", "Невский", "Большевиков", 35, 2, 6, 90),

None)

add\_abonent(cursor, "pk610", "939503", "Валентина", "Смирнов", "89126670956", "valentina@gmail.com", "действующий",

add\_address(cursor, "Санкт-Петербург", "Невский", "Обуховской обороны", 220, 12, 15, 500),

None)

add\_abonent(cursor, "kl407", "5830303", "Захар", "Алексеев", "89123774478", "zahar@gmail.com", "действующий",

add\_address(cursor, "Санкт-Петербург", "Петроградский", "Кронверкский", 30, 3, 1, 6),

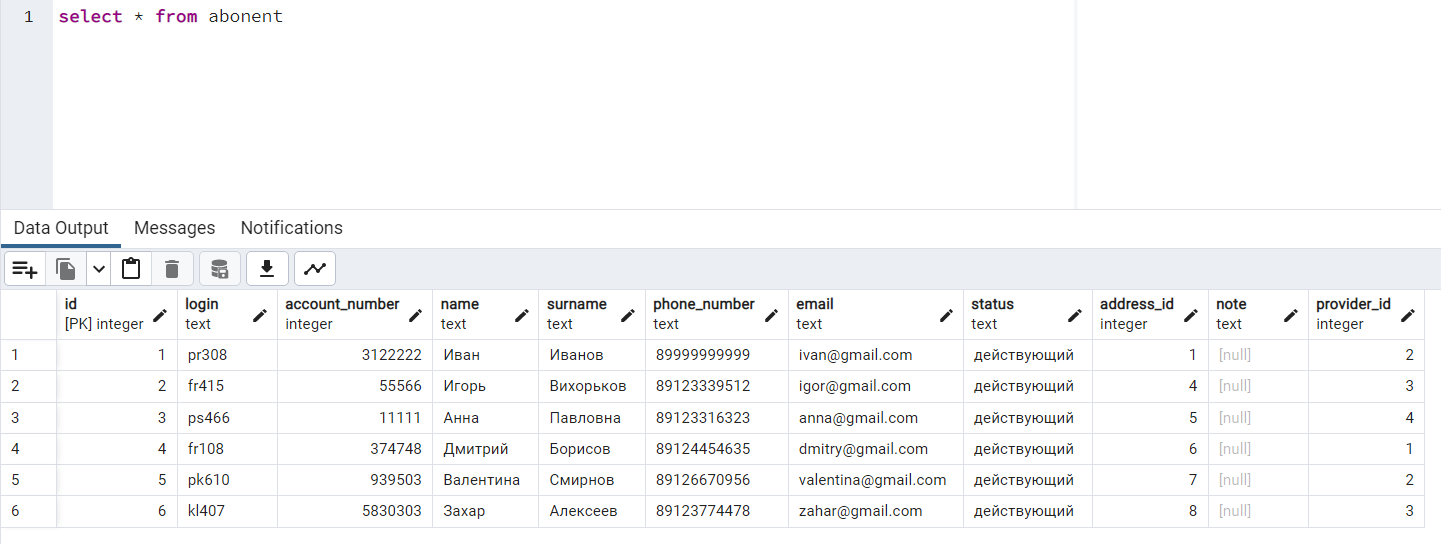
None)

conn.commit()

cursor.close()

conn.close()

| insert into switch(ip, model, port\_count, house\_id, provider\_id, uplink\_port) values ('172.22.33.55', 'DGS-3000', 5, 2, 2, 1); insert into switch(ip, model, port\_count, house\_id, provider\_id, uplink\_port) values ('172.22.34.55', 'DGS-3000', 5, 5, 3, 1); insert into switch(ip, model, port\_count, house\_id, provider\_id, uplink\_port) values ('172.22.35.55', 'DGS-3000', 5, 6, 4, 1); insert into switch(ip, model, port\_count, house\_id, provider\_id, uplink\_port) values ('172.22.36.55', 'DGS-3000', 5, 7, 1, 1); insert into switch(ip, model, port\_count, house\_id, provider\_id, uplink\_port) values ('172.22.37.55', 'DGS-3000', 5, 8, 2, 1); insert into switch(ip, model, port\_count, house\_id, provider\_id, uplink\_port) values ('172.22.38.55', 'DGS-3000', 5, 9, 3, 1); |
| --- |



| INSERT INTO equipment\_connections(equipment1\_id, equipment2\_id, speed) VALUES (38, 39, 1000); |
| --- |

def show\_abonents(cursor):

cursor.execute(

"""

SELECT

abonent.id,

abonent.login,

abonent.name,

abonent.surname,

abonent.address\_id,

abonent.provider\_id,

house.id,

district.id

FROM abonent

INNER JOIN address ON abonent.address\_id = address.id

INNER JOIN house ON address.house\_id = house.id

INNER JOIN district ON house.district\_id = district.id

"""

)

results = cursor.fetchall()

cursor.close()

connection.close()

header = ["abonent\_id", "login", "name", "surname", "address\_id", "provider\_id", "house\_id", "district\_id"]

data = pd.DataFrame(results, columns=header)

print(data)