МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики  
Кафедра суперкомпьютеров и общей информатики

**Отчет по лабораторной работе № 1**

Дисциплина: «Корпоративные базы данных»

Выполнил: Неженский М.С.

Группа: 6133-010402D

Самара 2023

# **Задание на лабораторную работу №1**

1. Выбрать предметную область

2. Разработать ER модель, включающую минимум 5-6 сущностей и типы связей: 1-N, N:M, 1-1.

3. Создать базу данных по модели в СУБД PostgreSQL.

4. Определить индексы, уникальные индексы.

5. Разработать типовые запросы к СУБД на языке SQL. Получение списков данных. Агрегация. Поиск.

6. Разработайте хранимые процедуры на языке PL/pgSQL для генерации случайных данных для базы данных.

7. Сгенерируйте тестовые данные при помощи разработанных процедур.

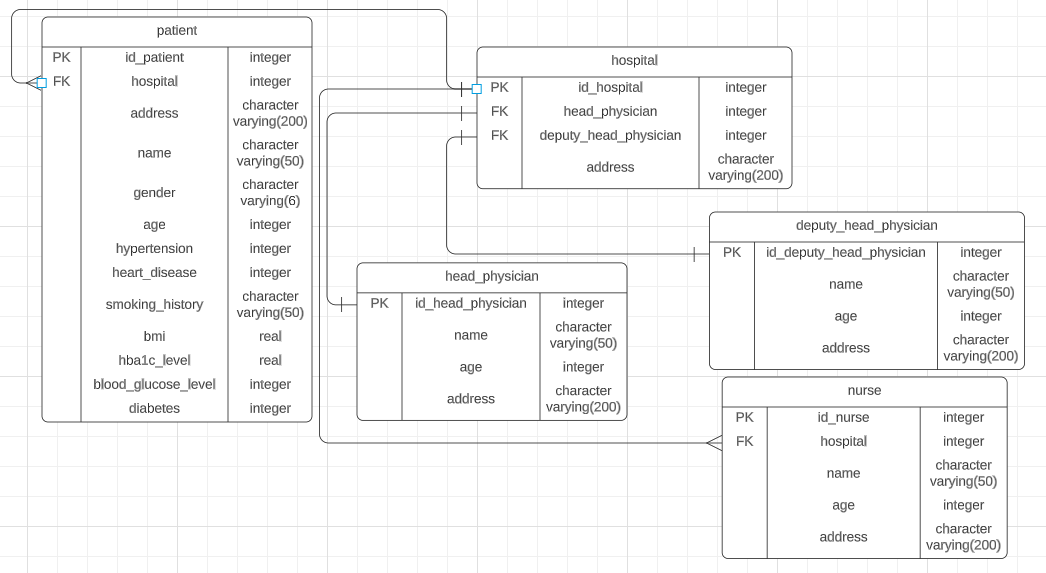
8. Протестируйте работу запросов на больших объёмах данных (Порядка 1 миллиона записей в основных таблицах).

9. Измените конфигурацию сервера PostgreSQL для достижения лучшей производительности на самых медленных запросах. Оптимизируйте схему БД и запросы для достижения лучшей производительности.

Пункты 6-7 допустимо реализовывать другими способами без PL/pgSQL

# **Ход работы**

**ER-диаграмма базы данных:**

  
Рисунок 1 – ER-диаграмма базы данных

В данной работе представлены связи 1-N, N:M, 1-1, описана модель больницы. Поля и названия классов пытался брать «говорящие».

1) patient – пациент и в данном классе описываются все анализы, конкретного человека и его данные.

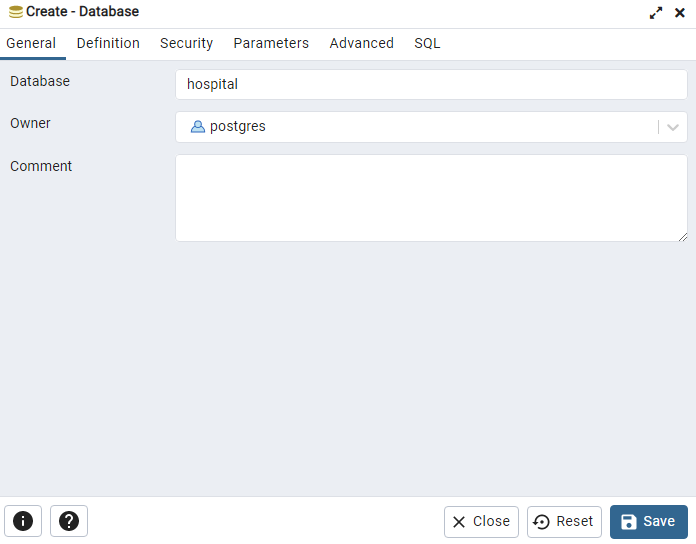
2) head\_physicial – главный врач больницы.

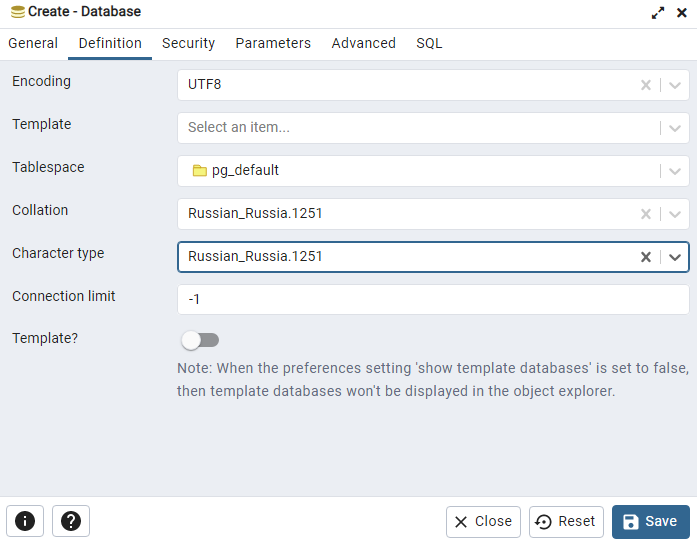
3) deputy\_head\_physicial – заместитель главного врача.

4) nurse – медперсонал.

5) hospital – больница.

**Создание базы данных:**





**Код создания таблиц в базе данных(Замечание: если вставлять код скорее всего выйдет ошибка из-за кавычек, поэтому двойные кавычки надо проставлять самостоятельно):**

CREATE TABLE IF NOT EXISTS hospital (

id\_hospital integer NOT NULL GENERATED ALWAYS AS IDENTITY (INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 1000000000 CACHE 1),

head\_physician integer,

deputy\_head\_physician integer,

address character varying(200) COLLATE pg\_catalog.”default” NOT NULL,

CONSTRAINT pkey\_hospital PRIMARY KEY (id\_hospital)

);

CREATE TABLE IF NOT EXISTS patient (

id\_patient integer NOT NULL GENERATED ALWAYS AS IDENTITY (INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 1000000000 CACHE 1),

hospital integer,

address character varying(200) COLLATE pg\_catalog.”default” NOT NULL,

name character varying(50) COLLATE pg\_catalog.”default” NOT NULL,

gender character varying(6) COLLATE pg\_catalog.”default” NOT NULL,

age integer,

hypertension integer,

heart\_disease integer,

smoking\_history character varying(50) COLLATE pg\_catalog.”default” NOT NULL,

bmi real,

HbA1c\_level real,

blood\_glucose\_level integer,

diabetes integer,

CONSTRAINT pkey\_patient PRIMARY KEY (id\_patient)

);

CREATE TABLE IF NOT EXISTS head\_physician (

id\_head\_physician integer NOT NULL GENERATED ALWAYS AS IDENTITY (INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 1000000000 CACHE 1),

name character varying(50) COLLATE pg\_catalog.”default” NOT NULL,

age integer,

address character varying(200) COLLATE pg\_catalog.”default” NOT NULL,

CONSTRAINT pkey\_head\_physician PRIMARY KEY (id\_head\_physician)

);

CREATE TABLE IF NOT EXISTS deputy\_head\_physician (

id\_deputy\_head\_physician integer NOT NULL GENERATED ALWAYS AS IDENTITY (INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 1000000000 CACHE 1),

name character varying(50) COLLATE pg\_catalog.”default” NOT NULL,

age integer,

address character varying(200) COLLATE pg\_catalog.”default” NOT NULL,

CONSTRAINT pkey\_deputy\_head\_physician PRIMARY KEY (id\_deputy\_head\_physician)

);

CREATE TABLE IF NOT EXISTS nurse (

id\_nurse integer NOT NULL GENERATED ALWAYS AS IDENTITY (INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 1000000000 CACHE 1),

hospital integer,

name character varying(50) COLLATE pg\_catalog.”default” NOT NULL,

age integer,

address character varying(200) COLLATE pg\_catalog.”default” NOT NULL,

CONSTRAINT pkey\_nurse PRIMARY KEY (id\_nurse)

)

**Код для создания FK-ключей для создания связей между таблицами:**

ALTER TABLE public.hospital ADD CONSTRAINT FK\_hospital\_head\_physician FOREIGN KEY (head\_physician) REFERENCES public.head\_physician(id\_head\_physician)

ON DELETE CASCADE

ON UPDATE CASCADE

ALTER TABLE public.hospital ADD CONSTRAINT FK\_hospital\_deputy\_head\_physician FOREIGN KEY (deputy\_head\_physician) REFERENCES public.deputy\_head\_physician (id\_deputy\_head\_physician)

ON DELETE CASCADE

ON UPDATE CASCADE

ALTER TABLE public.nurse ADD CONSTRAINT FK\_nurse\_hospital FOREIGN KEY (hospital) REFERENCES public.hospital(id\_hospital)

ON DELETE CASCADE

ON UPDATE CASCADE

ALTER TABLE public.patient ADD CONSTRAINT FK\_patient\_hospital FOREIGN KEY (hospital) REFERENCES public.hospital(id\_hospital)

ON DELETE CASCADE

ON UPDATE CASCADE

**Код для заполнения данными таблиц(данный код написан на языке python в среде разработки jupyter lab):**

import csv

import random

from mimesis import Person

from mimesis import Address

from mimesis.locales import Locale

person = Person(Locale.RU)

address = Address(Locale.RU)

my\_file = open("C:\head\_physician.csv", "w+", newline='')

with my\_file:

writer = csv.writer(my\_file)

for i in range(0, 26):

newRow = []

newRow.append(i)

newRow.append(person.full\_name())

newRow.append(random.randint(30, 65))

newRow.append(address.address())

writer.writerow(newRow)

my\_file.close()

deputy\_head\_physician\_file = open("C:\deputy\_head\_physician.csv", "w+", newline='')

with deputy\_head\_physician\_file:

writer = csv.writer(deputy\_head\_physician\_file)

for i in range(0, 26):

newRow = []

newRow.append(i)

newRow.append(person.full\_name())

newRow.append(random.randint(26, 65))

newRow.append(address.address())

writer.writerow(newRow)

deputy\_head\_physician\_file.close()

hospital\_file = open("C:\hospital.csv", "w+", newline='')

with hospital\_file:

writer = csv.writer(hospital\_file)

import numpy as np

spam1 = np.random.permutation(range(1, 26))

spam2 = np.random.permutation(range(1, 26))

for i in range(0, 26):

newRow = []

newRow.append(i)

newRow.append(spam1[i-1])

newRow.append(spam2[i-1])

newRow.append(address.address())

writer.writerow(newRow)

hospital\_file.close()

nurse = open("C:\ nurse.csv", "w+", newline='')

with nurse:

writer = csv.writer(nurse)

import numpy as np

for i in range(0, 101):

newRow = []

newRow.append(i)

newRow.append(random.randint(1, 25))

newRow.append(person.full\_name())

newRow.append(random.randint(20, 65))

newRow.append(address.address())

writer.writerow(newRow)

nurse.close()

patient = open("C:\patient.csv", "w+", newline='')

with patient:

writer = csv.writer(patient)

import numpy as np

for i in range(0, 1000001):

newRow = []

newRow.append(i)

newRow.append(random.randint(1, 25))

newRow.append(address.address())

newRow.append(person.full\_name())

newRow.append(random.choice(['М', 'Ж']))

newRow.append(random.randint(18, 85))

newRow.append(random.randint(0, 1))

newRow.append(random.randint(0, 1))

newRow.append(random.choice(['Никогда не курил(a)', 'Нет информации', 'Курил ранее', 'Курит сейчас']))

newRow.append(round(random.uniform(10, 96),2))

newRow.append(round(random.uniform(3, 10),2))

newRow.append(random.randint(80, 301))

newRow.append(random.randint(0, 1))

writer.writerow(newRow)

patient.close()

**Код для записи данных из сгенерированных файлов в базу данных:**

COPY head\_physician

FROM 'C:\head\_physician.csv' with delimiter ',' csv header encoding 'windows-1251';

COPY deputy\_head\_physician

FROM 'C:\deputy\_head\_physician.csv' with delimiter ',' csv header encoding 'windows-1251';

COPY hospital

FROM 'C:\hospital.csv' with delimiter ',' csv header encoding 'windows-1251';

SELECT \*

FROM hospital

COPY nurse

FROM 'C:\nurse.csv' with delimiter ',' csv header encoding 'windows-1251';

SELECT \*

FROM nurse

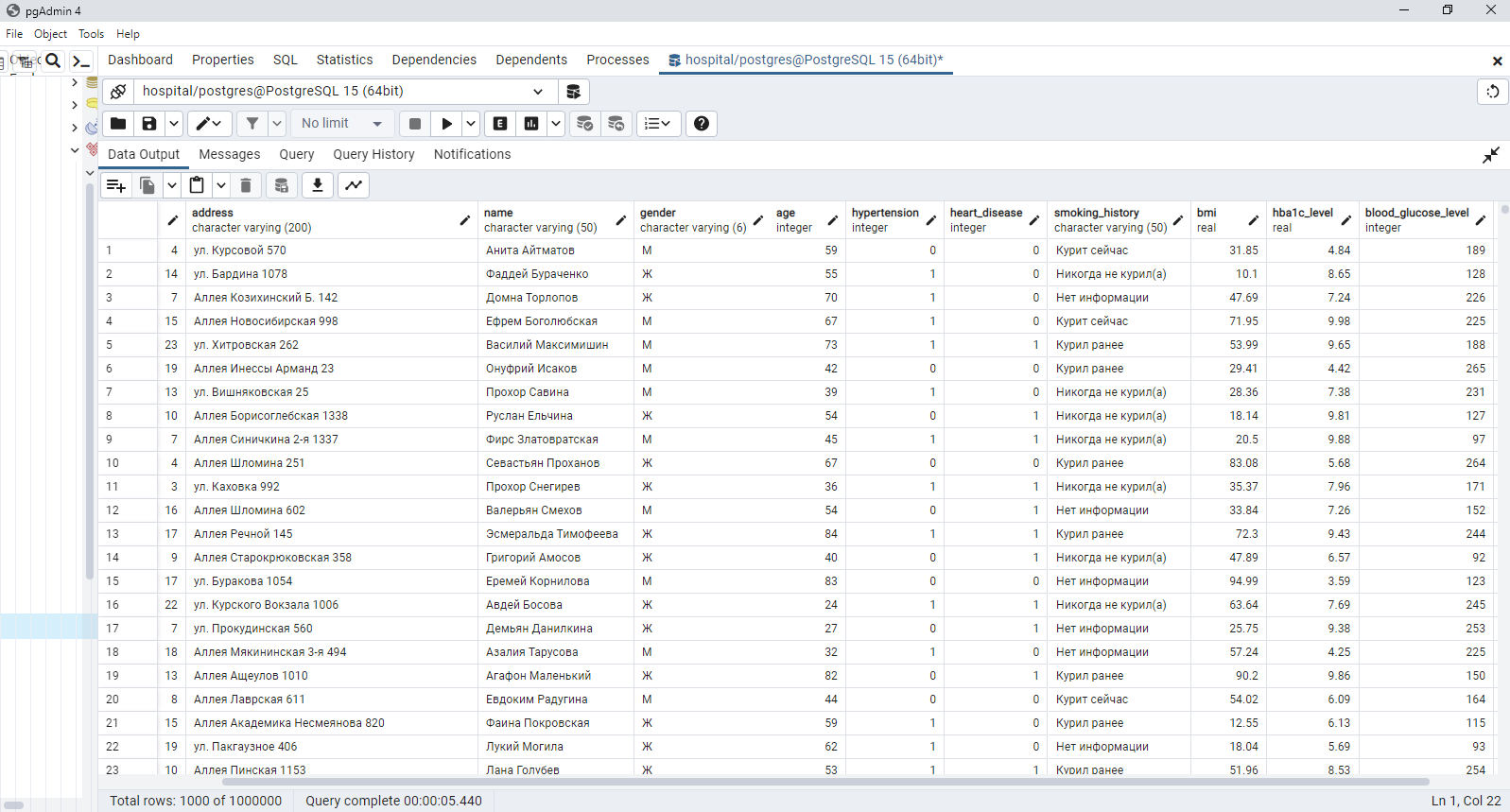
**Не с первого раза все идеально получилось поэтому использовал еще эти команды:**

TRUNCATE TABLE hospital – **очистка данных таблицы hospital**

DROP TABLE hospital – **удаление таблицы hospital**

**Основные запросы (тут уже миллион записей в классе patient):**

1. SELECT \* FROM patient



2) SELECT \* FROM nurse

WHERE hospital = '5'

3)INSERT INTO public.head\_physician values(DEFAULT, 'Валентин Дудкин', '48', 'ул.Компьютерная, 7')

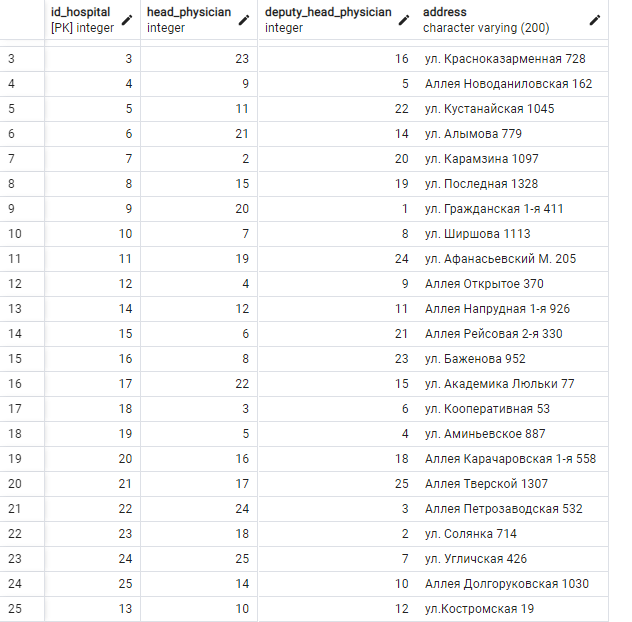


4) UPDATE hospital

SET head\_physician = '10'

WHERE address = 'ул.Костромская 19'

SELECT \* FROM hospital

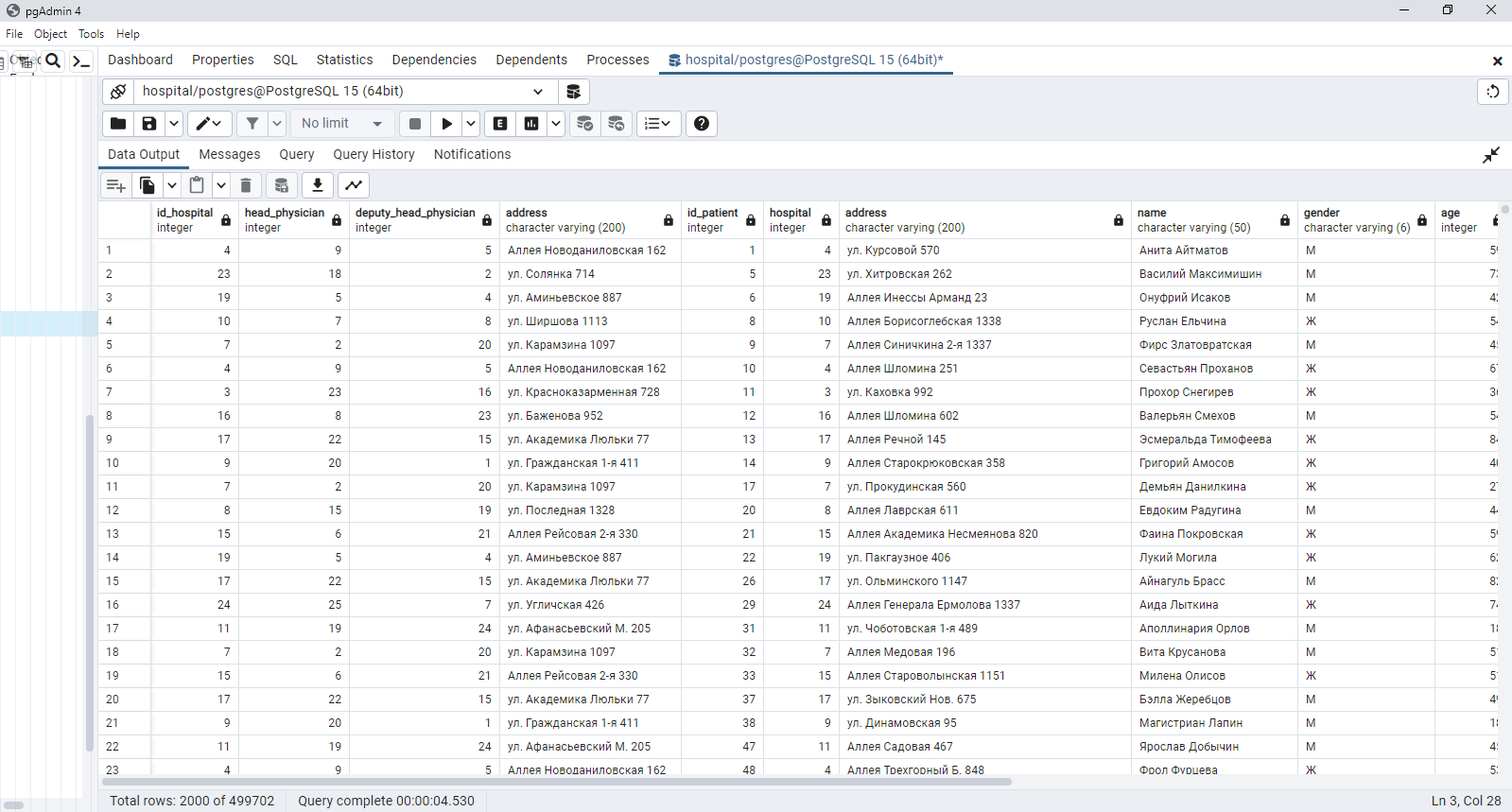


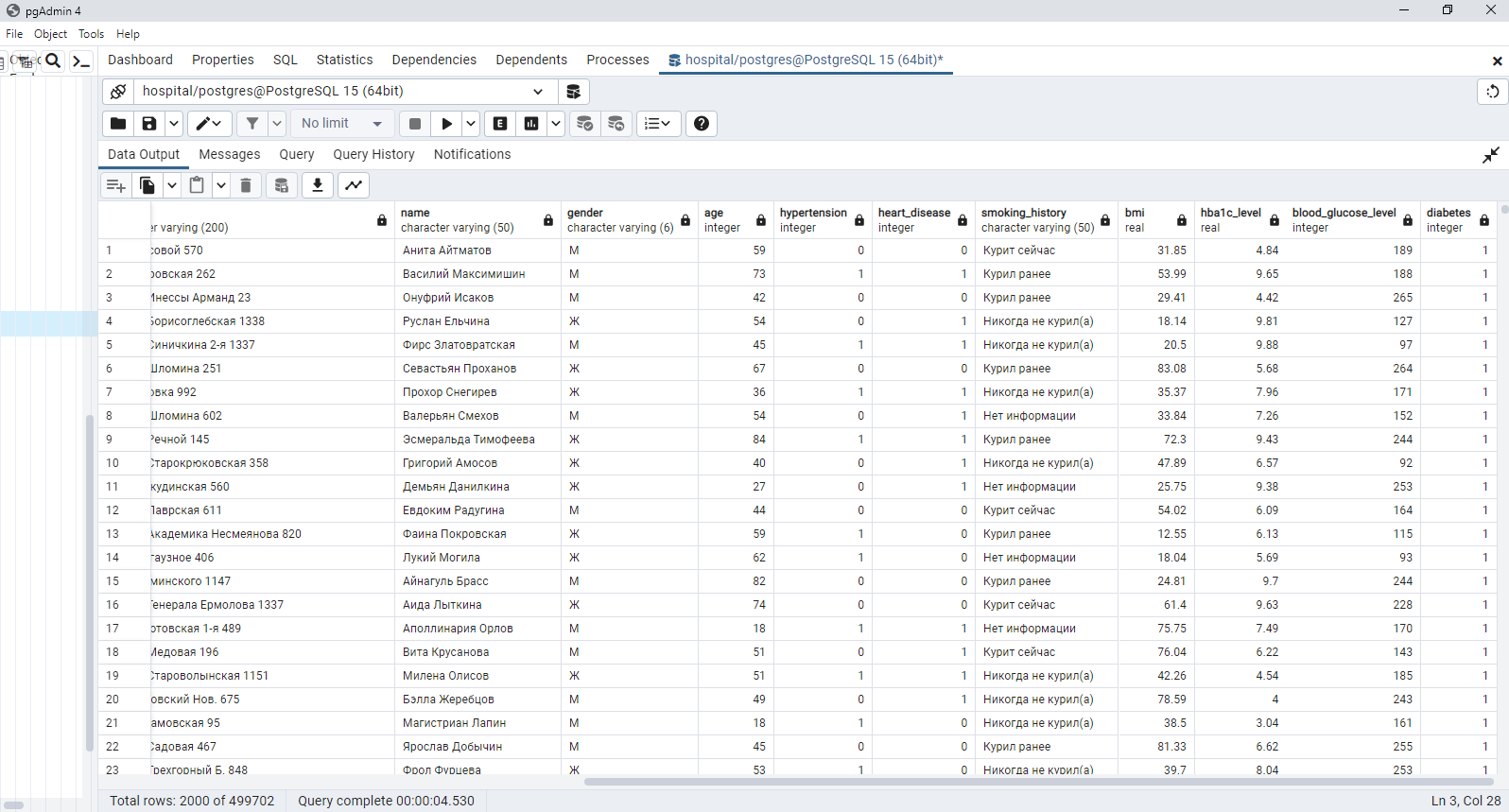
**Запрос с поиском всех данных из таблиц hospital и patient о всех больных диабетом (левая и правая часть, полученной таблицы):**

SELECT \* FROM hospital

FULL JOIN patient on hospital.id\_hospital = patient.hospital

WHERE patient.diabetes = '1'



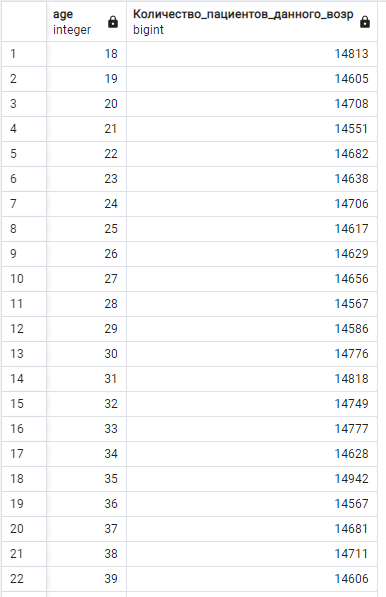


**Запрос с использованием агрегатной функции:**

SELECT age, COUNT(\*) as Количество\_пациентов\_данного\_возраста

FROM patient

GROUP BY patient.age

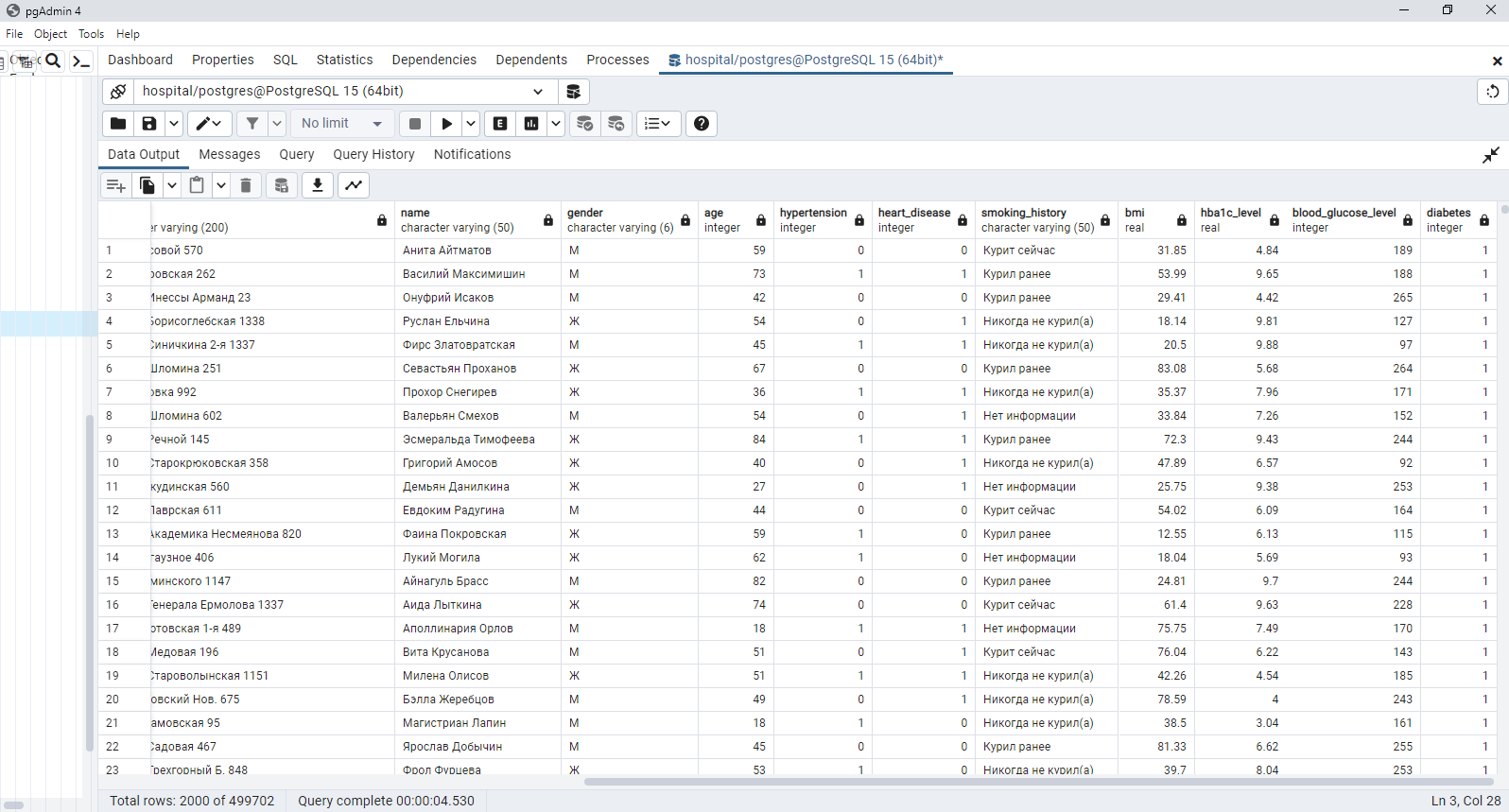


**Будем улучшать производительность данного запроса:**

SELECT \* FROM hospital

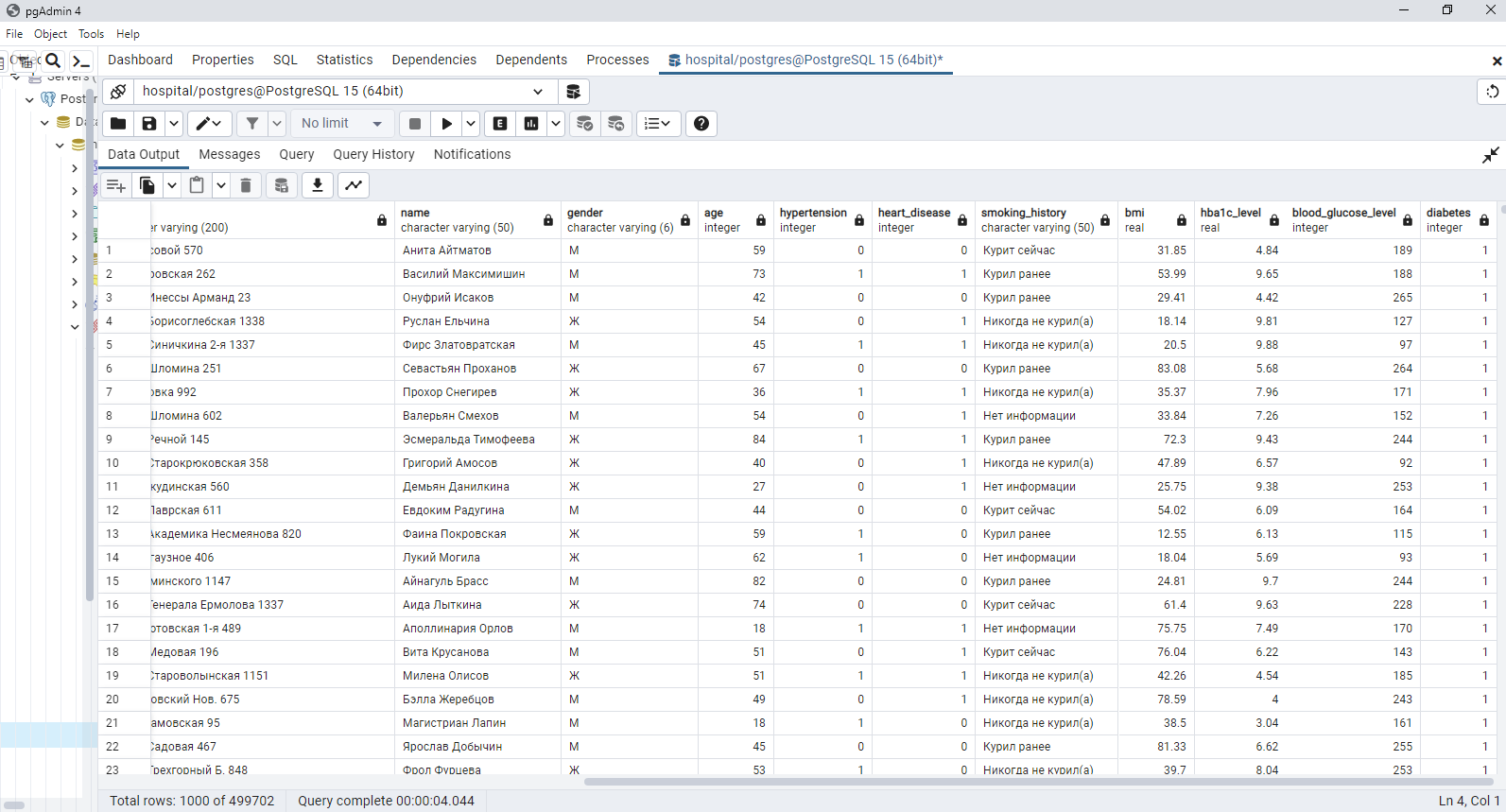
FULL JOIN patient on hospital.id\_hospital = patient.hospital

WHERE patient.diabetes = '1'



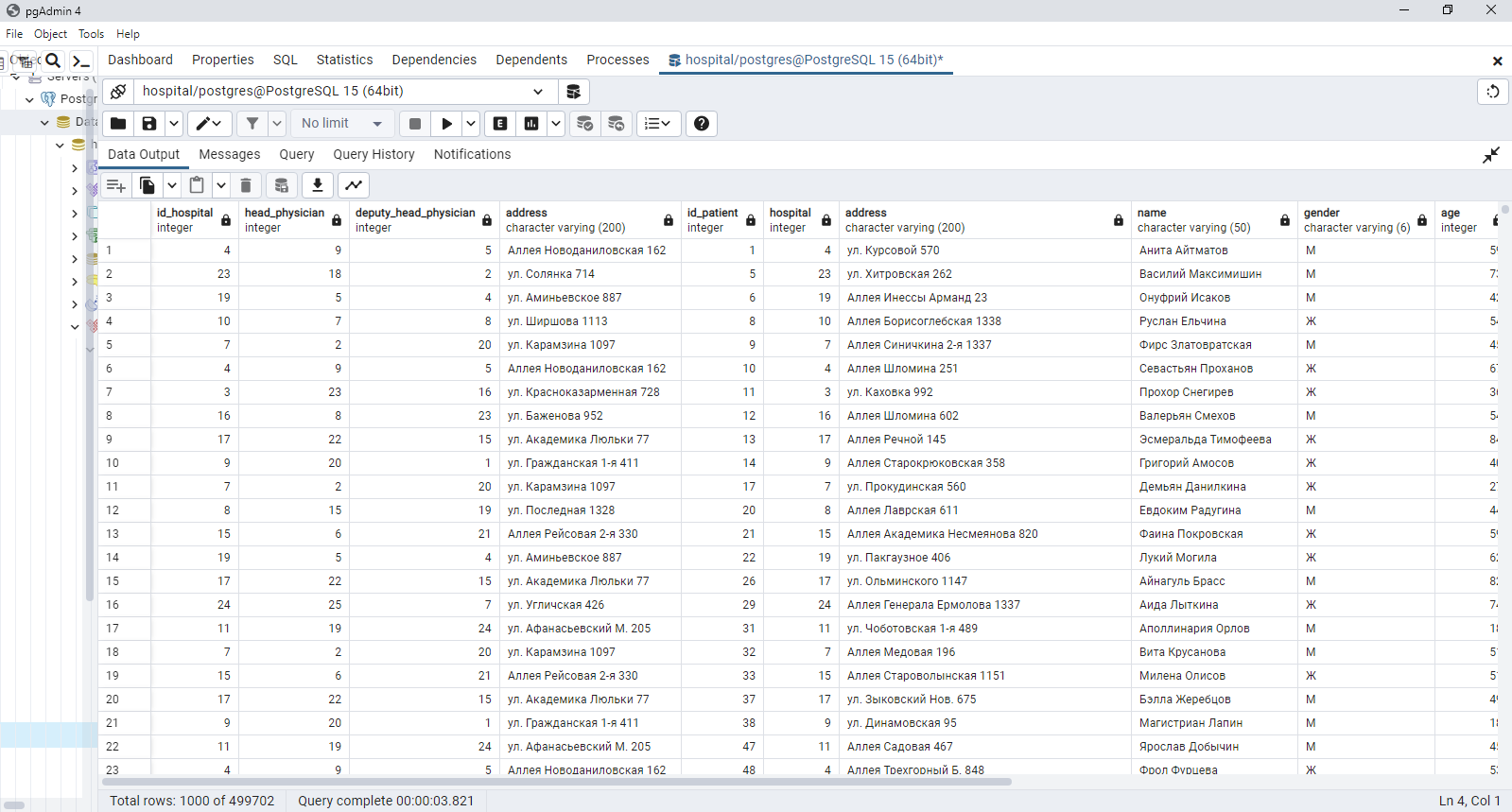
Время данного запроса 4,530 с.

Изменим shared\_buffers = 512MB, тут указано начальное значение, попробуем выделить больше памяти, я выделил 2ГБ. Буфер PostgreSQL называется shared\_buffer, который является наиболее эффективным настраиваемым параметром для большинства операционных систем. Этот параметр устанавливает, сколько выделенной памяти будет использоваться PostgreSQL для кеширования. результат запроса стал лучше:



Теперь 4,044 с.

**effective\_cache\_size** предоставляет оценку памяти, доступной для кэширования диска. Это всего лишь ориентир, а не точный объем выделенной памяти или кеша. Он не выделяет фактическую память, но сообщает оптимизатору объем кеша, доступный в ядре. Если значение этого параметра установлено слишком низким, планировщик запросов может принять решение не использовать некоторые индексы, даже если они будут полезны. Поэтому установка большого значения всегда имеет смысл. Было 4ГБ, поставил 6ГБ. Результат стал лучше.



Можно видеть, что поиск выполнился за 3,821 с.

Теперь попробуем оптимизировать сам запрос, предположим нам важно именно имена больных людей диабетом, номер больницы и адрес больницы в которой лежат больные. Оптимизированный запрос:

SELECT patient.name, patient.diabetes, patient.hospital, hospital.id\_hospital, hospital.address as Адрес\_больницы

FROM hospital

FULL JOIN patient on hospital.id\_hospital = patient.hospital

WHERE patient.diabetes = '1'



Скорость значительно увеличилась. Время теперь составляет 2,084 с.