Санкт-Петербургский политехнический университет Петра Великого Институт компьютерных наук и технологий  
Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторным работам №1-4**

**Дисциплина**: Базы данных

Выполнил студент гр. 3530901/70203 И.Д. Иванов

(подпись)

Преподаватель А.В. Мяснов

(подпись)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Санкт-Петербург

2020

Оглавление

[1. Лабораторная работа №1 3](#_Toc42107490)

[1.1 Цели работы 3](#_Toc42107491)

[1.2 Программа работы 3](#_Toc42107492)

[1.3 Выполнение работы 3](#_Toc42107493)

[1.3.1 Ссылка на созданный на GitLab проект: 3](#_Toc42107494)

[1.3.2 Предметная область: 3](#_Toc42107495)

[1.3.3 Описание таблиц: 3](#_Toc42107496)

[1.3.4 Схема БД: 5](#_Toc42107497)

[1.3.5 Скрипт для создания БД: 6](#_Toc42107498)

[1.3.6 Скрипт для изменения БД по заданию преподавателя: 8](#_Toc42107499)

[1.3.7 Окончательная схема БД 10](#_Toc42107500)

[1.4 Вывод 10](#_Toc42107501)

[2. Лабораторная работа №2 11](#_Toc42107502)

[2.1 Цель работы 11](#_Toc42107503)

[2.2 Программа работы 11](#_Toc42107504)

[2.3 Выполнение работы 11](#_Toc42107505)

[2.3.1 Подключение к БД 11](#_Toc42107506)

[2.3.2 Генерация различных необходимых случайных значений 12](#_Toc42107507)

[2.3.3 Заполнение БД 12](#_Toc42107508)

[2.4 Вывод 22](#_Toc42107509)

[3. Лабораторная работа №3 23](#_Toc42107510)

[3.1 Цель работы 23](#_Toc42107511)

[3.2 Программа работы 23](#_Toc42107512)

[3.3 Выполнение работы 23](#_Toc42107513)

[3.3.1 Стандартные запросы 23](#_Toc42107514)

[3.3.2 Индивидуальные задания 31](#_Toc42107515)

[3.4 Вывод 32](#_Toc42107516)

[4. Лабораторная работа №4 33](#_Toc42107517)

[4.1 Цель работы 33](#_Toc42107518)

[4.2 Программа работы 33](#_Toc42107519)

[4.3 Выполнение работы 33](#_Toc42107520)

[4.3.1 Типовые запросы пользователей 33](#_Toc42107521)

[4.3.2 Моделирование нагрузки БД и снятие показателей 34](#_Toc42107522)

[4.3.3 Оптимизация 37](#_Toc42107523)

[4.3.4 Сравнительный анализ результатов 38](#_Toc42107524)

[4.4 Вывод 44](#_Toc42107525)

# Лабораторная работа №1

## Цели работы

Познакомиться с языком описания сущностей и ограничений БД SQL-DDL, ознакомиться со способами организации данных в SQL-БД, а также спроектировать схему собственной базы данных и создать скрипт, генерирующего БД согласно схеме.

## Программа работы

* Создание проекта для работы в GitLab
* Выбор задания (предметной области), описание набора данных и требований к хранимым данным
* Формирование в свободном формате схемы БД, соответствующей заданию
* Создание скрипта, генерирующего БД согласно схеме
* Изменение скрипта создания БД согласно требованиям преподавателя

## Выполнение работы

### Ссылка на созданный на GitLab проект:

<http://gitlab.icc.spbstu.ru/BigAwesomeTurtle/bar-db>

### Предметная область:

База данных для сети баров.

### Описание таблиц:

1. menu\_item

* id – порядковый номер в таблице
* name – название позиции
* description – описание позиции
* weight – вес
* category\_id – принадлежность к какой-либо категории (например, салаты, холодные закуски и т.д.)

1. menu\_category

* id – порядковый номер категории в таблице
* name – название категории
* adult – является ли категория доступной только для совершеннолетних

1. regular\_customer

* id – порядковый номер клиента в таблице
* name – имя клиента
* telephone – телефон клиента
* discount – скидка клиента
* points – накопленные бонусные баллы
* favourite\_bar\_id – id любимого бара

1. bar

* id – порядковый номер бара в таблице
* address – адрес бара
* admin\_id – порядковый номер администратора бара
* telephone – телефон бара
* profit – прибыль бара

1. order\_history

* id – порядковый номер заказа в таблице
* bar\_id – id бара, а котором произошёл заказ
* date\_and\_time – дата и время заказа

1. worker

* id – порядковый номер сотрудника в таблице
* name – имя сотрудника
* bar\_id – id бара, в котором работает сотрудник
* telephone – телефон сотрудника
* bonus – премия сотрудника
* entry\_date – дата начала работы
* position\_id – id должности
* address – адрес проживания сотрудника
* schedule – расписание работы

1. position

* id – порядковый номер должности в таблице
* name – название должности
* salary – зарплата сотрудников на данной должности
* responsibilities – обязанности
* discount – скидка сотрудников на данной должности

1. worker\_history

* id – порядковый номер в таблице
* worker\_id – id сотрудника
* date\_from – дата начала работы сотрудника
* date\_to – дата окончания работы сотрудника
* bar\_id – id бара, в котором работал сотрудник
* position\_id – id должности

1. promotion

* id – порядковый номер акции в таблице
* date\_from – дата начала акции
* date\_to – дата окончания акции
* category\_id – id категории, на которую распространяется акция
* menu\_item\_id – id позиции, на которую распространяется акция
* terms – условия акции
* discount – размер скидки

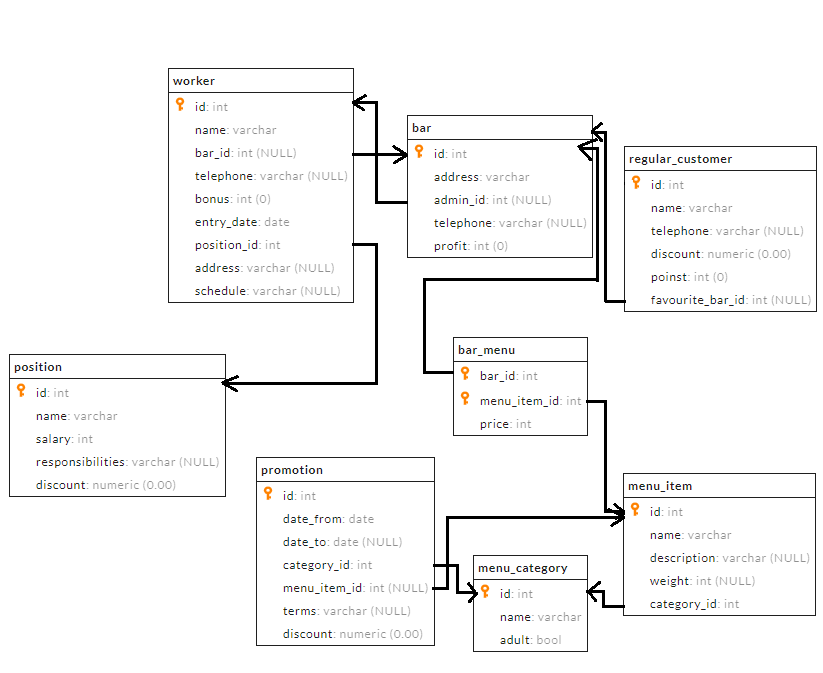
1. order\_items

* order\_id – id заказа
* menu\_item\_id – id позиции
* price – цена позиции
* amount – количество заказанных элементов позиции

1. bar\_menu

* bar\_id – id бара
* menu\_item\_id – id позиции
* price – цена позиции

### Схема БД:



*Рис.1. Схема БД*

### Скрипт для создания БД:

Листинг 1.

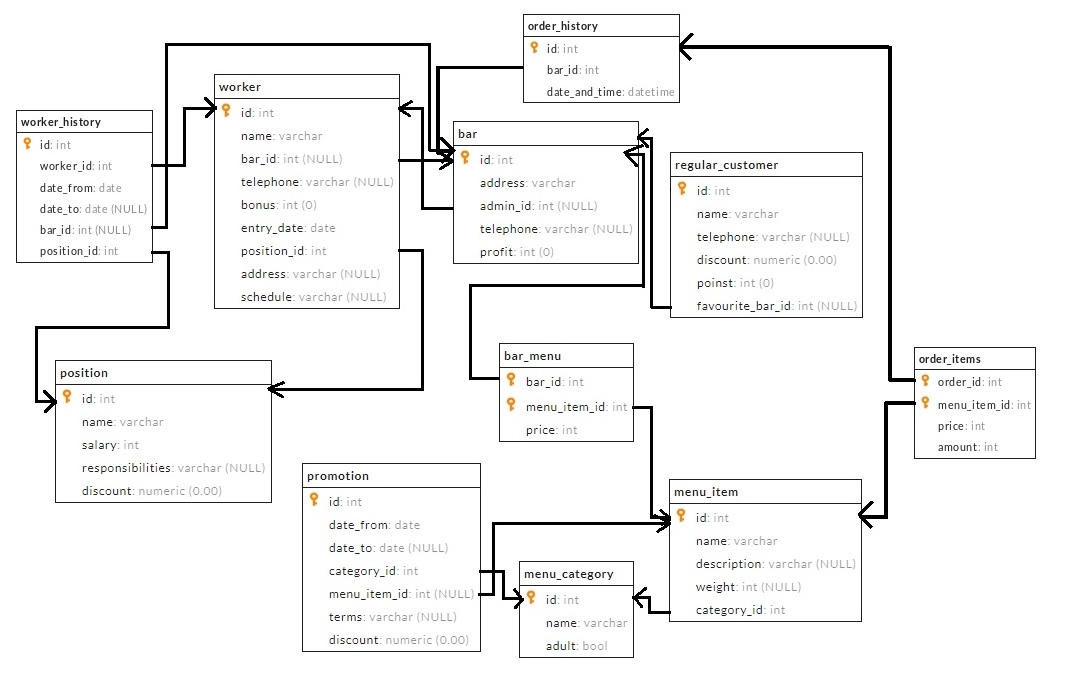
|  |
| --- |
| DROP DATABASE IF EXISTS bar\_db;  CREATE DATABASE bar\_db;  \c bar\_db  START TRANSACTION;  CREATE TABLE IF NOT EXISTS public.position (  id SERIAL NOT NULL,  name VARCHAR(50) NOT NULL,  salary INT NOT NULL,  responsibilities VARCHAR(200) NULL,  discount NUMERIC(4,2) DEFAULT 0.00 NOT NULL,  PRIMARY KEY (id)  );  CREATE TABLE IF NOT EXISTS public.bar (  id SERIAL NOT NULL,  address VARCHAR(100) NOT NULL,  admin\_id INT NULL,  telephone VARCHAR(15) NULL,  profit INT DEFAULT 0 NOT NULL,  PRIMARY KEY (id)  );  CREATE TABLE IF NOT EXISTS public.worker (  id SERIAL NOT NULL,  name VARCHAR(50) NOT NULL,  bar\_id INT NULL,  telephone VARCHAR(15) NULL,  bonus INT DEFAULT 0 NOT NULL,  entry\_date DATE NOT NULL,  position\_id INT NOT NULL,  address VARCHAR(100) NULL,  schedule VARCHAR(50) NULL,  PRIMARY KEY (id),  CONSTRAINT worker\_bar\_id\_foreign  FOREIGN KEY (bar\_id)  REFERENCES public.bar (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT worker\_position\_id\_foreign  FOREIGN KEY (position\_id)  REFERENCES public.position (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION  );  ALTER TABLE bar ADD CONSTRAINT bar\_admin\_id\_foreign  FOREIGN KEY (admin\_id)  REFERENCES public.worker (id)  ON DELETE CASCADE  ON UPDATE NO ACTION;  CREATE TABLE IF NOT EXISTS public.menu\_category (  id SERIAL NOT NULL,  name VARCHAR(50) NOT NULL,  adult BOOL NOT NULL,  PRIMARY KEY (id)  );  CREATE TABLE IF NOT EXISTS public.menu\_item (  id SERIAL NOT NULL,  name VARCHAR(50) NOT NULL,  description VARCHAR(400) NULL,  weight NUMERIC NULL,  category\_id INT NOT NULL,  recomended\_price INT NOT NULL,  PRIMARY KEY (id),  CONSTRAINT menu\_item\_category\_id\_foreign  FOREIGN KEY (category\_id)  REFERENCES public.menu\_category (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION  );  CREATE TABLE IF NOT EXISTS public.bar\_menu (  bar\_id INT NOT NULL,  menu\_item\_id INT NOT NULL,  price INT NOT NULL,  PRIMARY KEY (bar\_id, menu\_item\_id),  CONSTRAINT bar\_menu\_bar\_id\_foreign  FOREIGN KEY (bar\_id)  REFERENCES public.bar (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT bar\_menu\_menu\_item\_id\_foreign  FOREIGN KEY (menu\_item\_id)  REFERENCES public.menu\_item (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION  );  CREATE TABLE IF NOT EXISTS public.promotion (  id SERIAL NOT NULL,  date\_from DATE NOT NULL,  date\_to DATE NULL,  category\_id INT NOT NULL,  menu\_item\_id INT NULL,  terms VARCHAR(400) NULL,  discount NUMERIC(4,2) DEFAULT 0.00 NOT NULL,  PRIMARY KEY (id),  CONSTRAINT promotion\_category\_id\_foreign  FOREIGN KEY (category\_id)  REFERENCES public.menu\_category (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT promotion\_menu\_item\_id\_foreign  FOREIGN KEY (menu\_item\_id)  REFERENCES public.menu\_item (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION  );  CREATE TABLE IF NOT EXISTS public.regular\_customer (  id SERIAL NOT NULL,  login VARCHAR(20) UNIQUE NOT NULL ,  password VARCHAR(512) NOT NULL,  name VARCHAR(50) NOT NULL,  telephone VARCHAR(15) NULL,  discount NUMERIC(4,2) DEFAULT 0.00 NOT NULL,  points INT DEFAULT 0 NOT NULL,  favourite\_bar\_id INT NULL,  PRIMARY KEY (id),  CONSTRAINT regular\_customer\_id\_foreign  FOREIGN KEY (favourite\_bar\_id)  REFERENCES public.bar (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION  );  COMMIT; |

### Скрипт для изменения БД по заданию преподавателя:

Листинг 2.

|  |
| --- |
| \c bar\_db  START TRANSACTION;  DROP TABLE IF EXISTS public.order\_history;  CREATE TABLE IF NOT EXISTS public.order\_history (  id SERIAL NOT NULL,  bar\_id INT NOT NULL,  date\_and\_time TIMESTAMP NOT NULL,  customer\_name VARCHAR(50) NULL,  PRIMARY KEY (id),  CONSTRAINT order\_history\_bar\_id\_foreign  FOREIGN KEY (bar\_id)  REFERENCES public.bar (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION  );  DROP TABLE IF EXISTS public.order\_items;  CREATE TABLE IF NOT EXISTS public.order\_items (  order\_id INT NOT NULL,  menu\_item\_id INT NOT NULL,  price INT NOT NULL,  amount INT DEFAULT 1 NOT NULL,  PRIMARY KEY (order\_id, menu\_item\_id),  CONSTRAINT order\_items\_order\_id\_foreign  FOREIGN KEY (order\_id)  REFERENCES public.order\_history (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT order\_items\_menu\_item\_id\_foreign  FOREIGN KEY (menu\_item\_id)  REFERENCES public.menu\_item (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION  );  DROP TABLE IF EXISTS public.worker\_history;  CREATE TABLE IF NOT EXISTS public.worker\_history (  id SERIAL NOT NULL,  worker\_id INT NOT NULL,  date\_from DATE NOT NULL,  date\_to DATE NULL,  bar\_id INT NULL,  position\_id INT NOT NULL,  salary INT NOT NULL,  PRIMARY KEY (id),  CONSTRAINT worker\_history\_worker\_id\_foreign  FOREIGN KEY (worker\_id)  REFERENCES public.worker (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT worker\_history\_bar\_id\_foreign  FOREIGN KEY (bar\_id)  REFERENCES public.bar (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT worker\_history\_position\_id\_foreign  FOREIGN KEY (position\_id)  REFERENCES public.position (id)  ON DELETE NO ACTION  ON UPDATE NO ACTION  );  COMMIT; |

### Окончательная схема БД



*Рис.2. Окончательный вид схемы БД*

## Вывод

В ходе выполнения данной лабораторной работы был создан проект на GitLab, выбрана предметная область для создания базы данных, описан набор данных и требований к хранимым данным, сформирована схема БД в графическом формате. Кроме того, было осуществлено знакомство с языком SQL, с помощью которого был разработан скрипт для создания БД, а также скрипт изменения БД согласно заданию преподавателя.

# Лабораторная работа №2

## Цель работы

Сформировать набор данных, позволяющий производить операции на реальных объемах данных.

## Программа работы

* + Реализация в виде программы параметризуемого генератора, который позволит сформировать набор связанных данных в каждой таблице.
  + Частные требования к генератору, набору данных и результирующему набору данных:
    - количество записей в справочных таблицах должно соответствовать ограничениям предметной области
    - количество записей в таблицах, хранящих информацию об объектах или субъектах должно быть параметром генерации
    - значения для внешних ключей необходимо брать из связанных таблиц
    - сохранение уже имеющихся данных в базе данных=

## Выполнение работы

### Подключение к БД

Для выполнения лабораторной работы был выбран язык Python. Подключение к базе данных осуществляется с помощью библиотеки psycopg2.

Пример подключения к БД и создания “курсора”, через который осуществляется взаимодействие с базой:

connection=psycopg2.connect(dbname='bar\_db', user='bar\_user', password='barbar', host='127.0.0.1' )

cursor=connection.cursor()

### Генерация различных необходимых случайных значений

Листинг 3.

|  |
| --- |
| def generate\_schedule():  week\_list=dicts.week\_list  day\_from = random.randint(0, len(week\_list)-1)  day\_to = random.randint(day\_from, len(week\_list)-1)  time\_from = random.randint(0,25)  time\_to = random.randint(0,25)  return "{}-{}, {}:00-{}:00".format(week\_list[day\_from],week\_list[day\_to],time\_from,time\_to)  def generate\_phone():  first = str(random.randint(900,999))  second = str(random.randint(1,888)).zfill(3)  last = (str(random.randint(1,9998)).zfill(4))  return '7{}{}{}'.format(first,second, last)  def get\_random\_date(start, end):  if(start.toordinal() >= end.toordinal()):  return start  random\_day = date.fromordinal(random.randint(start.toordinal(), end.toordinal()))  return random\_day  def random\_date\_and\_time(start, end):  return start + timedelta(seconds=random.randint(0, int((end - start).total\_seconds())), ) |

Функция generate\_schedule генерирует расписание для сотрудника бара.

Функция generate\_phone генерирует номер телефона в формате 7 \*\*\* \*\*\* \*\*\*\*.

Функция get\_random\_date генерирует случайную дату.

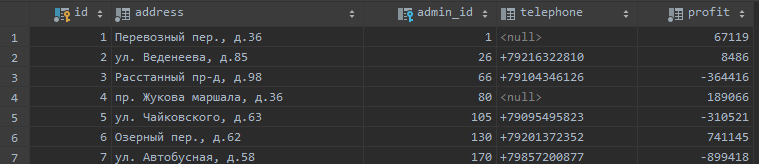
Функция random\_date\_and\_time нужна для генерации случайной даты с временем.

### Заполнение БД

#### Заполнение таблицы bar

Листинг 4.

|  |
| --- |
| def bar\_generation(amount):  insert\_addresses=""  addr=[]  for i in range(amount):  address=str(random.choice(addresses))  address=address[0: len(address)-1] + ", д." + str(random.randint(1,99))  addr.append(address)  insert\_addresses=insert\_addresses + "(\'"+address+"\'),"  exec\_str='INSERT INTO public.bar (address) VALUES {}'.format(insert\_addresses)  cursor.execute(exec\_str[:len(exec\_str)-1])  for i in range(amount):  has\_teleph=random.randint(1,5) != 1  has\_profit=random.randint(1,5) != 1  if(has\_teleph):  cursor.execute('UPDATE public.bar SET telephone=%s WHERE address=%s',(generate\_phone(),addr[i]))  if(has\_profit):  cursor.execute('UPDATE public.bar SET profit=%s WHERE address=%s',  (str(random.randint(-1000000, 1000000)), addr[i])) |

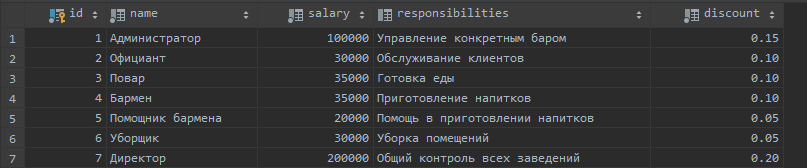


*Рис.3. Пример результатов заполнения таблицы bar*

#### Заполнение таблицы position

Листинг 5.

|  |
| --- |
| def position\_generation():  position\_list = dicts.position\_list  insert\_position=""  for elem in position\_list:  insert\_position=insert\_position + "( "+"\'"+elem["name"]+"\'"+", "+str(elem["salary"])+", "+"\'"+elem["respons"]+"\'"+", "+str(elem["discount"])+"),"  exec\_str='INSERT INTO public.position (name, salary, responsibilities, discount) VALUES {}'.format(insert\_position)  cursor.execute(exec\_str[:len(exec\_str)-1]) |

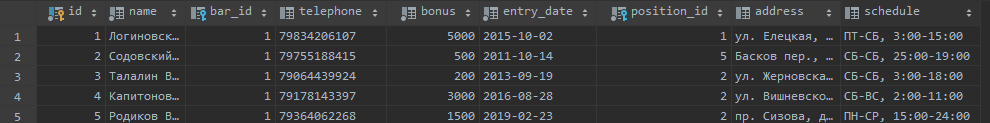


*Рис.4. Пример результатов заполнения таблицы position*

#### Заполнение таблицы worker

Листинг 6.

|  |
| --- |
| def worker\_generation(adding\_flag, worker\_amount=0, bar\_args=0, position\_args=0):    cursor.execute("SELECT count(id) FROM bar")  bar\_amount=cursor.fetchone()[0]  cursor.execute("SELECT count(id) FROM position")  position\_amount=cursor.fetchone()[0]  insert\_worker=""  if not adding\_flag:  for i in range(1, bar\_amount+1):    bar\_size=random.randint(0,2)  insert\_worker+=worker\_sql("Администратор",i)    waiter\_size=dicts.bar\_personal\_size[bar\_size]["waiter\_size"]  cook\_size=dicts.bar\_personal\_size[bar\_size]["cook\_size"]  barmen\_size=dicts.bar\_personal\_size[bar\_size]["barmen\_size"]  insert\_worker+=worker\_sql("Помощник бармена",i)  for j in range(0,waiter\_size):  insert\_worker+= worker\_sql("Официант",i)  for j in range(0,cook\_size):  insert\_worker+= worker\_sql("Повар",i)  insert\_worker+= worker\_sql("Уборщик",i)  for j in range(0,barmen\_size):  insert\_worker+= worker\_sql("Бармен",i)  insert\_worker+= worker\_sql("Директор",i)  else:  bar\_list=()  position\_list=()  if(bar\_args!=0):  bar\_list=bar\_args.split(',')  else:  bar\_list = list(range(1,bar\_amount+1))  if(position\_args!=0):  position\_list=position\_args.split(',')  else:  position\_list=list(range(2,position\_amount+1))  for i in range(0,int(worker\_amount)):  insert\_worker+= worker\_sql(int(random.choice(position\_list)),int(random.choice(bar\_list)))  exec\_str='INSERT INTO public.worker (name, bar\_id, telephone, bonus, entry\_date, position\_id, address, schedule) VALUES {}'.format(insert\_worker)  cursor.execute(exec\_str[:len(exec\_str)-1])  update\_admin()  def worker\_sql(pos,id):  pos\_index=0  if type(pos)==type(""):  for index,elem in enumerate(dicts.position\_list):  if pos == elem["name"]:  pos\_index=index+1  else:  pos\_index=pos  address=str(random.choice(addresses))  address=address[0: len(address)-1] + ", д." + str(random.randint(1,99)) + ", кв " + str(random.randint(1,1000))  insert\_worker="( "+"\'"+str(random.choice(names))+"\'"+", "+str(id)+", "+generate\_phone()+", "+str(random.choice(dicts.bonus\_list))+", "+"\'"+str(get\_random\_date(opening\_date,  date.today()))+"\'"+", "+str(pos\_index)+", "+"\'"+address+"\'"+", "+"\'"+generate\_schedule()+"\'"+"),"    return insert\_worker  def update\_admin():  cursor.execute("SELECT id, bar\_id FROM worker WHERE position\_id=1")  needed\_updates=cursor.fetchall()  for elem in needed\_updates:  cursor.execute('UPDATE public.bar SET admin\_id=%s WHERE id=%s',(elem[0],elem[1])) |

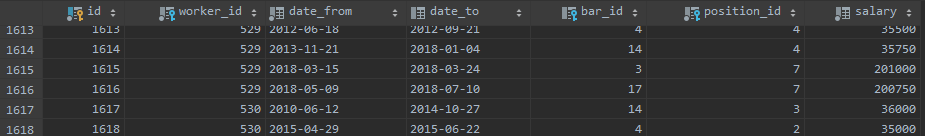


*Рис.5. Пример результатов заполнения таблицы worker*

#### Заполнение таблицы worker\_history

Листинг 7.

|  |
| --- |
| def worker\_history\_generation(adding\_flag, history\_amount=0, worker\_args=0, bar\_args=0, position\_args=0):  if worker\_args==0:  cursor.execute("SELECT count(id) FROM worker")  amount=cursor.fetchone()[0]  else:  worker\_args=worker\_args.split(',')  amount=len(worker\_args)  cursor.execute("SELECT count(id) FROM bar")  bar\_amount=cursor.fetchone()[0]  cursor.execute("SELECT count(id) FROM position")  position\_amount=cursor.fetchone()[0]  insert\_history=""  for i in range(1,amount+1):  if not adding\_flag:  number\_of\_works=random.randint(0,4)  else:  number\_of\_works=math.floor(int(history\_amount)/amount)  reminder=int(int(history\_amount)%amount)  if i>(amount-reminder):  number\_of\_works+=1  date\_begin = opening\_date  curr\_id=-1;  if(worker\_args==0):  curr\_id=i  else:  curr\_id=worker\_args[i-1]  cursor.execute("SELECT entry\_date, bar\_id, position\_id FROM public.worker WHERE id=%s",(curr\_id,))  curr\_job=cursor.fetchone()  for j in range(0, number\_of\_works):  date\_from = get\_random\_date(date\_begin, curr\_job[0] - timedelta(days=1))  date\_to=get\_random\_date(date\_begin, curr\_job[0] - timedelta(days=1))  while(date.toordinal(date\_from)>date.toordinal(date\_to)):  date\_from = get\_random\_date(date\_begin, curr\_job[0] - timedelta(days=1))  date\_to=get\_random\_date(date\_begin, curr\_job[0] - timedelta(days=1))    if(position\_args==0):  random\_position=random.randint(1, position\_amount)  else:  position\_args\_=position\_args.split(',')  random\_position=random.choice(position\_args\_)  if(bar\_args==0):  random\_bar=random.randint(1, bar\_amount)  else:  bar\_args\_=bar\_args.split(',')  random\_bar=random.choice(bar\_args\_)  cursor.execute("SELECT salary FROM public.position WHERE id=%s",(random\_position,))  salary=cursor.fetchone()[0] + random.choice(dicts.bonus\_list)  insert\_history=insert\_history + "( "+str(curr\_id)+", "+"\'"+str(date\_from)+"\'"+", "+"\'"+str(date\_to)+"\'"+", "+str(random\_bar)+", "+"\'"+str(random\_position)+"\'"+", "+str(salary)+"),"  date\_begin=date\_to  if not adding\_flag:  cursor.execute("SELECT salary FROM public.position WHERE id=%s",(curr\_job[2],))  salary=cursor.fetchone()[0]  cursor.execute("SELECT bonus FROM public.worker WHERE id=%s",(i,))  salary+=cursor.fetchone()[0]  cursor.execute('INSERT INTO public.worker\_history (worker\_id, date\_from, bar\_id, position\_id, salary) VALUES (%s,%s,%s,%s,%s)',  (curr\_id, str(curr\_job[0]), curr\_job[1],curr\_job[2],salary))  exec\_str='INSERT INTO public.worker\_history (worker\_id, date\_from, date\_to, bar\_id, position\_id, salary) VALUES {}'.format(insert\_history)  cursor.execute(exec\_str[:len(exec\_str)-1]) |

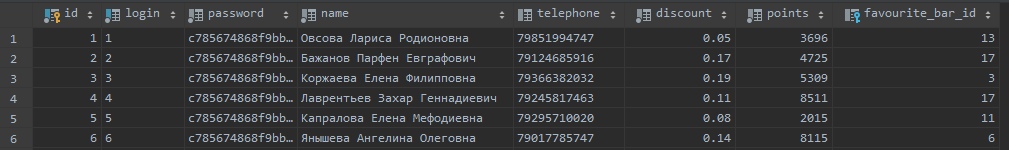


*Рис.6. Пример результатов заполнения таблицы worker\_history*

#### Заполнение таблицы regular\_customer

Листинг 8.

|  |
| --- |
| def regular\_customer\_generation(customer\_amount, bar\_args=0):    if(bar\_args==0):  cursor.execute("SELECT count(id) FROM bar")  bar\_amount=cursor.fetchone()[0]  else:  bar\_args=bar\_args.split(',')  random\_bar=random.choice(bar\_args)  insert\_regular=""  for i in range(0, int(customer\_amount)):  if(bar\_args==0):  random\_bar=random.randint(1, bar\_amount)  else:  random\_bar=random.choice(bar\_args)  insert\_regular=insert\_regular + "( "+"\'"+random.choice(names)+"\'"+", "+ str(i)+", "+"\'"+"c785674868f9bb555c26d09e02509056f28a8bfd1b8dc296978d4db523778d2842c811f3722633520abdc8d6d25989d8536d168b666acce6cda38f759a8a1949"+"\'"+", "+generate\_phone()+", "+str(random.randint(1,20)/100)+", "+str(random.randint(1,10000))+", "+str(random\_bar)+"),"  exec\_str='INSERT INTO public.regular\_customer (name, login, password, telephone, discount, points, favourite\_bar\_id) VALUES {}'.format(insert\_regular)  cursor.execute(exec\_str[:len(exec\_str)-1]) |

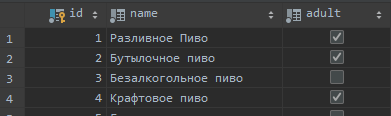


*Рис.7. Пример результатов заполнения таблицы regular\_customer*

#### Заполнение таблицы menu\_category

Листинг 9.

|  |
| --- |
| def menu\_category\_generation():  category\_list = dicts.category\_list  insert\_category=""  for elem in category\_list:  insert\_category=insert\_category + "( "+"\'"+elem["name"]+"\'"+", "+str(elem["adult"])+"),"  exec\_str='INSERT INTO public.menu\_category (name, adult) VALUES {}'.format(insert\_category)  cursor.execute(exec\_str[:len(exec\_str)-1]) |

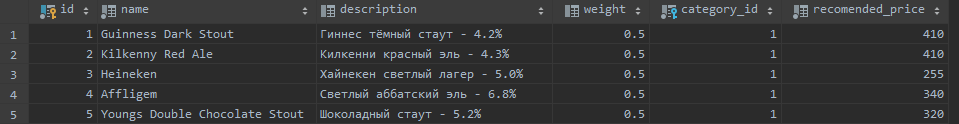


*Рис.8. Пример результатов заполнения таблицы menu\_category*

#### Заполнение таблицы menu\_item

Листинг 10.

|  |
| --- |
| def menu\_item\_generation():  menu\_list=dicts.menu\_list  insert\_item=""  for i in range(0,len(menu\_list)):  for elem in menu\_list[i]:  insert\_item=insert\_item + "( "+"\'"+elem["name"]+"\'"+", "+"\'"+elem["description"]+"\'"+", "+ "\'"+str(elem["weight"])+"\'"+", "+str(i+1)+", "+str(elem["recomended\_price"])+"),"  exec\_str='INSERT INTO public.menu\_item ( name, description, weight, category\_id, recomended\_price) VALUES {}'.format(insert\_item)  cursor.execute(exec\_str[:len(exec\_str)-1]) |

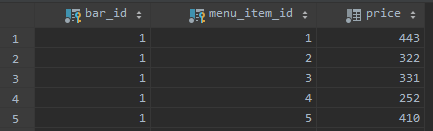


*Рис.9. Пример результатов заполнения таблицы menu\_item*

#### Заполнение таблицы bar\_menu

Листинг 11.

|  |
| --- |
| def bar\_menu\_generation(adding\_flag, bar\_args=0):  cursor.execute("SELECT count(id) FROM menu\_item")  item\_amount=cursor.fetchone()[0]  if not adding\_flag:  cursor.execute("SELECT count(id) FROM bar")  bar\_amount=cursor.fetchone()[0]  else:  bar\_amount=1    insert\_bar\_menu=""  for i in range(1, bar\_amount+1):  if not adding\_flag:  curr\_bar=i  else:  curr\_bar=bar\_args  for j in range(1, item\_amount+1):  add\_item=random.randint(0,9)  if(add\_item!=0):  cursor.execute("SELECT recomended\_price FROM public.menu\_item WHERE id=%s",(j,))  price=cursor.fetchone()[0]  price=int(price\*random.uniform(0.6,1.5))  insert\_bar\_menu=insert\_bar\_menu + "( "+str(curr\_bar)+", "+str(j)+", "+ str(price)+"),"  exec\_str='INSERT INTO public.bar\_menu ( bar\_id, menu\_item\_id, price) VALUES {}'.format(insert\_bar\_menu)  cursor.execute(exec\_str[:len(exec\_str)-1]) |

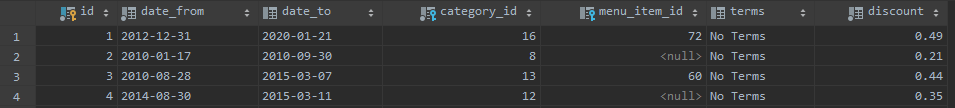


*Рис.10. Пример результатов заполнения таблицы bar\_menu*

#### Заполнение таблицы promotion

Листинг 12.

|  |
| --- |
| def promotion\_generation(current\_amount, old\_amount, category\_args=0, menu\_item\_args=0):    if(category\_args==0):  cursor.execute("SELECT count(id) FROM menu\_category")  category\_amount=cursor.fetchone()[0]  else:  category\_args=category\_args.split(',')  if(menu\_item\_args==0):  cursor.execute("SELECT count(id) FROM menu\_item")  item\_amount=cursor.fetchone()[0]  else:  menu\_item\_args=menu\_item\_args.split(',')  insert\_promo=""  for i in range(0, int(old\_amount)):  date\_from = get\_random\_date(opening\_date, date.today()-timedelta(days=1))  date\_to=get\_random\_date(opening\_date, date.today()-timedelta(days=1))  while(date.toordinal(date\_from)>=date.toordinal(date\_to)):  date\_from = get\_random\_date(opening\_date, date.today()-timedelta(days=1))  date\_to=get\_random\_date(opening\_date, date.today()-timedelta(days=1))  date\_to\_str= "\'"+str(date\_to)+"\'"  if(category\_args==0):  category\_id,menu\_item\_id = category\_or\_item(category\_amount,item\_amount)  elif menu\_item\_args==0:  category\_id=int(random.choice(category\_args))  menu\_item\_id="NULL"  else:  category\_id= int(random.choice(category\_args))  menu\_item\_id= int(random.choice(menu\_item\_args))  date\_to\_str= "\'"+str(date\_to)+"\'"  insert\_promo=insert\_promo + "( "+"\'"+str(date\_from)+"\'"+", "+date\_to\_str+", "+ str(category\_id)+", "+str(menu\_item\_id)+", "+"\'No Terms\'"+", "+str(random.randint(1,50)/100)+"),"  for i in range(0, int(current\_amount)):  date\_from = get\_random\_date(opening\_date, date.today())  date\_to\_check=random.randint(0,9)  if(date\_to\_check!=0):  date\_to=get\_random\_date(date.today(), date.today()+timedelta(days=365))  date\_to\_str= "\'"+str(date\_to)+"\'"  else:  date\_to="NULL"  if(category\_args==0):  category\_id,menu\_item\_id = category\_or\_item(category\_amount,item\_amount)  elif menu\_item\_args==0:  category\_id=int(random.choice(category\_args))  menu\_item\_id="NULL"  else:  category\_id= int(random.choice(category\_args))  menu\_item\_id= int(random.choice(menu\_item\_args))  insert\_promo=insert\_promo + "( "+"\'"+str(date\_from)+"\'"+", "+date\_to\_str+", "+ str(category\_id)+", "+str(menu\_item\_id)+", "+"\'No Terms\'"+", "+str(random.randint(1,50)/100)+"),"  exec\_str='INSERT INTO public.promotion (date\_from, date\_to, category\_id, menu\_item\_id, terms, discount) VALUES {}'.format(insert\_promo)  cursor.execute(exec\_str[:len(exec\_str)-1]) |

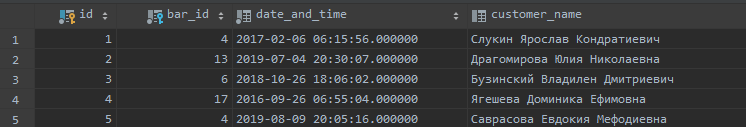


*Рис.11. Пример результатов заполнения таблицы promotion*

#### Заполнение таблицы order\_history

Листинг 13.

|  |
| --- |
| def order\_history\_generation(amount, bar\_args=0):  if(bar\_args==0):  cursor.execute("SELECT count(id) FROM bar")  bar\_amount=cursor.fetchone()[0]  else:  bar\_args=bar\_args.split(',')  insert\_order\_hist=""  for i in range(0, int(amount)):  if(bar\_args==0):  random\_bar=random.randint(1,bar\_amount)  else:  random\_bar=int(random.choice(bar\_args))  insert\_order\_hist=insert\_order\_hist + "( "+str(random\_bar)+", "+"\'"+str(random\_date\_and\_time(date.today().replace(day=1, month=1,  year=2010),date.today()))+ " " + str(random.randint(0,23)) + ":" + str(random.randint(0,59)) + ":" + str(random.randint(0,59))+"\'"+", "+"\'"+ random.choice(names)+"\'"+"),"  exec\_str='INSERT INTO public.order\_history (bar\_id, date\_and\_time, customer\_name) VALUES {}'.format(insert\_order\_hist)  cursor.execute(exec\_str[:len(exec\_str)-1]) |

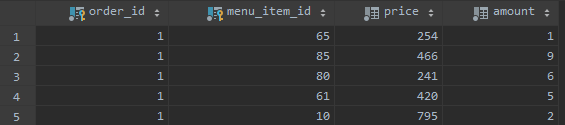


*Рис.12. Пример результатов заполнения таблицы order\_history*

#### Заполнение таблицы order\_items

Листинг 14.

|  |
| --- |
| def order\_items\_generation(adding\_flag, order\_args=0):  cursor.execute("SELECT count(id) FROM menu\_item")  item\_amount=cursor.fetchone()[0]  if not adding\_flag:  cursor.execute("SELECT count(id) FROM order\_history")  order\_amount=cursor.fetchone()[0]  else:  order\_args=order\_args.split(',')  order\_amount=len(order\_args)  insert\_order\_items=""  for i in range(1, order\_amount+1):  if not adding\_flag:  curr\_order=i  else:  curr\_order=int(order\_args[i-1])  already\_ordered=[]  position\_amount=random.randint(1,20)  for j in range(0,position\_amount):  curr\_item=random.randint(1,item\_amount)  while curr\_item in already\_ordered:  curr\_item=random.randint(1,item\_amount)  already\_ordered.append(curr\_item)  curr\_item\_amount=random.randint(1,10)  cursor.execute("SELECT bar\_id FROM order\_history WHERE id=%s",(curr\_order,))  curr\_bar=cursor.fetchone()[0]  cursor.execute("SELECT price FROM bar\_menu WHERE bar\_id=%s AND menu\_item\_id=%s",(curr\_bar, curr\_item))  price=cursor.fetchone()  if(price==None):  continue  else:  price=price[0]  insert\_order\_items=insert\_order\_items + "( "+str(curr\_order)+", "+str(curr\_item)+", "+ str(price)+", "+str(curr\_item\_amount)+"),"    exec\_str='INSERT INTO public.order\_items (order\_id,menu\_item\_id, price, amount) VALUES {}'.format(insert\_order\_items)  cursor.execute(exec\_str[:len(exec\_str)-1]) |



*Рис.13. Пример результатов заполнения таблицы order\_items*

## Вывод

В ходе выполнения данной лабораторной работы мной был написан генератор данных для заполнения БД реальным объёмом данных.

# Лабораторная работа №3

## Цель работы

Познакомиться с языком создания запросов управления данными SQL-DML.

## Программа работы

* Изучение SQL-DML;
* Выполнение всех запросов из списка стандартных запросов;
* Получение у преподавателя и реализация SQL-запросов в соответствии с индивидуальным заданием.

## Выполнение работы

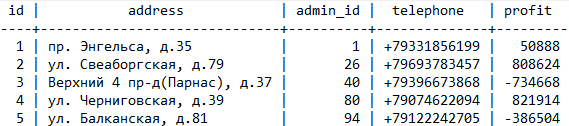
### Стандартные запросы

#### Сделайте выборку всех данных из каждой таблицы

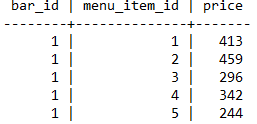
Листинг 15.

|  |
| --- |
| SELECT \* FROM bar \g 'dml\_res/1.txt'  SELECT \* FROM bar\_menu \g | cat >> 'dml\_res/1.txt'  SELECT \* FROM menu\_category \g | cat >> 'dml\_res/1.txt'  SELECT \* FROM menu\_item \g | cat >> 'dml\_res/1.txt'  SELECT \* FROM order\_history \g | cat >> 'dml\_res/1.txt'  SELECT \* FROM order\_items \g | cat >> 'dml\_res/1.txt'  SELECT \* FROM position \g | cat >> 'dml\_res/1.txt'  SELECT \* FROM promotion \g | cat >> 'dml\_res/1.txt'  SELECT \* FROM regular\_customer \g | cat >> 'dml\_res/1.txt'  SELECT \* FROM worker \g | cat >> 'dml\_res/1.txt'  SELECT \* FROM worker\_history \g | cat >> 'dml\_res/1.txt' |

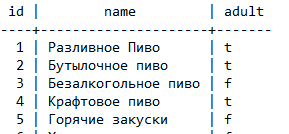
Результаты работы:



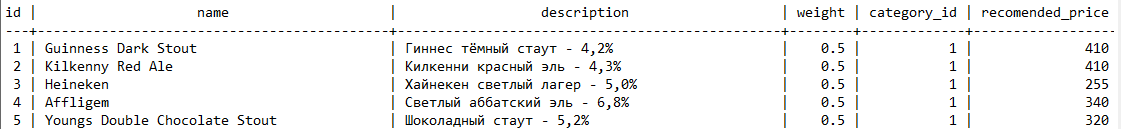
*Рис.14. Таблица bar*



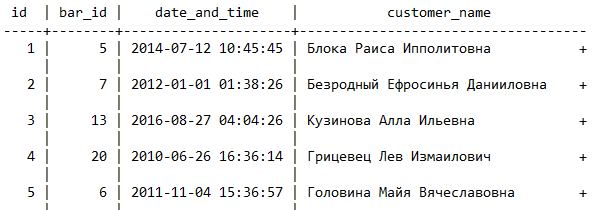
*Рис.15. Таблица bar\_menu*



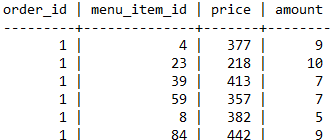
*Рис.16. Таблица menu\_category*



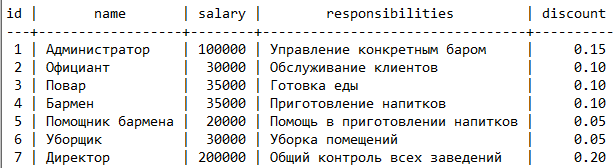
*Рис.17. Таблица menu\_item*



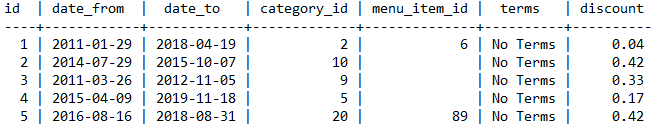
*Рис.18. Таблица order\_history*



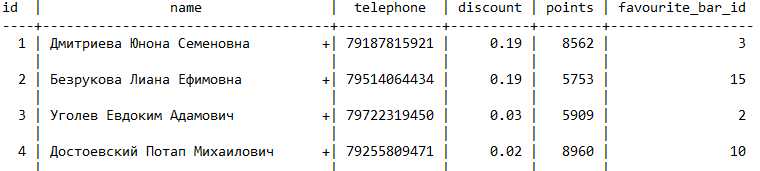
*Рис.19. Таблица order\_items*



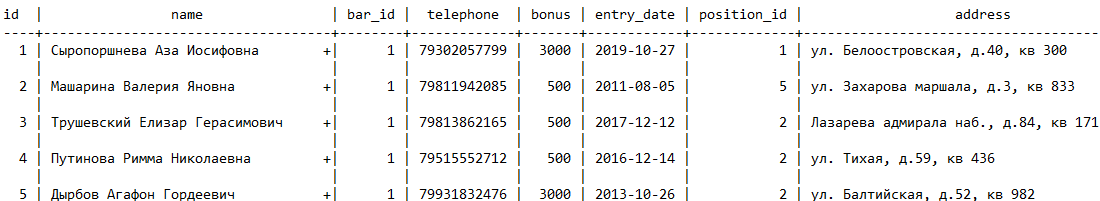
*Рис.20. Таблица position*



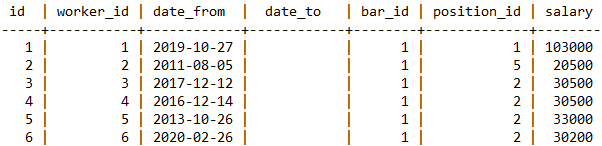
*Рис.21. Таблица promotion*



*Рис.22. Таблица regular\_customer*



*Рис.23. Таблица worker*



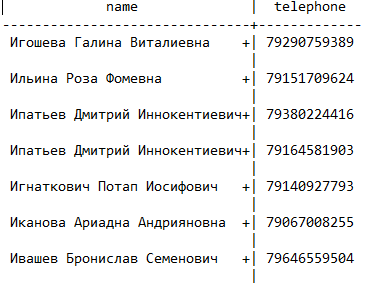
*Рис.24. Таблица worker\_history*

#### Сделайте выборку данных из одной таблицы при нескольких условиях, с использованием логических операций, LIKE, BETWEEN, IN (не менее 3-х разных примеров)

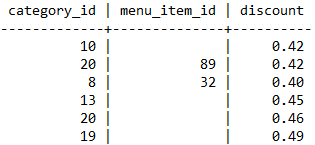
Листинг 16.

|  |
| --- |
| SELECT name,telephone  FROM regular\_customer  WHERE name LIKE 'И%' \g 'dml\_res/2.txt'  SELECT category\_id, menu\_item\_id, discount  FROM promotion  WHERE discount BETWEEN 0.4 AND 0.5 \g | cat >> 'dml\_res/2.txt'  SELECT name, salary  FROM position  WHERE salary IN (20000,35000) \g | cat >> 'dml\_res/2.txt'  SELECT name, address, bar\_id  FROM worker  WHERE address LIKE '\_п%' AND bar\_id BETWEEN 10 AND 20 \g | cat >> 'dml\_res/2.txt' |

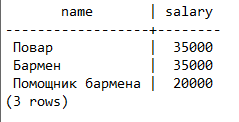
Результат:



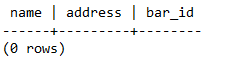
*Рис.25. Like*



*Рис.26. Between*



*Рис.27. In*



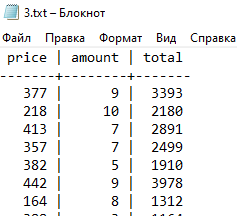
*Рис.28. Like, between*

#### Создайте в запросе вычисляемое поле

Листинг 17.

|  |
| --- |
| SELECT price, amount, (price\*amount) AS total  FROM order\_items \g 'dml\_res/3.txt' |

Результат:



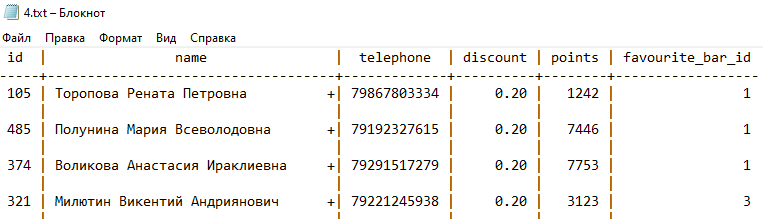
*Рис.29. Результаты запроса 3*

#### Сделайте выборку всех данных с сортировкой по нескольким полям

Листинг 18.

|  |
| --- |
| SELECT \* FROM regular\_customer  ORDER BY discount DESC, favourite\_bar\_id \g 'dml\_res/4.txt' |

Результат:



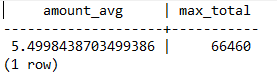
*Рис.30. Результат запроса 4*

#### Создайте запрос, вычисляющий несколько совокупных характеристик таблиц

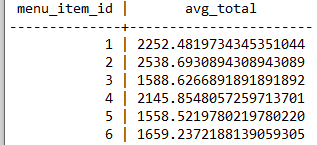
Листинг 19.

|  |
| --- |
| SELECT AVG(amount) AS amount\_avg, MAX(price\*amount) AS max\_total  FROM order\_items \g 'dml\_res/5.txt'  SELECT menu\_item\_id,AVG(price\*amount) AS avg\_total  FROM order\_items  GROUP BY menu\_item\_id  ORDER BY menu\_item\_id \g | cat >> 'dml\_res/5.txt' |

Результат:



*Рис.31. Результат запроса 5\_1*



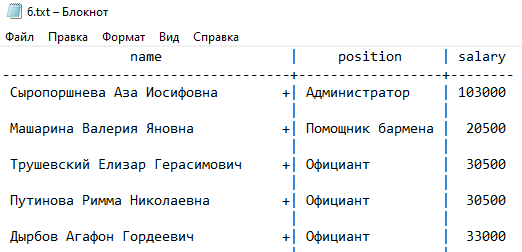
*Рис.32. Результат запроса 5\_2*

#### Сделайте выборку данных из связанных таблиц (не менее двух примеров)

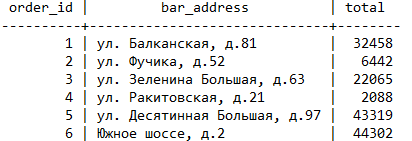
Листинг 20.

|  |
| --- |
| SELECT worker.name, position.name AS position, (position.salary+worker.bonus) AS salary  FROM worker  INNER JOIN position ON position.id = worker.position\_id \g 'dml\_res/6.txt'  SELECT order\_history.id AS order\_id,  bar.address AS bar\_address,  SUM(order\_items.price \* order\_items.amount) AS total  FROM order\_history  LEFT JOIN bar ON bar.id = order\_history.bar\_id  LEFT JOIN order\_items ON order\_items.order\_id = order\_history.id  GROUP BY order\_history.id,bar.address \g | cat >> 'dml\_res/6.txt' |

Результат:



*Рис.33. Результат запроса 6\_1*



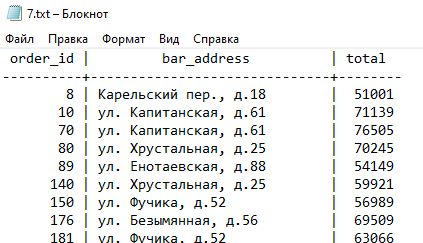
*Рис.34. Результат запроса 6\_2*

#### Создайте запрос, рассчитывающий совокупную характеристику с использованием группировки, наложите ограничение на результат группировки

Листинг 21.

|  |
| --- |
| SELECT order\_history.id AS order\_id,  bar.address AS bar\_address,  SUM(order\_items.price \* order\_items.amount) AS total  FROM order\_history  LEFT JOIN bar ON bar.id = order\_history.bar\_id  LEFT JOIN order\_items ON order\_items.order\_id = order\_history.id  GROUP BY order\_history.id,bar.address  HAVING SUM(order\_items.price \* order\_items.amount)>50000 \g 'dml\_res/7.txt' |

Результат:



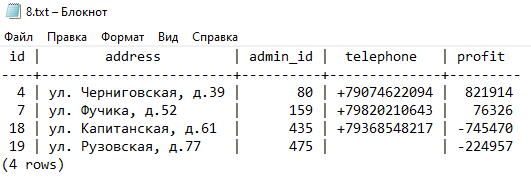
*Рис.35. Результат запроса 7*

#### Придумайте и реализуйте пример использования вложенного запроса

Листинг 22.

|  |
| --- |
| SELECT \* FROM bar  WHERE admin\_id IN  (SELECT id FROM worker WHERE position\_id=1 AND bonus=5000) \g 'dml\_res/8.txt' |

Результат:



*Рис.36. Результат запроса 8*

#### С помощью оператора INSERT добавьте в каждую таблицу по одной записи

Листинг 23.

|  |
| --- |
| INSERT INTO bar (address,telephone,profit) VALUES ('ул. Политехническая, д.15','+79817610521',100000);  INSERT INTO bar\_menu ( bar\_id, menu\_item\_id, price) VALUES ((SELECT COUNT(id) FROM bar), 5, 300);  INSERT INTO menu\_category (name, adult) VALUES ('Безалкогольные коктейли', False);  INSERT INTO menu\_item ( name, description, weight, category\_id, recomended\_price)  VALUES ('Безалкогольный мохито', 'Освежающий безалкогольный коктейль', 0.4,  (SELECT id FROM menu\_category WHERE name= 'Безалкогольные коктейли'), 300);  INSERT INTO order\_history (bar\_id, date\_and\_time, customer\_name)  VALUES ((SELECT COUNT(id) FROM bar), '2020-03-31 15:02:03','Иванов Илья Дмитриевич');  INSERT INTO order\_items (order\_id,menu\_item\_id, price, amount)  VALUES ((SELECT COUNT(id) FROM order\_history), 5, 300, 10);  INSERT INTO position (name, salary, responsibilities, discount)  VALUES ('Охранник', 35000, 'Охрана бара', 0.10);  INSERT INTO promotion (date\_from, date\_to, category\_id, menu\_item\_id, terms, discount)  VALUES ('2020-03-31','2020-04-30',1,NULL,'No Terms',0.25);  INSERT INTO regular\_customer (name, telephone, discount, points, favourite\_bar\_id)  VALUES ('Иванов Илья Дмитриевич', '79817610521', 0.15, 1000, (SELECT COUNT(id) FROM bar));  INSERT INTO worker (name, bar\_id, telephone, bonus, entry\_date, position\_id, address, schedule)  VALUES ('Яковлев Антон Владимирович', (SELECT COUNT(id) FROM bar), '79999999999', 5000,  '2020-03-31', 1, 'ул. Политехническая, д.32, кв.28', 'СР-ВС, 18:00-03:00');  INSERT INTO worker\_history (worker\_id, date\_from, date\_to, bar\_id, position\_id, salary)  VALUES (10,'2018-05-10','2019-03-19',7,3,36000); |

#### С помощью оператора UPDATE измените значения нескольких полей у всех записей, отвечающих заданному условию

Листинг 24.

|  |
| --- |
| UPDATE bar SET admin\_id=5 WHERE admin\_id IS NULL;  UPDATE position SET salary = salary+5000 WHERE salary < 40000; |

#### С помощью оператора DELETE удалите запись, имеющую максимальное (минимальное) значение некоторой совокупной характеристики

Листинг 25.

|  |
| --- |
| DELETE FROM order\_items WHERE price= ( SELECT MAX(price) FROM order\_items); |

#### С помощью оператора DELETE удалите записи в главной таблице, на которые не ссылается подчиненная таблица (используя вложенный запрос)

Листинг 26.

|  |
| --- |
| INSERT INTO bar (address,telephone,profit) VALUES ('ул. Тестовая, д.140','+79817613521',100000);  DELETE FROM bar WHERE id NOT IN ( SELECT bar\_id FROM worker GROUP BY bar\_id); |

### Индивидуальные задания

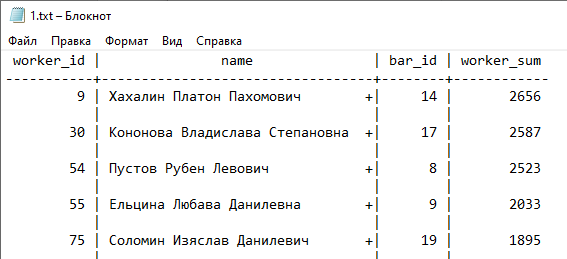
#### Вывести сотрудников, которые работали во всех барах более половины от времени их существования. Время существования считать от минимальной даты назначения до максимальной даты назначения в этом баре.

Алгоритм работы: суммируем время работы каждого сотрудника в каждом баре (возможно на разных позициях) и отбираем тех сотрудников, у которых в каждом баре проработано времени больше половины от того, сколько существует бар (максимальная дата назначения минус минимальная дата назначения в этом баре).

Листинг 27.

|  |
| --- |
| SELECT worker\_id, worker.name, worker\_history.bar\_id,  SUM(worker\_history.date\_to-worker\_history.date\_from) as worker\_sum  FROM worker\_history  LEFT JOIN worker ON worker\_history.worker\_id= worker.id  GROUP BY worker\_history.bar\_id, worker\_id, worker.name  HAVING SUM(worker\_history.date\_to-worker\_history.date\_from) > (SELECT bar\_detla FROM  (SELECT worker\_history.bar\_id as curr\_bar, MAX(date\_from)-MIN(date\_from) as bar\_detla FROM worker\_history  GROUP BY worker\_history.bar\_id) as tmp\_table WHERE tmp\_table.curr\_bar=worker\_history.bar\_id)/2  ORDER BY worker\_id \g 'dml\_ind\_res/1.txt' |

Результат:



*Рис.37. Результат запроса*

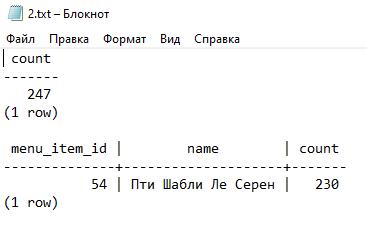
#### Вывести позиции, которые участвуют в большинстве заказов, сумма которых более заданной

Алгоритм работы: выбираем заказы с суммой более заданной. Далее считаем, сколько раз каждая позиция встречается в выбранных заказах и выбираем позиции, которые встречаются в большинстве (больше половины) заказов.

Листинг 28.

|  |
| --- |
| SELECT COUNT(tmp\_table.order\_id) FROM (SELECT order\_id FROM order\_items  GROUP BY order\_id  HAVING SUM(price\*amount)>50000) as tmp\_table  \g 'dml\_ind\_res/2.txt'  SELECT menu\_item\_id, menu\_item.name, COUNT(order\_id) FROM order\_items  LEFT JOIN menu\_item ON order\_items.menu\_item\_id=menu\_item.id  WHERE order\_id IN (SELECT order\_id FROM order\_items GROUP BY order\_id HAVING SUM(price\*amount)>50000)  GROUP BY order\_items.menu\_item\_id, menu\_item.name  HAVING COUNT(order\_id) > (SELECT COUNT(tmp\_table.order\_id) FROM (SELECT order\_id FROM order\_items  GROUP BY order\_id  HAVING SUM(price\*amount)>50000) as tmp\_table)/2  \g | cat >> 'dml\_ind\_res/2.txt' |

Результат:



*Рис.38. Результат запроса*

## Вывод

В ходе выполнения данной лабораторной работы было осуществлено знакомство с языком создания запросов управления данными SQL-DML.

# Лабораторная работа №4

## Цель работы

Знакомство с проблемами, возникающими при высокой нагрузке на базу данных, и методами их решения, путем оптимизации запросов.

## Программа работы

* Написание параметризированных типовых запросов пользователей
* Моделирование нагрузки базы данных
* Снятие показателей работы сервиса и построение соответствующих графиков
* Применение возможных оптимизаций запросов и повторное снятие показателей
* Сравнительный анализ результатов

## Выполнение работы

### Типовые запросы пользователей

Листинг 29.

|  |
| --- |
| query\_1="SELECT AVG(amount) AS amount\_avg, MAX(price\*amount) AS max\_total FROM order\_items;"  query\_2="SELECT menu\_item\_id,AVG(price\*amount) AS avg\_total FROM order\_items GROUP BY menu\_item\_id ORDER BY menu\_item\_id;"  query\_3="SELECT worker.name, position.name AS position, (position.salary+worker.bonus) AS salary FROM worker INNER JOIN position ON position.id = worker.position\_id WHERE worker.name=%(name)s;"  query\_4="INSERT INTO public.order\_history (bar\_id, date\_and\_time, customer\_name) VALUES (%(bar)s,%(date)s,%(name)s);"  query\_5="INSERT INTO public.regular\_customer (name, telephone, discount, points, favourite\_bar\_id) VALUES (%(name)s,%(telephone)s, 0.05,100,%(f\_bar)s);"  query\_6="SELECT id FROM menu\_item WHERE name = %(name)s;"  query\_7="SELECT order\_history.id AS order\_id, bar.address AS bar\_address, SUM(order\_items.price \* order\_items.amount) AS total FROM order\_history LEFT JOIN bar ON bar.id = order\_history.bar\_id LEFT JOIN order\_items ON order\_items.order\_id = order\_history.id GROUP BY order\_history.id,bar.address,order\_items.amount HAVING order\_items.amount>%(amount)s;"  query\_8="SELECT \* FROM bar WHERE admin\_id IN (SELECT id FROM worker WHERE position\_id=1 AND bonus=%(bonus)s);" |

### Моделирование нагрузки БД и снятие показателей

Листинг 30.

|  |
| --- |
| def constant\_threads(num\_threads):  curr\_res\_cosntant\_threads.clear()  for t in range(num\_threads):  dbt = DBThread\_constant\_threads(t)  dbt.start()  while threading.activeCount() > 1:  time.sleep(1)  plot\_x=[k for k in range(401,10000,200)]  plot\_y=[]  for i in range(len(curr\_res\_cosntant\_threads[0])):  curr\_sum=0  for j in range(num\_threads):  curr\_sum+=curr\_res\_cosntant\_threads[j][i]  plot\_y.append(curr\_sum/num\_threads)  plt.plot(plot\_x,plot\_y,linewidth=2.0)  plt.xlabel('Запросов в секунду')  plt.ylabel('Время ответа на один запрос, мс')  if before:  plt.title("Before. Num threads: {}".format(num\_threads))  else:  plt.title("After. Num threads: {}".format(num\_threads))  plt.show()  def dinamic\_threads(num\_querys):  plot\_x=[k for k in range(1,31)]  plot\_y=[]  for num\_threads in range(1,31):  curr\_res\_dinamic\_threads.clear()  for t in range(num\_threads):  dbt = DBThread\_dinamic\_threads(num\_querys)  dbt.start()  while threading.activeCount() > 1:  time.sleep(1)  plot\_y.append(sum(curr\_res\_dinamic\_threads)/len(curr\_res\_dinamic\_threads))  plt.plot(plot\_x,plot\_y,linewidth=2.0)  plt.xlabel('Количество потоков')  plt.ylabel('Время ответа на один запрос, мс')  if before:  plt.title("Before. Dinamic thread num")  else:  plt.title("After. Dinamic thread num")  plt.show()  class DBThread\_constant\_threads(threading.Thread):  def \_\_init\_\_(self,curr\_thread):  self.curr\_thread=curr\_thread  threading.Thread.\_\_init\_\_(self)  self.conn = psycopg2.connect(dbname='bar\_db', user='bar\_user', password='barbar', host='127.0.0.1' )  self.cur = self.conn.cursor()  global before  if not before:  self.cur.execute("PREPARE query3 (int) AS SELECT worker.name, position.name AS position, (position.salary+worker.bonus) AS salary FROM worker INNER JOIN position ON position.id = worker.position\_id WHERE worker.name=$1;")  self.cur.execute("PREPARE query4 (int, date, varchar(50)) AS INSERT INTO public.order\_history (bar\_id, date\_and\_time, customer\_name) VALUES ($1,$2,$3);")  self.cur.execute("PREPARE query5 (varchar(50), varchar(15), int) AS INSERT INTO public.regular\_customer (name, telephone, discount, points, favourite\_bar\_id) VALUES ($1,$2,0.05,100,$3);")  self.cur.execute("PREPARE query8 (varchar(50)) AS SELECT id FROM menu\_item WHERE name = $1;")  self.cur.execute("PREPARE query7 (int) SELECT order\_history.id AS order\_id, bar.address AS bar\_address, SUM(order\_items.price \* order\_items.amount) AS total FROM order\_history LEFT JOIN bar ON bar.id = order\_history.bar\_id LEFT JOIN order\_items ON order\_items.order\_id = order\_history.id GROUP BY order\_history.id,bar.address,order\_items.amount HAVING order\_items.amount>$1")  self.cur.execute("PREPARE query8 (int) AS SELECT \* FROM bar WHERE admin\_id IN (SELECT id FROM worker WHERE position\_id=1 AND bonus=$1);")  def run(self):  for i in range(401,5000,200):  results=[]  for j in range(0,i):  print(j)  random\_query=random.choice(query\_list)  if before:  results.append(exec\_query\_before(random\_query,self.cur))  else:  results.append(exec\_query\_after(random\_query,self.cur))  if len(curr\_res\_cosntant\_threads)<self.curr\_thread+1:  curr\_res\_cosntant\_threads.append([])  curr\_res\_cosntant\_threads[self.curr\_thread].append(sum(results)/len(results))  self.conn.commit()  class DBThread\_dinamic\_threads(threading.Thread):  def \_\_init\_\_(self,num\_querys):  self.num\_querys=num\_querys  threading.Thread.\_\_init\_\_(self)  self.conn = psycopg2.connect(dbname='bar\_db', user='bar\_user', password='barbar', host='127.0.0.1' )  self.cur = self.conn.cursor()  global before  if not before:  self.cur.execute("PREPARE query3 (int) AS SELECT worker.name, position.name AS position, (position.salary+worker.bonus) AS salary FROM worker INNER JOIN position ON position.id = worker.position\_id WHERE worker.name=$1;")  self.cur.execute("PREPARE query4 (int, date, varchar(50)) AS INSERT INTO public.order\_history (bar\_id, date\_and\_time, customer\_name) VALUES ($1,$2,$3);")  self.cur.execute("PREPARE query5 (varchar(50), varchar(15), int) AS INSERT INTO public.regular\_customer (name, telephone, discount, points, favourite\_bar\_id) VALUES ($1,$2,0.05,100,$3);")  self.cur.execute("PREPARE query8 (varchar(50)) AS SELECT id FROM menu\_item WHERE name = $1;")  self.cur.execute("PREPARE query7 (int) SELECT order\_history.id AS order\_id, bar.address AS bar\_address, SUM(order\_items.price \* order\_items.amount) AS total FROM order\_history LEFT JOIN bar ON bar.id = order\_history.bar\_id LEFT JOIN order\_items ON order\_items.order\_id = order\_history.id GROUP BY order\_history.id,bar.address,order\_items.amount HAVING order\_items.amount>$1")  self.cur.execute("PREPARE query8 (int) AS SELECT \* FROM bar WHERE admin\_id IN (SELECT id FROM worker WHERE position\_id=1 AND bonus=$1);")  def run(self):  results=[]  for j in range(0,self.num\_querys+1):  print(j)  random\_query=random.choice(query\_list)  if before:  results.append(exec\_query\_before(random\_query,self.cur))  else:  results.append(exec\_query\_after(random\_query,self.cur))  curr\_res\_dinamic\_threads.append(sum(results)/len(results))  self.conn.commit() |

Функция constant\_threads и класс DBThread\_constant\_threads работают с постоянным количеством потоков. В constant\_threads создаётся нужное число потоков, которые затем создают переменное количество (в моём случае от 401 до 10000 с шагом 200) случайных пользовательских запросов к БД.

Функция dinamic\_threads и класс DBThread\_dinamic\_threads работают с изменяющимся количеством потоков, но с постоянным количеством запросов (в моём случае 2000).

В функциях constant\_threads и dinamic\_threads строятся графики для отображения результатов. В первом случае на них показывается зависимость времени, в среднем потраченного на 1 запрос относительно количества запросов. Во втором случае – относительно числа потоков.

### Оптимизация

Для оптимизации запросов были выбраны 2 метода: создание индексов и использование PREPARE.

Индексы добавляются разово в функции *optimize*, в то время как PREPARE нужно вызывать отдельно для каждого клиента, поэтому он реализуется в классе потока.

Листинг 31.

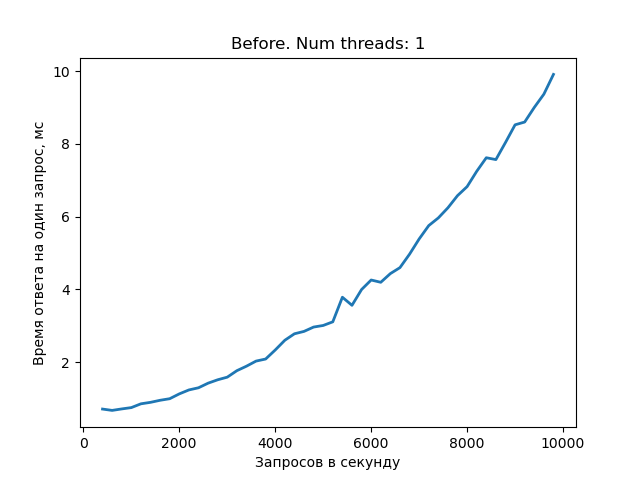
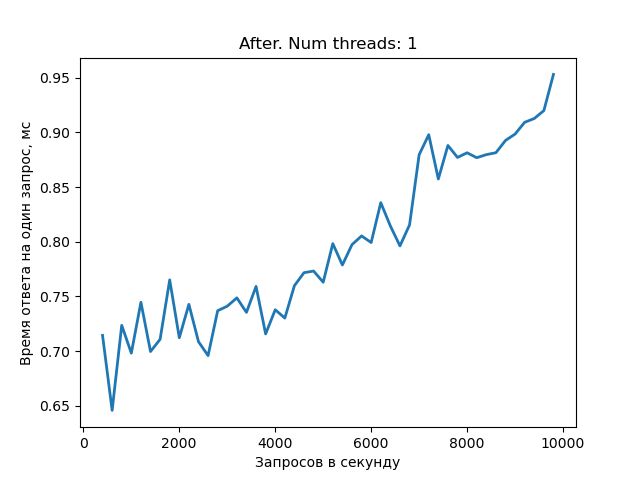
|  |
| --- |
| def optimize():  cursor.execute("CREATE INDEX ON worker(name);")  cursor.execute("CREATE INDEX ON menu\_item(name);")  cursor.execute("CREATE INDEX ON worker(bonus);")  cursor.execute("CREATE INDEX ON order\_items(amount);")  global before  before = False before = False |

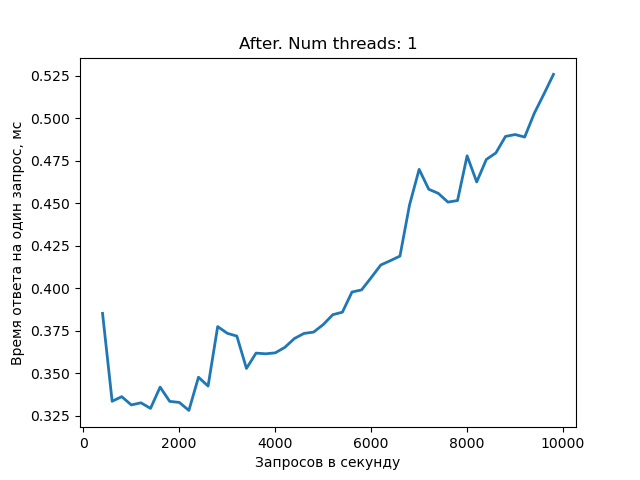
Листинг 32.

|  |
| --- |
| if not before:  self.cur.execute("PREPARE query3 (int) AS SELECT worker.name, position.name AS position, (position.salary+worker.bonus) AS salary FROM worker INNER JOIN position ON position.id = worker.position\_id WHERE worker.name=$1;")  self.cur.execute("PREPARE query4 (int, date, varchar(50)) AS INSERT INTO public.order\_history (bar\_id, date\_and\_time, customer\_name) VALUES ($1,$2,$3);")  self.cur.execute("PREPARE query5 (varchar(50), varchar(15), int) AS INSERT INTO public.regular\_customer (name, telephone, discount, points, favourite\_bar\_id) VALUES ($1,$2,0.05,100,$3);")  self.cur.execute("PREPARE query8 (varchar(50)) AS SELECT id FROM menu\_item WHERE name = $1;")  self.cur.execute("PREPARE query7 (int) SELECT order\_history.id AS order\_id, bar.address AS bar\_address, SUM(order\_items.price \* order\_items.amount) AS total FROM order\_history LEFT JOIN bar ON bar.id = order\_history.bar\_id LEFT JOIN order\_items ON order\_items.order\_id = order\_history.id GROUP BY order\_history.id,bar.address,order\_items.amount HAVING order\_items.amount>$1")  self.cur.execute("PREPARE query8 (int) AS SELECT \* FROM bar WHERE admin\_id IN (SELECT id FROM worker WHERE position\_id=1 AND bonus=$1);") |

### Сравнительный анализ результатов

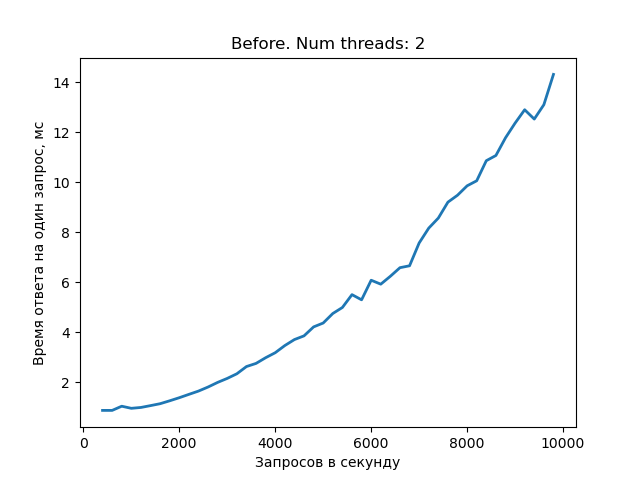
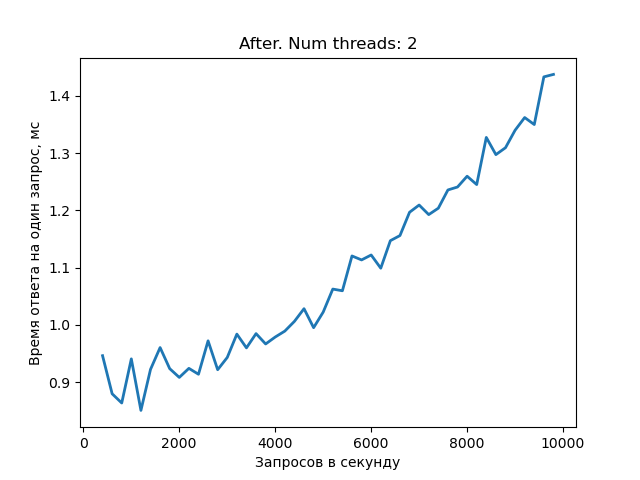
#### Анализ графических результатов

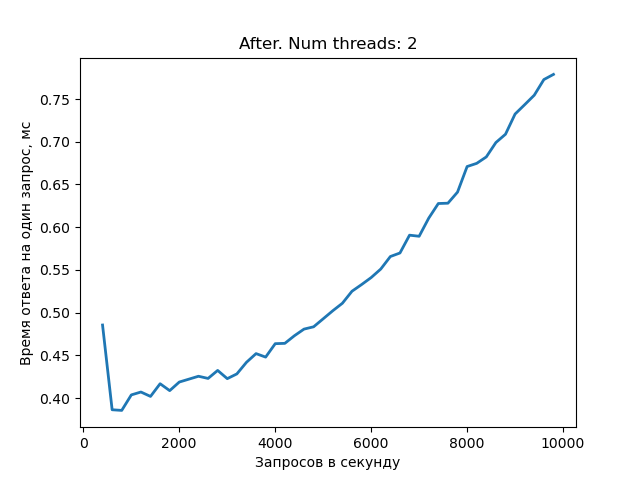
 



*Рис.39. График зависимости времени выполнения запроса от числа запросов при одном потоке: 1 – до оптимизации, 2 – после создания индексов, 3 – после создания индексов и PREPARE*

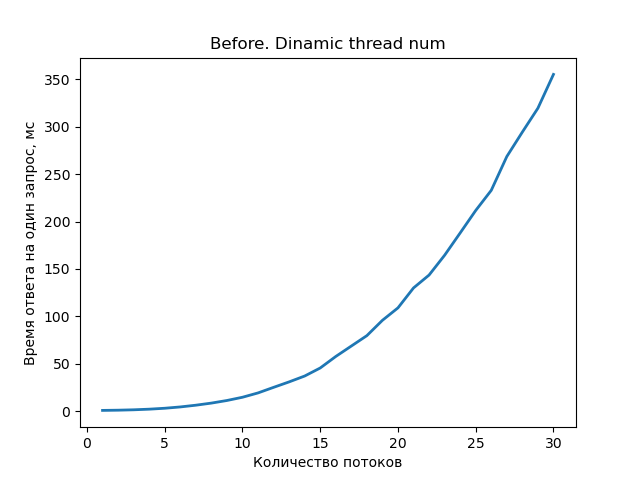
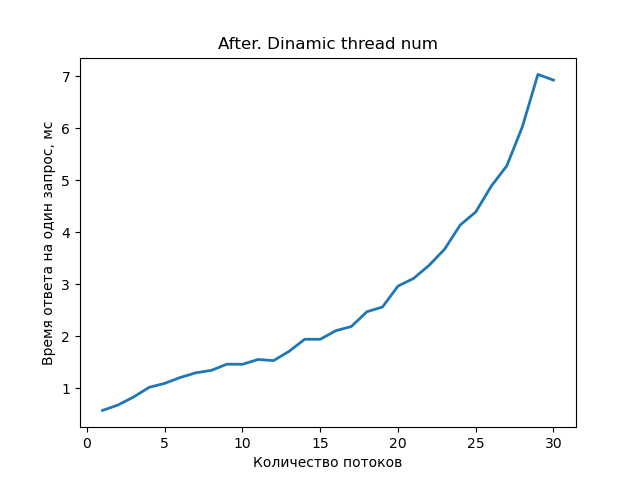
Видно, что уже при добавлении индексов запросы стали выполняться в разы быстрее (в 10 раз при 10000 запросов). После добавления PREPARE запросы стали выполняться ещё примерно в 2 раза быстрее.

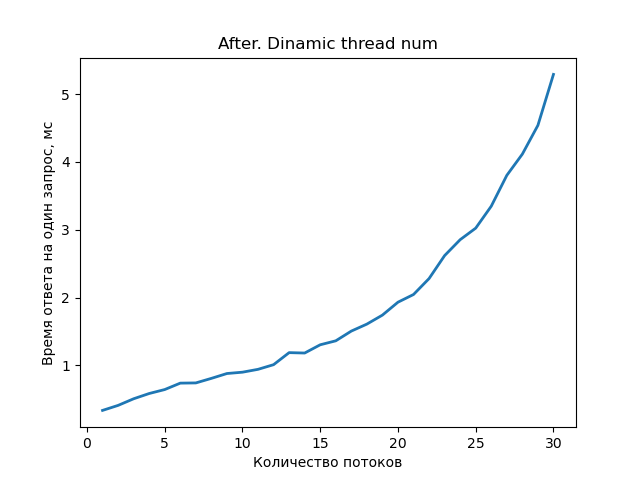
 



*Рис.40.* *График зависимости времени выполнения запроса от числа запросов при двух потоках: 1 – до оптимизации, 2 – после создания индексов, 3 – после создания индексов и PREPARE*

При двух активных потоках видна аналогичная ситуация. Уже после добавления индексов запросы стали выполняться в разы быстрее (также в 10 раз при 10000 запросов), а после добавления PRERARE скорость выполнения запросов также увеличилась примерно вдвое.



*Рис.41.* *График зависимости времени выполнения запроса от числа потоков: 1 – до оптимизации, 2 – после создания индексов, 3 – после создания индексов и PREPARE*

При динамическом числе потоков результаты оптимизации видны ещё сильнее. После создания индексов и добавления PREPARE удаётся уменьшить время запроса на 2 порядка (с 350 мс до 5 мс при 30 потоках).

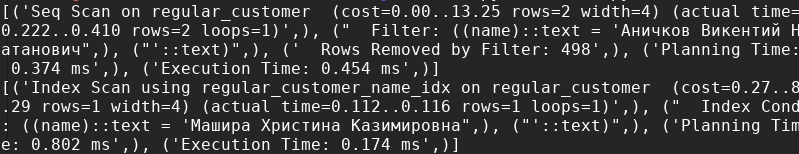
Можно сделать вывод о том, что оба рассмотренных вида оптимизации работают. Наилучший результат можно получить при использовании обоих видов одновременно, но создание индексов даёт более значительный результат.

#### Анализ результатов explain

Запрос:

EXPLAIN ANALYZE SELECT points FROM regular\_customer WHERE name = %(name)s;

Результат:



*Рис.42. Вывод explain analyze*

В данном примере планировщик до оптимизации решил использовать последовательное сканирование. Это означает, что условие будет проверяться для каждого просканированного узла и выводиться будут те строки, которые удовлетворяют ему.

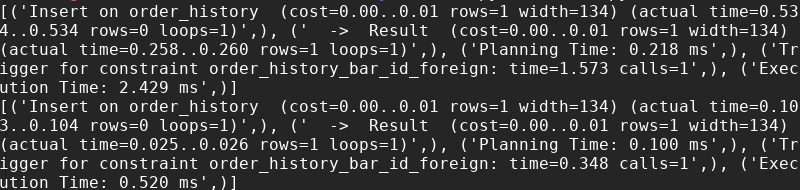
Строка «Rows Removed» выводится, когда условие фильтра отбрасывает минимум одну просканированную строку или потенциальную пару соединения, если это узел соединения.

После оптимизации планировщик выбирает индексное сканирование, что приводит к снижению времени выполнения запроса, так как мы выбираем очень узкую область таблицы и выборка данных с использование индекса в таком случае работает значительно быстрее.

Запрос:

EXPLAIN ANALYZE INSERT INTO public.order\_history (bar\_id, date\_and\_time, customer\_name) VALUES (%(bar)s,%(date)s,%(name)s);

Результат:



*Рис.43. Вывод explain analyze*

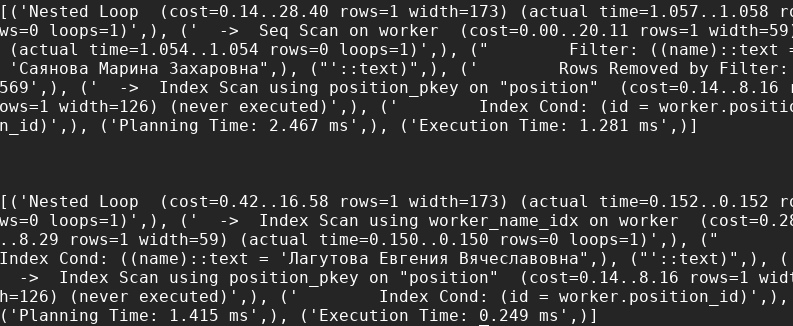
Когда выполняется запрос INSERT, собственно изменение данных в таблице происходит в узле верхнего уровня Insert. Узлы плана более низких уровней выполняют работу по нахождению старых строк или вычислению новых данных. Узел, изменяющий данные, может выполняться значительное время, но планировщик не учитывает эту работу в оценке общей стоимости. Это связано с тем, что эта работа будет одинаковой при любом правильном плане запроса, и поэтому на выбор плана она не влияет.

#### Анализ результатов explain с более сложными запросами

Запрос:

EXPLAIN ANALYZE SELECT worker.name, position.name AS position, (position.salary+worker.bonus) AS salary FROM worker INNER JOIN position ON position.id = worker.position\_id WHERE worker.name=%(name)s;

Результат:



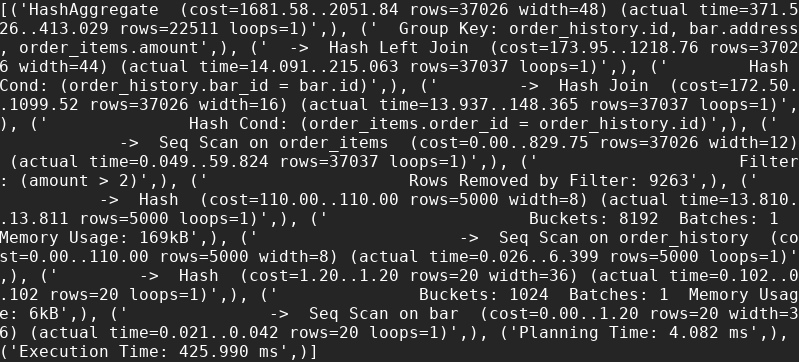
*Рис.44. Вывод explain analyze*

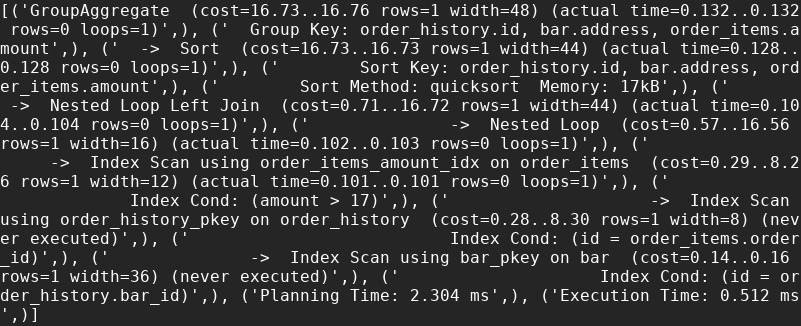
В данном случае появляется узел соединения с вложенным циклом, на вход которому поступают данные от его потомков, узлов сканирования. Первым узлом сканирования является последовательное сканирование, вторым узлом является индексное сканирование, так как поиск ведётся по id, для которого создан индекс. После оптимизации оба потомка используют индексное сканирование. Время выполнения запроса улучшилось.

Запрос:

SELECT order\_history.id AS order\_id, bar.address AS bar\_address, SUM(order\_items.price \* order\_items.amount) AS total FROM order\_history LEFT JOIN bar ON bar.id = order\_history.bar\_id LEFT JOIN order\_items ON order\_items.order\_id = order\_history.id GROUP BY order\_history.id,bar.address,order\_items.amount HAVING order\_items.amount>%(amount)s;

Результат:





*Рис.45. Вывод explain analyze*

В данном случае до оптимизации используется HashAggregate и Hash Join-ы, которым поступают данные от большого числа узлов с последовательным сканированием. После оптимизации используются GroupAggregate, сортировка и узлы с индексным сканированием. Время выполнения запроса значительно улучшилось.

## Вывод

В ходе выполнения данной лабораторной работы было осуществлено знакомство с проблемами, возникающими при высокой нагрузке на базу данных и методами их решения, путем оптимизации запросов. Лучшим видом оптимизации из рассмотренных оказалось создание индексов. Примеры до и после создания индексов зачастую различны на порядок, в то время как c использованием PREPARE разница менее заметна. Но, вероятно, дело в выбранных типовых запросах, которых в данном случае хорошо оптимизировались созданием индексов. В некоторых случаях создание индексов может не помочь, а, наоборот, ухудшить время выполнения, так что наилучший способ оптимизации, конечно, всегда зависит от конкретной сложившейся ситуации.