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 _____ П.А. Меньшов
(подпись)

Преподаватель _____А.В. Мяснов
(подпись)

“___”_____ 2021 г.

Санкт-Петербург
2021

Лабораторная работа №1. Проектирование БД

Разработка структуры БД

Цели работы

Познакомиться с основами проектирования схемы БД, способами организации данных в SQL-БД.

Программа работы

- Создание проекта для работы в GitLab.
- Выбор задания (предметной области), описание набора данных и требований к хранимым данным в свободном формате в wiki своего проекта в GitLab.
- Формирование в свободном формате (предпочтительно в виде графической схемы) схемы БД, соответствующей заданию. Должно получиться не менее 7 таблиц.
- Согласование с преподавателем схемы БД. Обоснование принятых решений и соответствия требованиям выбранного задания.
- Выкладывание схемы БД в свой проект в GitLab.
- Демонстрация результатов преподавателю.

- Самостоятельное изучение SQL-DDL.
- Создание скрипта БД в соответствии с согласованной схемой. Должны присутствовать первичные и внешние ключи, ограничения на диапазоны значений. Демонстрация скрипта преподавателю.
- Создание скрипта, заполняющего все таблицы БД данными.
- Выполнение SQL-запросов, изменяющих схему созданной БД по заданию преподавателя. Демонстрация их работы преподавателю.

Выполнение работы

В качестве предмета проекта была выбрана база данных сети магазинов электронных сигарет ("вейп-шопов"). Кроме истории заказов и всей необходимой информации о товарах, клиентах и магазинах, также хранится история поставок на склады от производителей.

БД содержит в себе таблицы: **shop, buyer, storage, manufacturer, order, product, product_type, product_to_product, ordered_product, batch, product_in_batch.**

1)**shop** - таблица, в которой хранятся данные о магазинах нашей сети (адрес, название).

```
create table if not exists shop
(
        shop_id serial not null
                constraint shop_pk
                        primary key,
        shop_address varchar default ''::character varying not null,
        shop_name varchar default ''::character varying not null
);
```

2)**buyer** - основная информация о клиенте сети (имя, телефон, e-mail).

```
create table if not exists buyer
(
        buyer_id serial not null
                constraint buyer_pk
                        primary key,
        buyer_name varchar default ''::character varying not null,
        phone varchar(20) not null,
        email varchar default ''::character varying not null
);
```

3)**storage** - таблица с информацией о складах, которые относятся к нашим магазинам (id магазина).

```
create table if not exists storage
(
        storage_id serial not null
```

```
                constraint storage_pk
                        primary key,
        shop_id integer not null
                constraint storage_shop_shop_id_fk
                        references shop
                                on delete cascade
);
```

4)**manufacturer** - информация о производителе, который поставляет партии товаров на склады (имя компании-производителя).

```
create table if not exists manufacturer
(
        manufacturer_id serial not null
                constraint manufacturer_pk
                        primary key,
        manufacturer_name varchar default ''::character varying not null
);
```

5)**order** - данные о заказе клиента (id клиента, id магазина, дата создания и дата выдачи заказа).

```
create table if not exists "order"
(
        order_id serial not null
                constraint order_pk
                        primary key,
        buyer_id integer not null
                constraint order_buyer_buyer_id_fk
                        references buyer,
        shop_id integer
                constraint order_shop_shop_id_fk
                        references shop,
        order_date_to date not null,
        order_date_from date not null
);
```

6)**product**- основная информация о продукте (название, цена, id типа продукта, описание).

```
create table if not exists product
(
        product_id serial not null
                constraint device_pk
                        primary key,
        product_name varchar default ''::character varying not null,
        price integer not null,
        product_type_id integer not null
                constraint product_product_type_product_type_id_fk
                        references product_type
                                on delete cascade,
```

```
        description varchar default ''::character varying not null
);
```

7)**product_type** - к примеру, жидкость, устройство, комплектующая - содержит информацию о типе (описание, название).

```
create table if not exists product_type
(
        product_type_id serial not null
                constraint product_type_pk
                        primary key,
        description varchar default ''::character varying not null,
        product_type_name varchar default ''::character varying not null
);
```

8)**product_to_product** - таблица с информацией о том, какие комплектующие подходят к каким устройствам (id продукта, id комплектующей, описание).

```
create table if not exists product_to_product
(
        product_id integer not null
                constraint product_to_product_product_product_id_fk
                        references product
                                on delete cascade,
        product_part_id integer not null
                constraint product_to_product_product_product_id_fk_2
                        references product
                                on delete cascade,
        description varchar default ''::character varying not null
);
```

9)**ordered_product** - таблица с информацией о заказанном продукте, которая служит для сопоставления продукта и заказа (id заказа, количество, id продукта).

```
create table if not exists ordered_product
(
        order_id integer not null
                constraint product_in_order_order_order_id_fk
                        references "order"
                                on delete cascade,
        quantity integer not null,
        product_id integer not null
                constraint ordered_product_product_product_id_fk
                        references product
                                on delete cascade
);
```

10)**batch** (партия) - информация о партиях продуктов, поставляемых производителем на склады (дата создания, дата доставки, информация, id производителя, id склада).

```
create table if not exists batch
(
        batch_id serial not null
                constraint batch_pk
                        primary key,
        batch_date_to date not null,
        batch_date_from date not null,
        batch_info varchar,
        manufacturer_id integer not null
                constraint batch_manufacturer_manufacturer_id_fk
                        references manufacturer,
        storage_id integer not null
                constraint batch_storage_storage_id_fk
                        references storage
                                on delete cascade
);
```

11)**product_in_batch** - таблица с информацией о поставляемом в партии продукте, служащая для сопоставления продукта и партии, в которой он поступает на склад. (id партии, описание, дата изготовления, количество, id продукта).

```
create table if not exists product_in_batch
(
        batch_id integer not null
                constraint product_in_batch_batch_batch_id_fk
                        references batch
                                on delete cascade,
        description varchar default ''::character varying not null,
        create_date date not null,
        quantity integer not null,
        product_id integer not null
                constraint product_in_batch_product_product_id_fk
                        references product
);
```

Также немаловажно отметить, что в таблицах **ordered_product**, **product_in_batch**, **product_to_product** нет собственных первичных ключей, поскольку эти таблицы, в силу нормализации данных, нужны лишь для хранения зависимостей между записями из других таблиц.

Скрипт заполнения БД данными:

```
INSERT INTO public.manufacturer (manufacturer_name)
                        VALUES
                        ('manufacturer 1'),
                        ('manufacturer 2'),
                        ('manufacturer 3');

INSERT INTO public.shop (shop_address, shop_name)
                        VALUES
                        ('shop_address 1', 'shop_name 1'),
                        ('shop_address 2', 'shop_name 2'),
                        ('shop_address 3', 'shop_name 3');

INSERT INTO public.buyer (buyer_name, phone, email)
                        VALUES
                        ('Poo Ting', '11-00-11', 'pt@government.ru'),
                        ('V. Milonov', '555', 'milonov@brain.net'),
                        ('Lebowski', '88005553535','thedude@yahoo.com');

INSERT INTO public.order (buyer_id, shop_id, order_date_to, order_date_from)
                        VALUES
                        (1, 1, '2019-01-01', '2019-01-30'),
                        (2, 3, '2020-02-01', '2020-09-01'),
                        (3, 2, '1966-10-31', '1966-11-01');


INSERT INTO public.product_type (description, product_type_name)
            VALUES
            ('description_1', 'device'),
            ('description_2', 'component_part'),
            ('description_3', 'liquid');

INSERT INTO public.product (product_name, price, product_type_id,
description)
            VALUES
            ('device', 4000, 1, 'description_1'),
            ('component_part', 500, 2, 'description_2'),
            ('liquid', 1000, 3, 'description_3');

INSERT INTO public.product_to_product(product_id, product_part_id,
description)
            VALUES
            (1, 2, 'description_1');

INSERT INTO public.ordered_product (order_id, quantity, product_id)
                        VALUES
                        (1, 1, 1),
                        (1, 2, 2),
                        (2, 1, 3);


INSERT INTO public.storage (shop_id)
                        VALUES
                        (1),
                        (2),
                        (3);

INSERT INTO public.batch (batch_date_to, batch_date_from, batch_info,
manufacturer_id, storage_id)
                        VALUES
                        ('1944-06-06', '1944-06-05', 'D Day', 1, 1),
                        ('1099-12-31', '1096-01-01', '1st crusade', 2, 2),
```

```
                        ('1969-06-20', '1961-06-16', 'Apollo 11', 3, 3);

INSERT INTO public.product_in_batch (batch_id, description, create_date,
quantity, product_id)
                        VALUES
                        (1, 'batch 1 device', '1337-08-02', 8800, 1),
                        (2, 'batch 2 liquid', '2001-08-02', 555, 1),
                        (2, 'batch 2 device', '2020-09-01', 3535, 3);
```

Индивидуальное задание:

1. Добавить возможность покупателям оставлять отзывы и оценки товарам

2. Добавить работников магазинов. Важно учесть, что:

- работник может представать в роли покупателя и совершать покупки в магазинах, а данные о человеке не хотелось бы дублировать
- работник занимается оформлением заказов и поставок
- работник нанимается по договору на какой-то период времени

Изменена структура БД в соответствии с заданием:

- Добавлена таблица review, каждая запись которой хранит id покупателя и id продукта, и кроме этого содержит отзыв с оценкой.
- Добавлена таблица employee, каждый кортеж которой содержит buyer_id и даты начала и конца договора сотрудника.

Логика FK buyer_id следующая: при приеме сотрудника на работу его данные автоматически добавляются в таблицу с покупателями, и для применения скидки сотрудника при покупке им в нашей сети магазинов, мы проводим поиск по buyer_id в таблице employee. Если что-то нашлось, то скидка применится. Добавление сотрудника в таблицу buyer со старта сделано для того, чтобы избежать дублирования основных данных о человеке.

Скрипт изменения БД в связи с индивидуальным заданием:

```
create table if not exists review
(
        review_id serial not null
                constraint review_pk
                        primary key,
        buyer_id serial not null
                constraint review_buyer_buyer_id_fk
                        references buyer,
        product_id serial not null
                constraint review_product_product_id_fk
```

```
                        references product
                                on delete cascade,
        rating integer not null,
        text varchar(200)
);

create unique index if not exists review_review_id_uindex
        on review (review_id);

create table if not exists employee
(
        employee_id serial not null
                constraint employee_pk
                        primary key,
        date_from date not null,
        date_to date not null,
        buyer_id integer not null
                constraint employee_buyer_buyer_id_fk
                        references buyer
                                on delete cascade
);

alter table batch add column if not exists employee_id integer not null
                        constraint batch_employee_employee_id_fk
                                references employee
                                        on delete cascade;

alter table "order" add column if not exists employee_id integer not null
                        constraint order_employee_employee_id_fk
                                references employee
                                        on delete cascade;


create unique index if not exists employee_buyer_id_uindex
        on employee (buyer_id);

create unique index if not exists employee_employee_id_uindex
        on employee (employee_id);
```

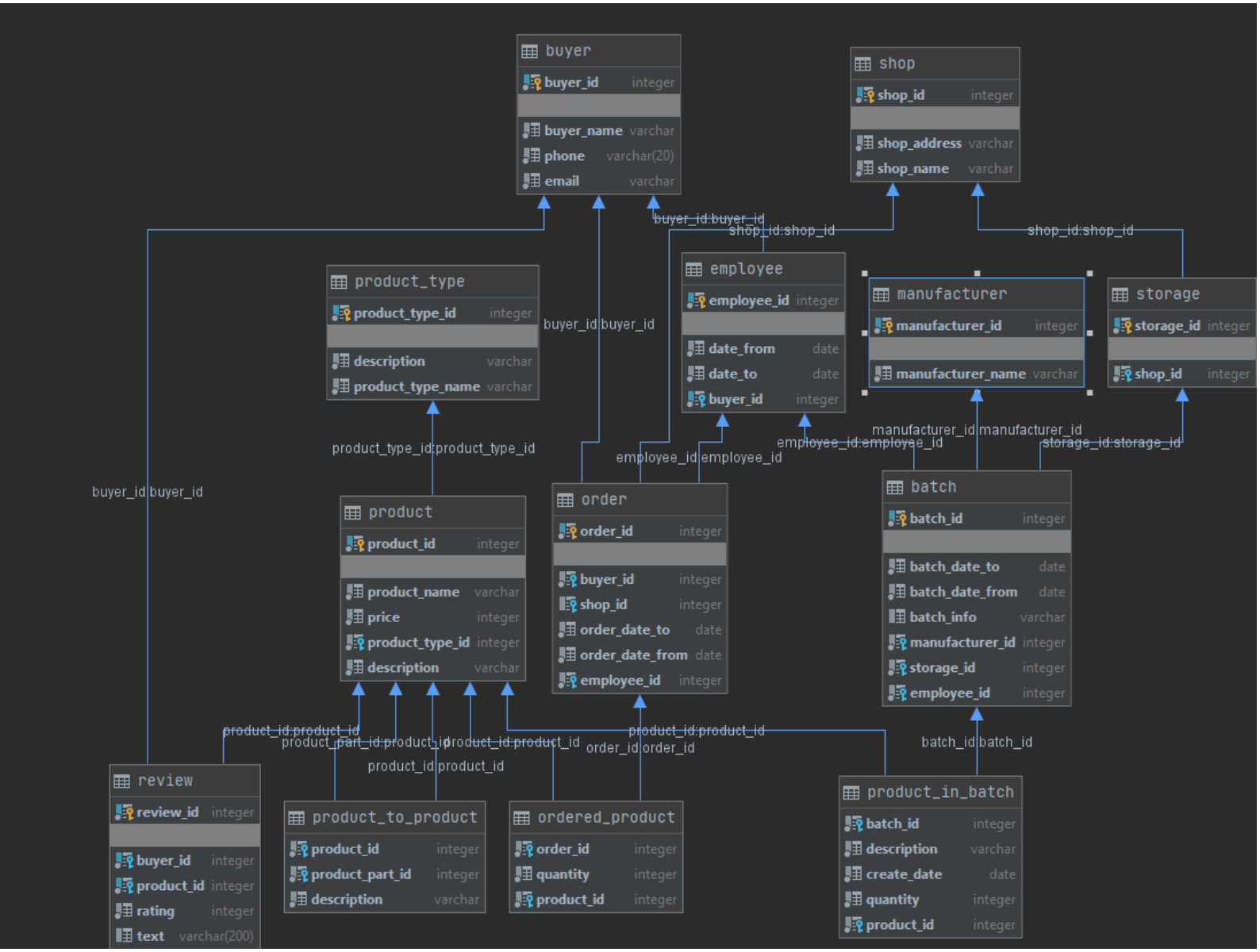


Рис 1. Итоговая схема БД.

Лабораторная работа №2. Генерация тестовых данных

Цели работы

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

Программа работы

1. Реализация в виде программы параметризуемого генератора, который позволит сформировать набор связанных данных в каждой таблице.
2. 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

Был выбран язык python и драйвер psycopg2, тип генерации – полносью случайный. Было написано консольное приложение, где в качестве аргументов можно задать количество записей, вставляемых в какую-либо таблицу. Значения по умолчанию – 0. Приложение может работать как с пустой базой данных, так и с уже заполненной, используя имеющиеся данные. Заполнять можно как по одной таблице, так и несколько или все сразу.

Кроме того, существует отдельный параметр ***--truncate*** .

Когда он равен 1, ко всем таблицам в базе данных применяется truncate table и restart identity.

Все ограничения базы данных соблюдены при генерации. Полный код приложения расположен по ссылке https://gitlab.icc.spbstu.ru/maynekeen/vape-shop_PostgreSQL/-/blob/master/lab2/main.py

Приложение использует модуль config.py для чтения параметров соединения с базой данных. Для чтения данных о соединении используется файл config.ini.

Пример файла конфигурации:

```
[postgres]
dbname = vape_shop
user = postgres
password = password
```

Создание соединения:

```
config = configparser.ConfigParser()
config.read('config.ini', encoding='utf-8')

connection = psycopg2.connect(
    dbname=config.get("postgres", "dbname"),
    user=config.get("postgres", "user"),
    password=config.get("postgres", "password"))

cursor = connection.cursor()
```

Пример метода для генерации продуктов:

```
def generate_products(amount):
    global products
    temp = []
    for i in range(int(amount)):
        temp.append(tuple((random_string(20), random.randint(50, 8000),
random.choice(productTypes)[0],
                           random_string(100),)))
    products_rows = tuple(temp)
    query = "INSERT INTO product (product_name, price, product_type_id,
description) VALUES (%s, %s, %s, %s)"
    cursor.executemany(query, products_rows)
    cursor.execute('SELECT product_id FROM product')
    products = cursor.fetchall()
```

Общая суть работы приложения такова: генерируются абсолютно случайные данные, подходящие под заданный в базе формат. Далее из этих случайных данных формируются кортежи, которые вставляются в параметризированные INSERT-запросы, которые затем исполняются.

Лабораторная работа №3. Язык SQL DML

Цели работы

Познакомиться с языком создания запросов управления данными SQL-DML.

Программа работы

- Изучение SQL-DML.
- Выполнение всех запросов из списка стандартных запросов. Демонстрация результатов преподавателю.
- Получение у преподавателя и реализация SQL-запросов в соответствии с индивидуальным заданием. Демонстрация результатов преподавателю.
- Сохранение в БД выполненных запросов SELECT в виде представлений, запросов INSERT, UPDATE или DELETE -- в виде ХП. Выкладывание скрипта в GitLab.

Выполнение работы

Скрипты с запросами можно найти по ссылке
https://gitlab.icc.spbstu.ru/maynekeen/vape-shop_PostgreSQL/-/blob/master/Report_lab3_VapeShop.md

Стандартные запросы

Сделайте выборку всех данных из каждой таблицы.

Запросы:

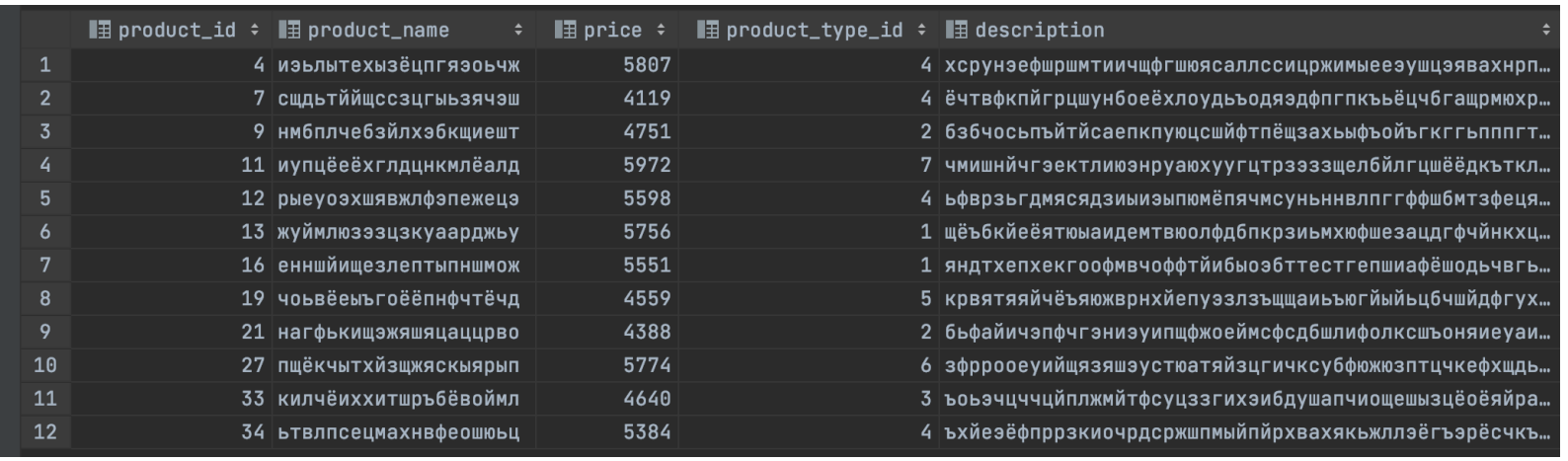
```
create view task1_1 as SELECT * FROM batch;
create view task1_2 as SELECT * FROM buyer;
create view task1_3 as SELECT * FROM employee;
create view task1_4 as SELECT * FROM manufacturer;
create view task1_5 as SELECT * FROM "order";
create view task1_6 as SELECT * FROM ordered_product;
create view task1_7 as SELECT * FROM product;
create view task1_8 as SELECT * FROM product_in_batch;
create view task1_9 as SELECT * FROM product_to_product;
create view task1_10 as SELECT * FROM product_type;
create view task1_11 as SELECT * FROM review;
create view task1_12 as SELECT * FROM shop;
create view task1_13 as SELECT * FROM storage;
```

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

Запросы:

```
-- Вывод продуктов, цена на которые лежит в диапазоне от 3000 до 6000.

create view task2_1 as SELECT * FROM product WHERE price BETWEEN 3000 AND 6000;
```

Результат:

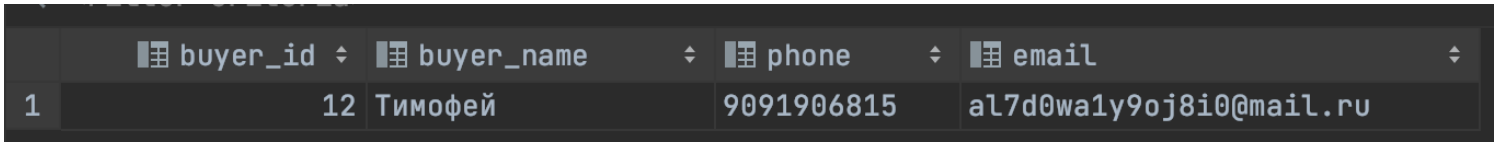


	product_id	product_name	price	product_type_id	description
1	4	иэьлытехызёцпгяэоьчж	5807	4	хсрунэефшршмтиичщфгшюясаллссицржимыееэушцэявахнрп…
2	7	сщдьтййщссзцгыьзячэш	4119	4	ёчтвфкпйгрцшунбоеёхлоудьъодяэдфпгпкъьёцчбгащрмюхр…
3	9	нмбплчебзйлхэбкщиешт	4751	2	бзбчосьпъйтйсаепкпуюцсшйфтпёщзахыыфъойъгкггьпппгт…
4	11	иупцёеёхглдцнкмлёалд	5972	7	чмишнйчгэектлиюэнруаюхуугцтрзэззщелбйлгцшёёдкътклл…
5	12	рыеуоэхшявжлфэпежецэ	5598	4	ьфврзьгдмясядзиыиэыпюмёпячмсуньннвлпггффшбмтзфеця…
6	13	жуймлюзэзцзкуаарджьу	5756	1	щёъбкйеёятюыаидемтвюолфдбпкрзиьмхюфшезацдгфчйнкхцл…
7	16	енншйищезлептыпншмож	5551	1	яндтхепхекгоофмвчоффтйибыоэбттестгепшиафёшодьчвгьл…
8	19	чоьвёеыъгоёёпнфчтёчд	4559	5	крвятяяйчёъяюжврнхйепуэзлзъщщаиьъюгйыйьцбчшйдфгухл…
9	21	нагфькищэжяшяцаццрво	4388	2	бьфайичэпфчгэниэуипщфжоеймсфсдбшлифолксшъоняиеуаил…
10	27	пщёкчытхйзщжяскыярып	5774	6	зфррооеуийщязяшэустюатяйзцгичксубфюжюзптцчкефхщдьл…
11	33	килчёиххитшръбёвоймл	4640	3	ъоьэчцччцйплжмйтфсуцззгихэибдушапчиощешызцёоёяйраз…
12	34	ьтвлпсецмахнвфеошюьц	5384	4	ъхйеэёфпррзкиочрдсржшпмыйпйрхвахякьжллэёгъэрёсчкъл…

Рис 2. Результат запроса

```
--Вывод покупателей в имени которых содержится паттерн 'имо'.

create view task2_2 as SELECT * FROM buyer WHERE buyer.buyer_name LIKE '%имо%';
```

Результат:



	buyer_id	buyer_name	phone	email
1	12	Тимофей	9091906815	al7d0wa1y9oj8i0@mail.ru

Рис 3. Результат запроса

```
--Вывод сотрудников, даты date_from которых '1939-07-05' и '1926-04-07'.

create view task2_3 as SELECT * FROM employee WHERE date_from IN ('1939-07-05', '1926-04-07');
```

Результат:

	employee_id	date_from	date_to	buyer_id
1	4	1939-07-05	1970-12-29	21
2	6	1926-04-07	1934-11-21	30

Рис 4. Результат запроса

Создайте в запросе вычисляемое поле.

--Рассчитываем количество суток, прошедших между date_to и date_from всех сотрудников.

```
sql create view task3 as SELECT date_from, date_to, employee_id,
abs(extract(epoch FROM date_to::timestamp - date_from::timestamp)/3600/24)
AS calc_place FROM employee;
```

Результат:

	date_from	date_to	employee_id	calc_place
1	1960-09-18	1950-01-13	1	3901
2	1976-05-18	1923-03-26	2	19412
3	1971-01-30	1984-01-13	3	4731
4	1939-07-05	1970-12-29	4	11500
5	1994-05-03	2003-03-01	5	3224
6	1926-04-07	1934-11-21	6	3150
7	1932-10-08	1932-10-17	7	9
8	1976-05-14	1961-05-10	8	5483
9	1954-03-26	1928-07-28	9	9372
10	1960-06-04	1935-05-28	10	9139

Рис 5. Результат запроса

Сделайте выборку всех данных с сортировкой по нескольким полям.

--Сортировка данных в таблице product_in_batch по количеству и описанию.

```
create view task4 as SELECT * FROM product_in_batch ORDER BY quantity,
description;
```

Результат:

	batch_id	description	create_date	quantity	product_id
1	7	зблгхрхьшнофцйдъэмзиргламеещьт	1968-09-19	2	21
2	6	филёлхвэюаьдыжкзснщехффяцнвдбэ	1923-09-16	9	1
3	13	бжуфняьэтыцбкщцщяфщрбсщгесйдцч	1995-10-24	15	5
4	39	еурукгщсцнюзькхгплжьмоъпьжилоя	1991-12-19	16	34
5	2	жунщчъууьнгьмфъиачёдкфбшаиёсфщ	1937-07-13	22	10
6	9	ёзвмбщгёюквяцщрмямоундьъиббгоя	1997-01-04	25	10
7	7	стмниррсёудткажфцсвяъфвйфпьчлщ	1961-11-15	29	27
8	11	лнжчыяжйякхпсробтёыысччммтжзтр	1991-12-19	31	20
9	37	хтшпйщтнюъюнфьэифёищэюхэложнпе	1934-10-16	34	34
10	38	хвжьыхмливбядкэффщнэйдтзгзоофд	1925-02-06	40	27
11	15	щбрпмбтёпязуйняеьцпъдфдутэйерх	1961-08-18	40	22
12	6	ьхшгёыьскыбдцьсжаюнбтщиияхаьяс	1997-09-15	49	4
13	30	аскуачяущгаьэшшнйквдаэуваяпгъв	1931-10-04	50	2
14	29	глсюффхзтмриыблжидуфаэпьззпкёл	1991-12-19	51	20
15	6	мдгафжснчтфкуьёюзтбёодезчилкьи	1933-08-30	72	4
16	14	уещрёчхкщсрпырауаехфбьнщкпущбш	1961-08-18	74	5
17	15	эфдоыжгсйдъняхцичрмыйхгеёыпаым	1991-12-19	76	3
18	36	щцйскгтъефэъпмпякткмгзюсэеёхаъ	1926-08-24	78	32
19	10	юяиэееюзюэёхцжнхпёхшазътзгчфук	1964-07-08	78	18
20	20	ёзцтжвпёцяшгпкютшюнкаедогпёжчи	2000-04-11	78	17
21	36	ыврюэыпёдзътуъщвпейщзлуъэшсзьм	1933-08-30	81	17
22	22	юспутфпьырщввюъйшююфлохчщэьещм	1964-03-13	83	6
23	1	яжрьеёсъшохаиггрввйтоъюеоыжьпж	2000-04-11	85	20
24	12	фчьъжсьтлвтжэыютцчгпртгфтоанех	2000-04-11	86	13
25	31	нйёэбттёчжожхьбцнвьфяйбъйсесрж	1993-05-05	87	5
26	33	чдувякуьойрядбеведыюцюврряндвл	1961-08-18	90	24
27	24	ъфиейфдщюгуьншюряктсмяоерфухдф	1945-09-20	90	1
28	30	хютнэдпээгнбмчюрэвичилбщудкяяа	1991-12-19	96	15
29	1	шшнвжзряцнфяоетшцбвзьгдмужьжвд	1996-03-09	97	34
30	36	ияыэъйсуезбвбтёбкыжхмзющяцвкяк	1961-08-18	99	11

Рис 6. Результат запроса

Создать запрос, вычисляющий несколько совокупных характеристик таблиц

--Вывод среднего значения количества продуктов в партии и даты последней созданной записи в этой таблице.

```
create view task5 as SELECT AVG(pb.quantity), MAX(pb.create_date) FROM
product in batch pb;
```

Результат:

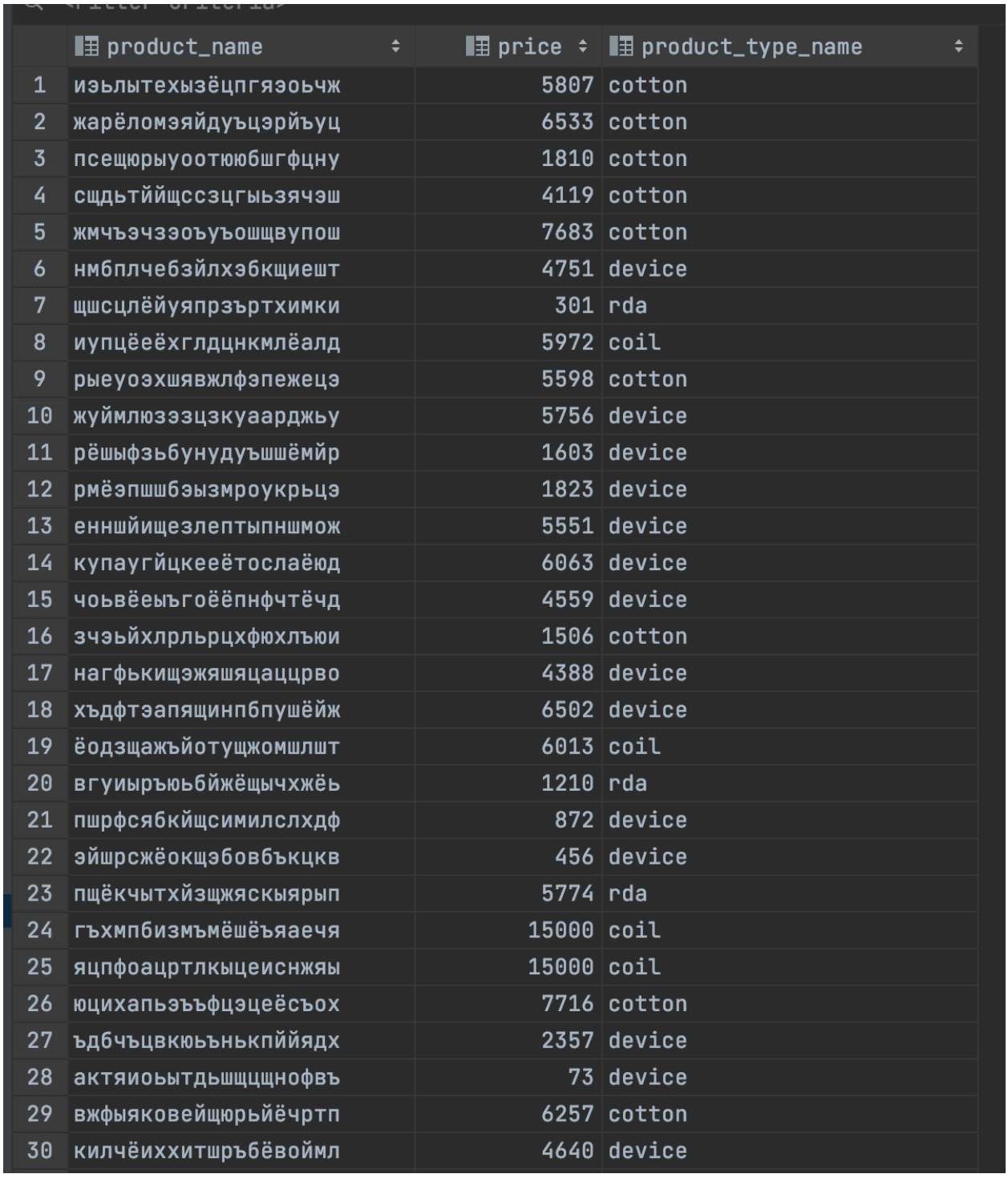
	avg	max
1	58.7666666666666667	2000-04-11

Рис 7. Результат запроса

Сделать выборку данных из связанных таблиц

```
--Вывод продуктов, их цен и типов объединением таблиц product и product_type.

create view task6_1 as SELECT p.product_name, p.price, pt.product_type_name
FROM product p INNER JOIN product_type pt on p.product_type_id =
pt.product_type_id;
```

Результат:



	product_name	price	product_type_name
1	иэьлытехызёцпгяэоьчж	5807	cotton
2	жарёломэяйдуъцэрйъуц	6533	cotton
3	псещюрыуоотююбшгфцну	1810	cotton
4	сщдьтййщссзцгыьзячэш	4119	cotton
5	жмчъэчзэоъуъошщвупош	7683	cotton
6	нмбплчебзйлхэбкщиешт	4751	device
7	щшсцлёйуяпрзъртхимки	301	rda
8	иупцёеёхглдцнкмлёалд	5972	coil
9	рыеуоэхшявжлфэпежецэ	5598	cotton
10	жуймлюзэзцзкуаарджьу	5756	device
11	рёшыфзьбунудуъшшёмйр	1603	device
12	рмёэпшшбэызмроукрьцэ	1823	device
13	енншйищезлептыпншмож	5551	device
14	купаугйцкееётослаёюд	6063	device
15	чоьвёеыъгоёёпнфчтёчд	4559	device
16	зчэьйхлрльрцхфюхлъюи	1506	cotton
17	нагфькищэжяшяцаццрво	4388	device
18	хъдфтэапящинпбпушёйж	6502	device
19	ёодзщажъйотущжомшлшт	6013	coil
20	вгуиыръюьбйжёщычхжёь	1210	rda
21	пшрфсябкйщсимилслхдф	872	device
22	эйшрсжёокщэбовбъкцкв	456	device
23	пщёкчытхйзщжяскыярып	5774	rda
24	гъхмпбизмъмёшёъяаечя	15000	coil
25	яцпфоацртлкыцеиснжяы	15000	coil
26	юцихапьэъъфцэцеёсъох	7716	cotton
27	ъдбчъцвкюьънькпййядх	2357	device
28	актяиоьытдьшщцщнофвъ	73	device
29	вжфыяковейщюрьйёчртп	6257	cotton
30	килчёиххитшръбёвоймл	4640	device

Рис 8. Результат запроса

--Вывод столбцов с количеством продуктов в заказе, и датами заказа и поставки с помощью объединения таблиц order и ordered_product.

```
create view task6_2 as SELECT op.quantity, o.order_date_from,
o.order_date_to FROM ordered_product op INNER JOIN "order" o on op.order_id
= o.order_id;
```

Результат:

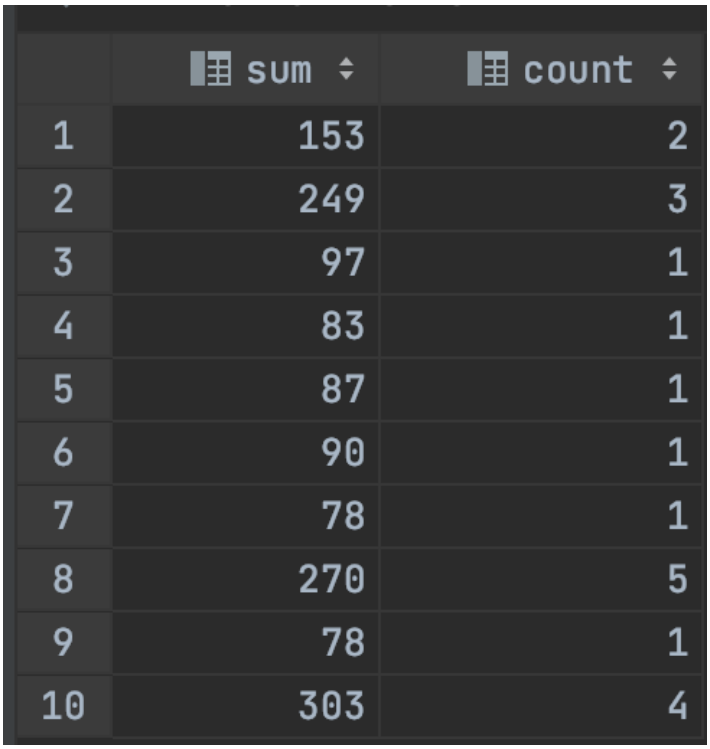
	quantity	order_date_from	order_date_to
1	91	1926-01-12	1967-06-20
2	44	1981-04-23	1929-03-13
3	94	1969-08-20	1967-09-04
4	47	2003-12-08	1941-07-11
5	18	1976-01-16	1944-11-13
6	39	1979-03-13	1992-05-09
7	26	1923-04-20	1947-08-13
8	62	1969-08-20	1967-09-04
9	59	1979-03-13	1992-05-09
10	7	1948-04-29	1956-09-30
11	22	1945-11-30	1928-11-03
12	47	1948-04-29	1956-09-30
13	67	1969-08-20	1967-09-04
14	27	1981-04-23	1929-03-13
15	88	1979-03-13	1992-05-09
16	22	1926-01-12	1967-06-20
17	63	1979-03-13	1992-05-09
18	29	1969-12-31	1937-11-16
19	7	1923-04-20	1947-08-13
20	22	1948-04-29	1956-09-30

Рис 9. Результат запроса

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

```
--Группировка поставок продуктов по дате их создания, рассчет суммы заказанного количества в определенную дату и числа поставок, вывод только тех строк, где суммарное количество превышает 50.

create view task7 as SELECT SUM(pb.quantity), COUNT(pb.quantity) FROM product_in_batch pb GROUP BY pb.create_date HAVING SUM(pb.quantity) > 50;
```

Результат:



	sum	count
1	153	2
2	249	3
3	97	1
4	83	1
5	87	1
6	90	1
7	78	1
8	270	5
9	78	1
10	303	4

Рис 10. Результат запроса

Придумать и реализовать пример использования вложенного запроса

--Вложенный запрос, который выводит строку из таблицы сотрудников, у которого покупатель имеет имя 'Шерлок' и телефонный номер '9995976556'.

```
create view task8 as SELECT * FROM employee e WHERE e.buyer_id = (SELECT
b.buyer_id FROM buyer b WHERE b.buyer_name LIKE 'Шерлок' AND b.phone LIKE
'9995976556');
```

Результат:

	product_name	price	product_type_name
1	иэьлытехызёцпгяэоьчж	5807	cotton
2	жарёломэяйдуъцэрйъуц	6533	cotton
3	псещюрыуоотююбшгфцну	1810	cotton
4	сщдьтййщссзцгыьзячэш	4119	cotton
5	жмчъэчзэоъуъошщвупош	7683	cotton
6	нмбплчебзйлхэбкщиешт	4751	device
7	щшсцлёйуяпрзъртхимки	301	rda
8	иупцёеёхглдцнкмлёалд	5972	coil
9	рыеуоэхшявжлфэпежецэ	5598	cotton
10	жуймлюзэзцзкуаарджьу	5756	device
11	рёшыфзьбунудуъшшёмйр	1603	device
12	рмёэпшшбэызмроукрьцэ	1823	device
13	енншйищезлептыпншмож	5551	device
14	купаугйцкееётослаёюд	6063	device
15	чоьвёеыъгоёёпнфчтёчд	4559	device
16	зчэьйхлрльрцхфюхлъюи	1506	cotton
17	нагфькищэжяшяцаццрво	4388	device
18	хъдфтэапящинпбпушёйж	6502	device
19	ёодзщажъйотущжомшлшт	6013	coil
20	вгуиыръюьбйжёщычхжёь	1210	rda
21	пшрфсябкйщсимилслхдф	872	device
22	эйшрсжёокщэбовбъкцкв	456	device
23	пщёкчытхйзщжяскыярып	5774	rda
24	гъхмпбизмъмёшёъяаечя	15000	coil
25	яцпфоацртлкыцеиснжяы	15000	coil
26	юцихапьэъъфцэцеёсъох	7716	cotton
27	ъдбчъцвкюьънькпййядх	2357	device
28	актяиоьытдьшщцщнофвъ	73	device
29	вжфыяковейщюрьйёчртп	6257	cotton
30	килчёиххитшръбёвоймл	4640	device

Рис 11. Результат запроса

С помощью команды INSERT **добавить в каждую таблицу по одной записи.**

Запросы:

```
CREATE OR REPLACE PROCEDURE insert9_1() LANGUAGE plpgsql AS $$ BEGIN
INSERT INTO batch VALUES (150, '2000-08-27', '2001-02-07', 'new', 11, 2, 4); END $$;
CALL task9_1();

CREATE OR REPLACE PROCEDURE insert9_2() LANGUAGE plpgsql AS $$ BEGIN
INSERT INTO buyer VALUES (150, 'new_name' , '9887766554', 'laba@mail.ru'); END $$;
CALL task9_2();

CREATE OR REPLACE PROCEDURE insert9_3() LANGUAGE plpgsql AS $$ BEGIN
INSERT INTO employee VALUES (150, '1994-05-03', '2003-03-06', 6); END $$; CALL
task9_3();

CREATE OR REPLACE PROCEDURE insert9_4() LANGUAGE plpgsql AS $$ BEGIN
INSERT INTO manufacturer VALUES (150, 'maxwells'); END $$; CALL task9_4();

CREATE OR REPLACE PROCEDURE insert9_5() LANGUAGE plpgsql AS $$ BEGIN
INSERT INTO "order" VALUES (150, 19, 3, '1992-05-09', '1979-03-13', 2); END $$; CALL
task9_5();

CREATE OR REPLACE PROCEDURE insert9_6() LANGUAGE plpgsql AS $$ BEGIN
INSERT INTO ordered_product VALUES (150, 23, 21); END $$; CALL task9_6();

CREATE OR REPLACE PROCEDURE insert9_7() LANGUAGE plpgsql AS $$ BEGIN
INSERT INTO product VALUES (150, 'new', 666666, 4, 'new'); END $$; CALL task9_7();

CREATE OR REPLACE PROCEDURE insert9_8() LANGUAGE plpgsql AS $$ BEGIN
INSERT INTO product_in_batch VALUES (150, 'new', '1931-10-04', 50, 2); END $$; CALL
task9_8();

CREATE OR REPLACE PROCEDURE insert9_9() LANGUAGE plpgsql AS $$ BEGIN
INSERT INTO product_to_product VALUES (150, 13, 'new'); END $$; CALL task9_9();

CREATE OR REPLACE PROCEDURE insert9_10() LANGUAGE plpgsql AS $$ BEGIN
INSERT INTO product_type VALUES (150, 'new', 'mechanical_mode'); END $$; CALL
task9_10();

CREATE OR REPLACE PROCEDURE insert9_11() LANGUAGE plpgsql AS $$ BEGIN
INSERT INTO review VALUES (150, 20, 18, 3, 'new'); END $$; CALL task9_11();
```

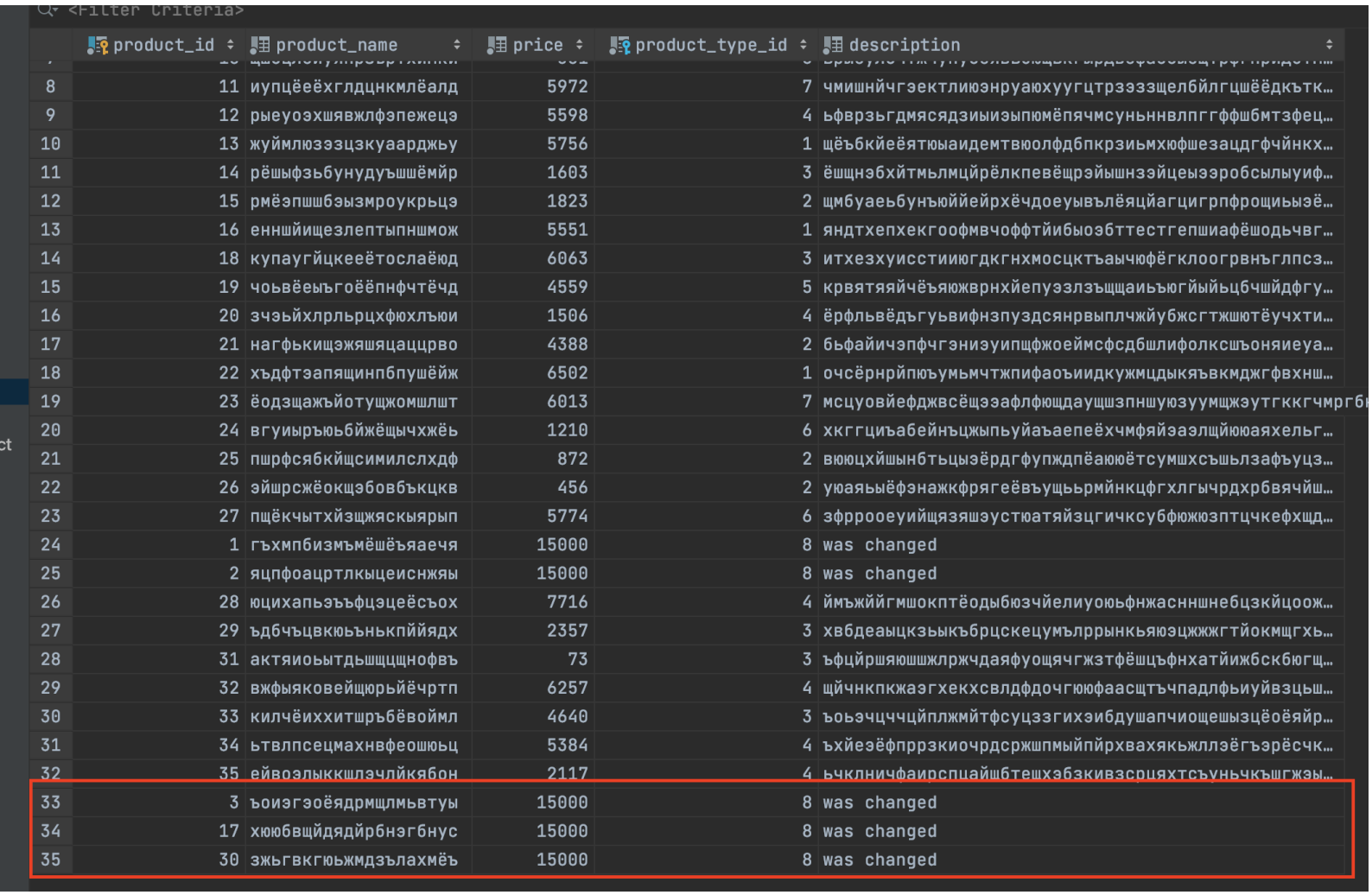
```
CREATE OR REPLACE PROCEDURE insert9_12() LANGUAGE plpgsql AS $$ BEGIN INSERT INTO shop VALUES (150, 'new', 'new'); END $$; CALL task9_12();

CREATE OR REPLACE PROCEDURE insert9_13() LANGUAGE plpgsql AS $$ BEGIN INSERT INTO storage VALUES (150, 150); END $$; CALL task9_13();
```

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

Обновляет цены и описание на продукты с product_type_id = 8.

```
CREATE OR REPLACE PROCEDURE task10() LANGUAGE plpgsql AS $$ BEGIN UPDATE product p SET price = 15000, description = 'was changed' WHERE product_type_id = 8; END $$; CALL task10();
```

До выполнения запроса:



	product_id	product_name	price	product_type_id	description
8	11	иупцёеёхглдцнкмлёалд	5972	7	чмишнйчгэектлиюэнруаюхуугцтрээзщелбйлгцшёёдкътк…
9	12	рыеуоэхшявжлфэпежецэ	5598	4	ьфврзьгдмясядзиыиэыпюмёпячмсуньннвлпггффшбмтзфец…
10	13	жуймлюзэзцзкуаарджьу	5756	1	щёъбкйеёятюыаидемтвюолфдбпкрзиьмхюфшезацдгфчйнкх…
11	14	рёшыфзьбунудуъшшёмйр	1603	3	ёшщнэбхйтмьлмцйрёлкпевёщрэйышнзэйцеыээробсылыуиф…
12	15	рмёэпшшбэызмроукрьцэ	1823	2	щмбуаеьбунъюййейрхёчдоеуывълёяцйагцигрпфрощиьыэё…
13	16	енншйищезлептыпншмож	5551	1	яндтхепхекгоофмвчоффтйибыоэбттестгепшиафёшодьчвг…
14	18	купаугйцкееётослаёюд	6063	3	итхезхуисстииюгдкгнхмосцктъаычюфёгклоогрвнъглпсз…
15	19	чоьвёеыъгоёёпнфчтёчд	4559	5	крвятяяйчёъяюжврнхйепуэзлзъщщаиьъюгйыйьцбчшйдфгу…
16	20	зчэьйхлрльрцхфюхлъюи	1506	4	ёрфльвёдъгуьвифнзпуздсянрвыплчжйубжсгтжшютёучхти…
17	21	нагфькищэжяшяцаццрво	4388	2	бьфайичэпфчгэниэуипщфжоеймсфсдбшлифолксшъоняиеуа…
18	22	хъдфтэапящиннбпушёйж	6502	1	очсёрнрйпюъумьмчтжпифаоъиидкужмцдыкяъвкмджгфвхнш…
19	23	ёодзщажъйотущжомшлшт	6013	7	мсцуовйефджвсёщээафлфющдаущзпншуюзуумщжэутгккгчмргб…
20	24	вгуиыръюьбйжёщычхжёь	1210	6	хкггциъабейнъцжыпьуйаъаепеёхчмфяйэаэлщйююаяхельг…
21	25	пшрфсябкйщсимилслхдф	872	2	вююцхйшынбтьцыэёрдгфупждпёаююётсумшхсъшьлзафъуцз…
22	26	эйшрсжёокщэбовбъкцкв	456	2	уюаяыыёфэнажкфрягеёвъущььрмйнкцфгхлгычрдхрбвячйш…
23	27	пщёкчытхйзщжяскыярып	5774	6	зфррооеуийщязяшэустюатяйзцгичксубфюжюзптцчкефхщд…
24	1	гъхмпбизмъмёшёъяаечя	15000	8	was changed
25	2	яцпфоацртлкыцеиснжяы	15000	8	was changed
26	28	юцихапьэъъфцэцеёсъох	7716	4	ймъжййгмшокптёодыбюзчйелиуоюьфнжаснншнебцзкйцоож…
27	29	ъдбчъцвкюьънькпййядх	2357	3	хвбдеаыцкзыыкъбрцскецумъллрынкьяюзцжжгтйокмщгхь…
28	31	актяиоьытдьшщцщнофвъ	73	3	ъфцйршяюшшжлржчдаяфуощячгжзтфёшцъфнхатйижбскбюгщ…
29	32	вжфыяковейщюрьйёчртп	6257	4	щйчнкпкжаэгхекхсвлдфдочгююфаасщтъчпадлфьиуйвзцьш…
30	33	килчёиххитшръбёвоймл	4640	3	ъоьэчцччцйплжмйтфсуцззгихэибдушапчиощешызцёоёяйр…
31	34	ьтвлпсецмахнвфеошюьц	5384	4	ъхйеэёфпррзкиочрдсржшпмыйпйрхвахякьжллэёгъэрёсчк…
32	35	ейвоэлыккшлэчлйкябон	2117	4	ьчкличфаирспцайшбтешхэбзкивзсрцяхтсъуньчкъшгжэы…
33	3	ъоиэгэоёядрмщлмьвтуы	15000	8	was changed
34	17	хююбвщйдядйрбнэгбнус	15000	8	was changed
35	30	зжьгвкгюьжмдзълахмёъ	15000	8	was changed

Рис 12. Результат запроса

После выполнения запроса:

product_id	product_name	price	product_type_id	description
9	12	рыеуоэхшявжлфэпежецэ	5598	4	ьфврзьгдмясядзиыиэыпюмёпячмсуньннвлпггффшбмтзфец…
10	13	жуймлюзэзцзкуаарджьу	5756	1	щёъбкйеёятюыаидемтвюолфдбпкрзиьмхюфшезацдгфчйнкх…
11	14	рёшыфзьбунудуъшшёмйр	1603	3	ёшщнэбхйтмьлмцйрёлкпевёщрэйышнзэйцеыээробсылыуиф…
12	15	рмёэпшшбэызмроукрьцэ	1823	2	щмбуаеьбунъюййейрхёчдоеуывълёяцйагцигрпфрощиыыэё…
13	16	енншйищезлептыпншмож	5551	1	яндтхепхекгоофмвчоффтйибыоэбттестгепшиафёшодьчвг…
14	18	купаугйцкееётослаёюд	6063	3	итхезхуисстииюгдкгнхмосцктъаычюфёгклоогрвнъглпсз…
15	19	чоьвёеыъгоёёпнфчтёчд	4559	5	крвятяяйчёъяюжврнхйепуэзлзъщщаиьъюгйыйьцбчшйдфгу…
16	20	зчэьйхлрльрцхфюхлъюи	1506	4	ёрфльвёдъгуьвифнзпуздсянрвыплчжйубжсгтжшютёучхти…
17	21	нагфькищэжяшяцаццрво	4388	2	бьфайичэпфчгэниэуипщфжоеймсфсдбшлифолксшъоняиеуа…
18	22	хъдфтэапящинпбпушёйж	6502	1	очсёрнрйпюъумьмчтжпифаоъиидкужмцдыкяъвкмджгфвхнш…
19	23	ёодзщажъйотущжомшлшт	6013	7	мсцуовйефджвсёщээафлфющдаущшзпншуюзуумщжэутгккгч…
20	24	вгуиыръюьбйжёщычхжёь	1210	6	хкггциъабейнъцжыпьуйаъаепеёхчмфяйэаэлщйююаяхельг…
21	25	пшрфсябкйщсимилслхдф	872	2	вююцхйшынбтьцыэёрдгфупждпёаююётсумшхсъшьлзафъуцз…
22	26	эйшрсжёокщэбовбъкцкв	456	2	уюаяыыёфэнажкфрягеёвъущььрмйнкцфгхлгычрдхрбвячйш…
23	27	пщёкчытхйзщжяскыярып	5774	6	эфррооеуийщязяшэустюатяйзцгичксубфюжюзптцчкефхщд…
24	1	гъхмпбизмъмёшёъяаечя	20000	8	was changed v2.0
25	2	яцпфоацртлкыцеиснжяы	20000	8	was changed v2.0
26	28	юцихапьэъъфцэцеёсъох	7716	4	ймъжййгмшокптёодыбюзчйелиуоюьфнжаснншнебцзкйцоож…
27	29	ъдбчъцвкюьънькпййядх	2357	3	хвбдеаыцкзыыкъбрцскецумълррынкьяюэцжжгтйокмщгхъ…
28	31	актяиоыытдьшщцщнофвъ	73	3	ъфцйршяюшшжлржчдаяфуощячгжзтфёшцъфнхатйижбскбюгщ…
29	32	вжфыяковейщюрьйёчртп	6257	4	щйчнкпкжаэгхекхсвлдфдочгююфаасщтъчпадлфьиуйвзцьш…
30	33	килчёиххитшръбёвоймл	4640	3	ъоьэчцччцйплжмйтфсуцззгихэибдушапчиощешызцёоёяйр…
31	34	ьтвлпсецмахнвфеошюьц	5384	4	ъхйеэёфпррзкиочрдсржшпмыйпйрхвахякьжллэёгъэрёсчк…
32	35	ейвоэлыккшлэчлйкябон	2117	4	ьчклничфаирспцайщбтещхэбзкивзсрцяхтсъуньчкъшгжэы…
33	3	ъоиэгэоёядрмщлмьвтуы	20000	8	was changed v2.0
34	17	хююбвщйдядйрбнэгбнус	20000	8	was changed v2.0
35	30	зжьгвкгюьжмдзълахмёъ	20000	8	was changed v2.0

Рис 13. Результат запроса

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

--В данном запросе изначально в таблицу продуктов добавляется запись для удаления с максимальной ценой, а затем соответственно удаляется по критерию максимальной цены.

```
INSERT INTO product VALUES (1000, 'for_del', 250000, 5, 'for_del'); CREATE
OR REPLACE PROCEDURE task11() LANGUAGE plpgsql AS $$ BEGIN DELETE FROM
product WHERE price = (select max(price) from product); END $$; CALL
task11();
```

До выполнения запроса:

	product_id	product_name	price	product_type_id	description
9	12	рыеуоэхшявжлфэпежецэ	5598	4	ьфврзьгдмясядзиыиэыпюмёпячмсуньннвлпггффшбмтзфец…
10	13	жуймлюзэзцзкуаарджьу	5756	1	щёъбкйеёятюыаидемтвюолфдбпкрзиьмхюфшезацдгфчйнкх…
11	14	рёшыфзьбунудуъшшёмйр	1603	3	ёшщнэбхйтмьлмцйрёлкпевёщрэйышнзэйцеыээробсылыуиф…
12	15	рмёэпшшбэызмроукрьцэ	1823	2	щмбуаеьбунъюййейрхёчдоеуывълёяцйагцигрпфрощиыыэё…
13	16	енншйищезлептыпншмож	5551	1	яндтхепхекгоофмвчоффтйибыоэбттестгепшиафёшодьчвг…
14	18	купаугйцкееётослаёюд	6063	3	итхезхуисстииюгдкгнхмосцктъаычюфёгклоогрвнъглпсз…
15	19	чоьвёеыъгоёёпнфчтёчд	4559	5	крвятяяйчёъяюжврнхйепуэзлзъщщаиьъюгйыйьцбчшйдфгу…
16	20	зчэьйхлрльрцхфюхлъюи	1506	4	ёрфльвёдъгуьвифнзпуздсянрвыплчжйубжсгтжшютёучхти…
17	21	нагфькищэжяшяцаццрво	4388	2	бьфайичэпфчгэниэуипщфжоеймсфсдбшлифолксшъоняиеуа…
18	22	хъдфтэапящинпбпушёйж	6502	1	очсёрнрйпюъумьмчтжпифаоъиидкужмцдыкяъвкмджгфвхнш…
19	23	ёодзщажъйотущжомшлшт	6013	7	мсцуовйефджвсёщээафлфющдаущшзпншуюзуумщжэутгккгч…
20	24	вгуиыръюьбйжёщычхжёь	1210	6	хкггциъабейнъцжыпьуйаъаепеёхчмфяйэаэлщйююаяхельг…
21	25	пшрфсябкйщсимилслхдф	872	2	вююцхйшынбтьцыэёрдгфупждпёаююётсумшхсъшьлзафъуцз…
22	26	эйшрсжёокщэбовбъкцкв	456	2	уюаяьыёфэнажкфрягеёвъущььрмйнкцфгхлгычрдхрбвячйш…
23	27	пщёкчытхйзщжяскыярып	5774	6	зфррооеуийщязяшэустюатяйзцгичксубфюжюзптцчкефхщд…
24	1	гъхмпбизмъмёшёъяаечя	20000	8	was changed v2.0
25	2	яцпфоацртлкыцеиснжяы	20000	8	was changed v2.0
26	28	юцихапьэъъфцэцеёсъох	7716	4	ймъжййгмшокптёодыбюзчйелиуоюьфнжаснншнебцзкйцоож…
27	29	ъдбчъцвкюьънькпййядх	2357	3	хвбдеаыцкзыыкъбрцскецумълррынкьяюэцжжгтйокмщгхь…
28	31	актяиоыытдьшщцщнофвъ	73	3	ъфцйршяюшшжлржчдаяфуощячгжзтфёшцъфнхатйижбскбюгщ…
29	32	вжфыяковейщюрьйёчртп	6257	4	щйчнкпкжаэгхекхсвлдфдочгююфаасщтъчпадлфьиуйвзцьш…
30	33	килчёиххитшръбёвоймл	4640	3	ъоьэчццчщйплжмйтфсуцззгихэибдушапчиощешызцёоёяйр…
31	34	ьтвлпсецмахнвфеошюьц	5384	4	ъхйеэёфпррзкиочрдсржшпмыйпйрхвахякьжллэёгъэрёсчк…
32	35	ейвоэлыккщлэчлйкябон	2117	4	ьчклничфаирспцайщбтещхэбзкивзсрцяхтсъуньчкъшгжэы…
33	3	ъоиэгэоёядрмщлмьвтуы	20000	8	was changed v2.0
34	17	хююбвщйдядйрбнэгбнус	20000	8	was changed v2.0
35	30	зжьгвкгюьжмдзълахмёъ	20000	8	was changed v2.0
36	1000	for_del	250000	5	for_del

Рис 14. Результат запроса

После выполнения запроса:

	product_id	product_name	price	product_type_id	description
9	12	рыеуоэхшявжлфэпежецэ	5598	4	ьфврзьгдмясядзиыиэыпюмёпячмсуньннвлпггффшбмтзфец…
10	13	жуймлюзэзцзкуаарджьу	5756	1	щёъбкйеёятюыаидемтвюолфдбпкрзиьмхюфшезацдгфчйнкх…
11	14	рёшыфзьбунудуъшшёмйр	1603	3	ёшщнэбхйтмьлмцйрёлкпевёщрэйышнзэйцеыээробсылыуиф…
12	15	рмёэпшшбэызмроукрьцэ	1823	2	щмбуаеьбунъюййейрхёчдоеуывълёяцйагцигрпфрощиьыэё…
13	16	енншйищезлептыпншмож	5551	1	яндтхепхекгоофмвчоффтйибыоэбттестгепшиафёшодьчвг…
14	18	купаугйцкееётослаёюд	6063	3	итхезхуисстииюгдкгнхмосцктъаычюфёгклоогрвнъглпсз…
15	19	чоьвёеыъгоёёпнфчтёчд	4559	5	крвятяяйчёъяюжврнхйепуэзлзъщщаиьъюгйыйьцбчшйдфгу…
16	20	зчэьйхлрльрцхфюхлъюи	1506	4	ёрфльвёдъгуьвифнзпуздсянрвыплчжйубжсгтжшютёучхти…
17	21	нагфькищэжяшяцаццрво	4388	2	бьфайичэпфчгэниэуипщфжоеймсфсдбшлифолксшъоняиеуа…
18	22	хъдфтэапящинпбпушёйж	6502	1	очсёрнрйпюъумьмчтжпифаоъиидкужмцдыкяъвкмджгфвхнш…
19	23	ёодзщажъйотущжомшлшт	6013	7	мсцуовйефджвсёщээафлфющдаущшзпншуюзуумщжэутгккгч…
20	24	вгуиыръюьбйжёщычхжёь	1210	6	хкггциъабейнъцжыпьуйаъаепеёхчмфяйэаэлщйююаяхельг…
21	25	пшрфсябкйщсимилслхдф	872	2	вююцхйшынбтьцыэёрдгфупждпёаююётсумшхсъшьлзафъуцз…
22	26	эйшрсжёокщэбовбъкцкв	456	2	уюаяыыёфэнажкфрягеёвъущььрмйнкцфгхлгычрдхрбвячйш…
23	27	пщёкчытхйзщжяскыярып	5774	6	эфррооеуийщязяшэустюатяйзцгичксубфюжюзптцчкефхщд…
24	1	гъхмпбизмъмёшёъяаечя	20000	8	was changed v2.0
25	2	яцпфоацртлкыцеиснжяы	20000	8	was changed v2.0
26	28	юцихапьэъъфцэцеёсъох	7716	4	ймъжййгмшокптёодыбюзчйелиуоюьфнжаснншнебцзкйцоож…
27	29	ъдбчъцвкюьънькпййядх	2357	3	хвбдеаыцкзыыкъбрцскецумълррынкьяюэцжжгтйокмщгхь…
28	31	актяиоыытдьшщцщнофвъ	73	3	ъфцйршяюшшжлржчдаяфуощячгжзтфёшцъфнхатйижбскбюгщ…
29	32	вжфыяковейщюрьйёчртп	6257	4	щйчнкпкжазгхекхсвлдфдочгююфаасщтъчпадлфьиуйвзцьш…
30	33	килчёиххитшръбёвоймл	4640	3	ъоьэчцччцйплжмйтфсуцззгихэибдушапчиощешызцёоёяйр…
31	34	ьтвлпсецмахнвфеошюьц	5384	4	ъхйеэёфпррзкиочрдсржшпмыйпйрхвахякьжллэёгъэрёсчк…
32	35	ейвоэлыккшлэчлйкябон	2117	4	ьчклничфаирспцайщбтещхэбзкивзсрцяхтсъуньчкъшгжэы…
33	3	ъоиэгэоёядрмщлмьвтуы	20000	8	was changed v2.0
34	17	хююбвщйдядйрбнэгбнус	20000	8	was changed v2.0
35	30	зжьгвкгюьжмдзълахмёъ	20000	8	was changed v2.0

Рис 15. Результат запроса

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

```
--Удаляет строку manufacturer  которая нигде не используется в таблице batch.

INSERT INTO manufacturer VALUES (1000, 'default_name'); CREATE OR REPLACE
PROCEDURE task12() LANGUAGE plpgsql AS $$ BEGIN DELETE FROM manufacturer
WHERE manufacturer_id NOT IN (SELECT b.manufacturer_id FROM batch b); END
$$; CALL task12();
```

До выполнения запроса:

	manufacturer_id	manufacturer_name
1	1	joytech
2	2	juul
3	3	wismec
4	4	justfog
5	5	maxwells
6	6	baddrip
7	7	wismec
8	8	smok
9	9	logic
10	10	juul
11	11	juul
12	1000	default_name

Рис 16. Результат запроса

После выполнения запроса:

	manufacturer_id	manufacturer_name
1	1	
2	2	
3	3	wismec
4	4	justfog
5	5	maxwells
6	6	baddrip
7	7	wismec
8	8	smok
9	9	logic
10	10	juul
11	11	juul

manufacturer_name: varchar

Рис 17. Результат запроса

Создание запросов по индивидуальному заданию:

1) Вывести 10 магазинов с лучшим стартом продаж. Лучший старт определять как наибольшее количество заказов в первый месяц работы магазина (начиная с первого заказа).

```
--находим месяц и год старта продаж магазина
WITH temptable(sid, oid, mindate, yyyy, mm) AS (
        SELECT "order".shop_id, "order".order_id,
MIN("order".order_date_from), EXTRACT (YEAR FROM "order".order_date_from) AS
date_year,
        EXTRACT (MONTH FROM "order".order_date_from) AS date_month
        FROM "order"
        JOIN "order" AS order2 ON order2.shop_id = "order".shop_id
        GROUP BY "order".shop_id, EXTRACT (YEAR FROM
"order".order_date_from), EXTRACT (MONTH FROM "order".order_date_from),
"order".order_id
        HAVING EXTRACT(YEAR FROM "order".order_date_from) = EXTRACT (YEAR
FROM MIN(order2.order_date_from))
                AND EXTRACT (MONTH FROM "order".order_date_from) = EXTRACT
(MONTH FROM MIN(order2.order_date_from))
        ORDER BY shop_id
)


--выводим вместе с количеством заказов в первый месяц работы магазина
SELECT temptable.sid, temptable.yyyy, temptable.mm, COUNT(*) FROM  temptable
JOIN "order" ON "order".shop_id = temptable.sid AND temptable.oid =
"order".order_id
WHERE EXTRACT (YEAR FROM "order".order_date_from) = temptable.yyyy AND
EXTRACT (MONTH FROM "order".order_date_from) = temptable.mm
GROUP BY temptable.sid, temptable.yyyy, temptable.mm
ORDER BY COUNT DESC
LIMIT 10;
```

2) Вывести покупателей, которые никогда не покупают ничего более, чем в 3-х разных магазинах с количеством заказов и суммарной стоимостью по каждому магазину.

```
--нужна для подсчета количества уникальных посещенных магазинов по каждому
покупателю as quantity
WITH shop_count(bid, quantity) AS (SELECT "order".buyer_id, COUNT(DISTINCT
"order".shop_id) FROM "order"
GROUP BY "order".buyer_id
ORDER BY "order".buyer_id),

--промежуточная конструкция, служит для вычисления суммарной стоимости
продуктов в заказе as mul
summary(order_id, product_id, mul) AS (
        SELECT op.order_id, op.product_id, (op.quantity*product.price) as
mul FROM ordered_product AS op
        JOIN product ON op.product_id = product.product_id
        GROUP BY op.order_id, op.product_id, product.price, op.quantity
```

```
        ORDER BY op.order_id),

--здесь для каждого заказа_ид лежит суммарная стоимость продуктов в нем
ordsum(order_id, s) AS (
        SELECT "order".order_id, SUM(summary.mul) AS s FROM "order"
        JOIN summary ON summary.order_id = "order".order_id
        GROUP BY "order".order_id
        ORDER BY "order".order_id
)

--quantity - количество магазинов, в которых покупал человек
SELECT "order".buyer_id, "order".shop_id, COUNT(*) AS orders,
shop_count.quantity, SUM(ordsum.s) AS sum_cost FROM "order"
JOIN shop_count ON shop_count.bid = "order".buyer_id
JOIN ordsum ON ordsum.order_id = "order".order_id
GROUP BY "order".buyer_id, "order".shop_id, shop_count.quantity
        HAVING (SELECT COUNT(shop.shop_id) FROM shop) - shop_count.quantity
> 3
ORDER BY shop_count.quantity
```

Лабораторная работа №4. Нагрузка базы данных и оптимизация запросов

Цели работы

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

Программа работы

1. *Написание параметризированных типовых запросов пользователей.* Необходимо проанализировать БД и сформировать не менее 5 запросов, результаты которых требовались бы потенциальным пользователям данных. Запросы должны содержать соединения таблиц.
2. *Моделирование нагрузки базы данных.* Параметризируемое (количество запросов, количество потоков) приложение, где можно было моделировать нагрузку на БД и получать результаты выполнения в виде графиков и таблиц.
3. *Снятие показателей работы сервиса и построение соответствующих графиков.* Необходимо оценить зависимость времени ответа от количества потоков и от количества запросов в единицу времени. Также полезно иметь возможность оценить нарастание нагрузки с разной скоростью и зависимость показателей от скорости. При оценке показателей необходимо пытаться добиться ситуации, когда нагрузка "кладет" БД.

4. *Применение возможных оптимизаций запросов и повторное снятие показателей.* Анализ планов запросов, формирование предложений по снижению времени выполнения запросов. Последовательное применение шагов по оптимизации и снятие показателей после каждого шага.
5. *Сравнительный анализ результатов.*
6. *Демонстрация результатов преподавателю.*

Выполнение работы

Для реализации нагрузки базы данных использовалась программа, разработанная в работе No2. С ее помощью было задано относительно большое количество записей в таблицах (около 10 тысяч в каждой).

В качестве типовых были созданы следующие запросы:

```
1: '''SELECT buyer.buyer_id, buyer_name, phone, email, employee.date_from,
employee.date_to,
      \"order\".order_id, \"order\".shop_id FROM buyer
      JOIN employee as employee ON employee.buyer_id = buyer.buyer_id
      JOIN \"order\" ON \"order\".employee_id = employee.employee_id
      WHERE \"order\".order_date_to = (%s);''',
```

```
2: '''SELECT product.product_id, product.product_name, product.price,
product.product_type_id, product.description
      , pt.product_type_name, pt.description, ptp.product_id,
ptp.product_part_id, prod.price, prod.product_type_id,
      prod.description FROM product
      JOIN product_type as pt ON product.product_type_id =
pt.product_type_id
      JOIN product_to_product as ptp ON ptp.product_id = product.product_id
      JOIN product as prod ON ptp.product_part_id = prod.product_id
      WHERE product.price = (%s);''',
```

```
3: '''SELECT manufacturer.manufacturer_id, manufacturer_name, batch_date_to,
batch_date_from, batch_info,
      b.storage_id, buyer.buyer_name, buyer.phone FROM manufacturer
      JOIN batch as b ON b.manufacturer_id = manufacturer.manufacturer_id
      JOIN employee ON b.employee_id = employee.employee_id
      JOIN buyer ON employee.buyer_id = buyer.buyer_id
      WHERE manufacturer.manufacturer_name = (%s);''',
```

```
4: '''SELECT review.buyer_id, review.product_id, product_name, price,
product_type_name, review.rating,
```

```
      review.text, buyer_name  FROM review
      JOIN buyer ON review.buyer_id = buyer.buyer_id
      JOIN product ON review.product_id = product.product_id
      JOIN product_type AS pt ON product.product_type_id =
pt.product_type_id
      WHERE rating = (%s);''',
```

```
5: '''SELECT \"storage\".storage_id, batch_date_to, employee.employee_id,
buyer_name as employee_name,
       batch_info, manufacturer_name, product_name, quantity, price,
create_date FROM \"storage\"
       JOIN batch ON batch.storage_id = \"storage\".storage_id
       JOIN employee ON batch.employee_id = employee.employee_id
       JOIN buyer ON employee.buyer_id = buyer.buyer_id
       JOIN manufacturer ON batch.manufacturer_id =
manufacturer.manufacturer_id
       JOIN product_in_batch as pib ON pib.batch_id = batch.batch_id
       JOIN product ON pib.product_id=product.product_id
       WHERE quantity = (%s);'''
```

Эти запросы являются достаточно сложными для успешной последующей оптимизации, так как содержат условия, группировку, соединения и вложенные запросы. Также в них уже заданы поля для параметров, которые в программе нагрузки будут генерироваться случайным образом.

Полученные запросы будем использовать для нагрузки БД. Они будут задаваться двумя способами: в одном потоке некоторое количество раз подряд и в нескольких потоках одновременно.

Для получения статистики по выполнению запросов и ответу БД на эти действия используем оператор EXPLAIN ANALYZE. Из результатов будет доступно время на планирование и выполнения определенного запроса. Эти данные будут сохраняться, а затем использоваться для построения графиков зависимости усредненного времени выполнения одного запроса от количества заданных запросов и зависимости времени от количества потоков.

Для оптимизации были использован два способа: добавления индексов и сохранение плана запроса.

Создание индексов

Индексы — это традиционное средство увеличения производительности БД. Используя индекс, сервер баз данных может находить и извлекать нужные строки гораздо быстрее, чем без него.

PostgreSQL поддерживает несколько типов индексов: B-дерево, хеш, GiST, SP-GiST, GIN и BRIN. Для разных типов индексов применяются разные алгоритмы, ориентированные на определённые типы запросов. По умолчанию команда CREATE INDEX создаёт индексы типа B-дерево, эффективные в большинстве случаев.

B-деревья могут работать в условиях на равенство и в проверках диапазонов с данными, которые можно отсортировать в некотором порядке. Точнее, планировщик запросов PostgreSQL может задействовать индекс-B-дерево, когда индексируемый столбец участвует в сравнении с одним из следующих операторов:

- <
- <=
- =
- >=
- >

Добавление индексов:

```
def optimize_create_indices():
    ind_con = psycopg2.connect(dbname=config.get("postgres", "dbname"),
                               user=config.get("postgres", "user"),
                               password=config.get("postgres", "password"))
# открыть соединение, чтобы создать индексы
    ind_cur = ind_con.cursor()
    ind_cur.execute("CREATE INDEX IF NOT EXISTS i1 ON
\"order\"(order_date_to);")
    ind_cur.execute("CREATE INDEX IF NOT EXISTS i2 ON product(price);")
    ind_cur.execute("CREATE INDEX IF NOT EXISTS i3 ON
manufacturer(manufacturer_name);")
    ind_cur.execute("CREATE INDEX IF NOT EXISTS i4 ON review(rating);")
    ind_cur.execute("CREATE INDEX IF NOT EXISTS i5 ON
product_in_batch(quantity);")

    ind_con.commit()
    print('indices created')
    ind_con.close()
```

Индексы были добавлены для тех атрибутов, которые встречаются при операторах WHERE, JOIN ON, GROUP BY.

Подготовленный оператор представляет собой объект на стороне сервера, позволяющий оптимизировать производительность приложений. Когда выполняется PREPARE, указанный оператор разбирается, анализируется и переписывается. При последующем выполнении команды EXECUTE

подготовленный оператор планируется и исполняется. Такое разделение труда исключает повторный разбор запроса, при этом позволяет выбрать наилучший план выполнения в зависимости от определённых значений параметров. Подготовленные операторы существуют только в рамках текущего сеанса работы с БД.

Подготовленные операторы дают наибольший выигрыш в производительности, когда в одном сеансе выполняется большое число однотипных операторов. Отличие в производительности особенно значительно, если операторы достаточно сложны для планирования или перезаписи, например, когда в запросе объединяется множество таблиц или необходимо применить несколько правил.

До использования индексов:

План запроса 1:

	QUERY PLAN text
1	Nested Loop (cost=0.57..216.54 rows=2 width=68) (actual time=1.063..1.063 rows=0 loops=1)
2	-> Nested Loop (cost=0.29..215.61 rows=2 width=20) (actual time=1.062..1.062 rows=0 loops=1)
3	-> Seq Scan on "order" (cost=0.00..199.00 rows=2 width=12) (actual time=1.061..1.061 rows=0 loops=1)
4	Filter: (order_date_to = '2020-01-01'::date)
5	Rows Removed by Filter: 10000
6	-> Index Scan using employee_pk on employee (cost=0.29..8.30 rows=1 width=16) (never executed)
7	Index Cond: (employee_id = "order".employee_id)
8	-> Index Scan using buyer_pk on buyer (cost=0.29..0.46 rows=1 width=52) (never executed)
9	Index Cond: (buyer_id = employee.buyer_id)
10	Planning Time: 0.395 ms
11	Execution Time: 1.094 ms

Рис 18. План запроса

План запроса 2:

	QUERY PLAN text
1	Nested Loop (cost=248.94..407.30 rows=1 width=544) (actual time=1.525..1.525 rows=0 loops=1)
2	-> Nested Loop (cost=248.66..406.84 rows=1 width=332) (actual time=1.525..1.525 rows=0 loops=1)
3	-> Hash Join (cost=248.51..398.65 rows=1 width=265) (actual time=1.524..1.525 rows=0 loops=1)
4	Hash Cond: (ptp.product_id = product.product_id)
5	-> Seq Scan on product_to_product ptp (cost=0.00..137.00 rows=5000 width=8) (actual time=0.010..0.426 rows=5000 loops=1)
6	-> Hash (cost=248.50..248.50 rows=1 width=257) (actual time=0.728..0.728 rows=2 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Seq Scan on product (cost=0.00..248.50 rows=1 width=257) (actual time=0.048..0.723 rows=2 loops=1)
9	Filter: (price = 3000)
10	Rows Removed by Filter: 5000
11	-> Index Scan using product_type_pk on product_type pt (cost=0.15..8.17 rows=1 width=71) (never executed)
12	Index Cond: (product_type_id = product.product_type_id)
13	-> Index Scan using device_pk on product prod (cost=0.28..0.46 rows=1 width=216) (never executed)
14	Index Cond: (product_id = ptp.product_part_id)
15	Planning Time: 0.396 ms
16	Execution Time: 1.567 ms

Рис 19. План запроса

План запроса 3:

	QUERY PLAN text
1	Nested Loop (cost=1.98..291.76 rows=156 width=148) (actual time=1.252..47.506 rows=149 loops=1)
2	-> Nested Loop (cost=1.70..219.32 rows=156 width=129) (actual time=0.852..13.403 rows=149 loops=1)
3	-> Hash Join (cost=1.41..163.27 rows=156 width=129) (actual time=0.543..3.112 rows=149 loops=1)
4	Hash Cond: (b.manufacturer_id = manufacturer.manufacturer_id)
5	-> Seq Scan on batch b (cost=0.00..147.00 rows=5000 width=121) (actual time=0.147..2.622 rows=5000 loops=1)
6	-> Hash (cost=1.40..1.40 rows=1 width=12) (actual time=0.012..0.012 rows=1 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Seq Scan on manufacturer (cost=0.00..1.40 rows=1 width=12) (actual time=0.008..0.009 rows=1 loops=1)
9	Filter: ((manufacturer_name)::text = 'juul'::text)
10	Rows Removed by Filter: 31
11	-> Index Scan using employee_pk on employee (cost=0.29..0.36 rows=1 width=8) (actual time=0.068..0.068 rows=1 loops=149)
12	Index Cond: (employee_id = b.employee_id)
13	-> Index Scan using buyer_pk on buyer (cost=0.29..0.46 rows=1 width=27) (actual time=0.227..0.227 rows=1 loops=149)
14	Index Cond: (buyer_id = employee.buyer_id)
15	Planning Time: 5.131 ms
16	Execution Time: 47.569 ms

Рис 20. План запроса

План запроса 4:

	QUERY PLAN text
1	Hash Join (cost=414.12..1053.51 rows=269 width=479) (actual time=3.776..17.818 rows=269 loops=1)
2	Hash Cond: (product.product_type_id = pt.product_type_id)
3	-> Hash Join (cost=385.67..1024.36 rows=269 width=477) (actual time=3.751..17.700 rows=269 loops=1)
4	Hash Cond: (buyer.buyer_id = review.buyer_id)
5	-> Seq Scan on buyer (cost=0.00..511.00 rows=25000 width=16) (actual time=0.005..11.593 rows=25000 loops=1)
6	-> Hash (cost=382.30..382.30 rows=269 width=465) (actual time=3.737..3.737 rows=269 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 140kB
8	-> Hash Join (cost=106.11..382.30 rows=269 width=465) (actual time=2.493..3.630 rows=269 loops=1)
9	Hash Cond: (product.product_id = review.product_id)
10	-> Seq Scan on product (cost=0.00..236.00 rows=5000 width=53) (actual time=0.004..0.827 rows=5002 loops=1)
11	-> Hash (cost=102.75..102.75 rows=269 width=416) (actual time=2.475..2.475 rows=269 loops=1)
12	Buckets: 1024 Batches: 1 Memory Usage: 126kB
13	-> Seq Scan on review (cost=0.00..102.75 rows=269 width=416) (actual time=0.146..2.352 rows=269 loops=1)
14	Filter: (rating = 4)
15	Rows Removed by Filter: 1231
16	-> Hash (cost=18.20..18.20 rows=820 width=10) (actual time=0.020..0.020 rows=8 loops=1)
17	Buckets: 1024 Batches: 1 Memory Usage: 9kB
18	-> Seq Scan on product_type pt (cost=0.00..18.20 rows=820 width=10) (actual time=0.009..0.010 rows=8 loops=1)
19	Planning Time: 3.264 ms
20	Execution Time: 17.921 ms

Рис 21. План запроса

План запроса 5:

	QUERY PLAN text
1	Hash Join (cost=406.64..774.11 rows=65 width=186) (actual time=5.416..9.765 rows=65 loops=1)
2	Hash Cond: (batch.manufacturer_id = manufacturer.manufacturer_id)
3	-> Nested Loop (cost=404.92..772.20 rows=65 width=182) (actual time=5.375..9.691 rows=65 loops=1)
4	-> Nested Loop (cost=404.63..742.02 rows=65 width=174) (actual time=5.367..8.885 rows=65 loops=1)
5	-> Nested Loop (cost=404.35..718.66 rows=65 width=170) (actual time=5.358..8.404 rows=65 loops=1)
6	-> Hash Join (cost=404.06..697.13 rows=65 width=170) (actual time=5.328..6.335 rows=65 loops=1)
7	Hash Cond: (pib.batch_id = batch.batch_id)
8	-> Hash Join (cost=194.56..487.46 rows=65 width=57) (actual time=3.116..4.078 rows=65 loops=1)
9	Hash Cond: (product.product_id = pib.product_id)
10	-> Seq Scan on product (cost=0.00..236.00 rows=5000 width=49) (actual time=0.011..0.727 rows=5002 loops=1)
11	-> Hash (cost=193.75..193.75 rows=65 width=16) (actual time=3.058..3.058 rows=65 loops=1)
12	Buckets: 1024 Batches: 1 Memory Usage: 12kB
13	-> Seq Scan on product_in_batch pib (cost=0.00..193.75 rows=65 width=16) (actual time=0.320..3.028 rows=65 loops=1)
14	Filter: (quantity = 50)
15	Rows Removed by Filter: 7435
16	-> Hash (cost=147.00..147.00 rows=5000 width=121) (actual time=2.179..2.180 rows=5000 loops=1)
17	Buckets: 8192 Batches: 1 Memory Usage: 826kB
18	-> Seq Scan on batch (cost=0.00..147.00 rows=5000 width=121) (actual time=0.008..0.967 rows=5000 loops=1)
19	-> Index Only Scan using storage_pk on storage (cost=0.28..0.33 rows=1 width=4) (actual time=0.031..0.031 rows=1 loops=65)
20	Index Cond: (storage_id = batch.storage_id)
21	Heap Fetches: 65
22	-> Index Scan using employee_pk on employee (cost=0.29..0.36 rows=1 width=8) (actual time=0.007..0.007 rows=1 loops=65)
23	Index Cond: (employee_id = batch.employee_id)
24	-> Index Scan using buyer_pk on buyer (cost=0.29..0.46 rows=1 width=16) (actual time=0.012..0.012 rows=1 loops=65)
25	Index Cond: (buyer_id = employee.buyer_id)
26	-> Hash (cost=1.32..1.32 rows=32 width=12) (actual time=0.031..0.031 rows=32 loops=1)
27	Buckets: 1024 Batches: 1 Memory Usage: 10kB
28	-> Seq Scan on manufacturer (cost=0.00..1.32 rows=32 width=12) (actual time=0.016..0.019 rows=32 loops=1)
29	Planning Time: 6.829 ms
30	Execution Time: 9.974 ms

Рис 22. План запроса

После использования индексов:

План запроса 1:

	QUERY PLAN text
1	Nested Loop (cost=4.87..28.87 rows=2 width=68) (actual time=0.043..0.043 rows=0 loops=1)
2	-> Nested Loop (cost=4.59..27.95 rows=2 width=20) (actual time=0.042..0.042 rows=0 loops=1)
3	-> Bitmap Heap Scan on "order" (cost=4.30..11.34 rows=2 width=12) (actual time=0.042..0.042 rows=0 loops=1)
4	Recheck Cond: (order_date_to = '2020-01-01'::date)
5	-> Bitmap Index Scan on i1 (cost=0.00..4.30 rows=2 width=0) (actual time=0.041..0.041 rows=0 loops=1)
6	Index Cond: (order_date_to = '2020-01-01'::date)
7	-> Index Scan using employee_pk on employee (cost=0.29..8.30 rows=1 width=16) (never executed)
8	Index Cond: (employee_id = "order".employee_id)
9	-> Index Scan using buyer_pk on buyer (cost=0.29..0.46 rows=1 width=52) (never executed)
10	Index Cond: (buyer_id = employee.buyer_id)
11	Planning Time: 4.843 ms
12	Execution Time: 0.080 ms

Рис 23. План запроса

План запроса 2:

	QUERY PLAN text
1	Nested Loop (cost=8.75..167.10 rows=1 width=544) (actual time=1.065..1.065 rows=0 loops=1)
2	-> Nested Loop (cost=8.46..166.64 rows=1 width=332) (actual time=1.064..1.064 rows=0 loops=1)
3	-> Hash Join (cost=8.31..158.45 rows=1 width=265) (actual time=1.064..1.064 rows=0 loops=1)
4	Hash Cond: (ptp.product_id = product.product_id)
5	-> Seq Scan on product_to_product ptp (cost=0.00..137.00 rows=5000 width=8) (actual time=0.015..0.634 rows=5000 loops=1)
6	-> Hash (cost=8.30..8.30 rows=1 width=257) (actual time=0.054..0.054 rows=2 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Index Scan using i2 on product (cost=0.28..8.30 rows=1 width=257) (actual time=0.048..0.050 rows=2 loops=1)
9	Index Cond: (price = 3000)
10	-> Index Scan using product_type_pk on product_type pt (cost=0.15..8.17 rows=1 width=71) (never executed)
11	Index Cond: (product_type_id = product.product_type_id)
12	-> Index Scan using device_pk on product prod (cost=0.28..0.46 rows=1 width=216) (never executed)
13	Index Cond: (product_id = ptp.product_part_id)
14	Planning Time: 3.928 ms
15	Execution Time: 1.104 ms

Рис 24. План запроса

План запроса 3:

	QUERY PLAN text
1	Nested Loop (cost=1.98..291.76 rows=156 width=148) (actual time=0.096..2.540 rows=149 loops=1)
2	-> Nested Loop (cost=1.70..219.32 rows=156 width=129) (actual time=0.090..2.049 rows=149 loops=1)
3	-> Hash Join (cost=1.41..163.27 rows=156 width=129) (actual time=0.082..1.620 rows=149 loops=1)
4	Hash Cond: (b.manufacturer_id = manufacturer.manufacturer_id)
5	-> Seq Scan on batch b (cost=0.00..147.00 rows=5000 width=121) (actual time=0.017..1.165 rows=5000 loops=1)
6	-> Hash (cost=1.40..1.40 rows=1 width=12) (actual time=0.015..0.015 rows=1 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Seq Scan on manufacturer (cost=0.00..1.40 rows=1 width=12) (actual time=0.010..0.013 rows=1 loops=1)
9	Filter: ((manufacturer_name)::text = 'juul'::text)
10	Rows Removed by Filter: 31
11	-> Index Scan using employee_pk on employee (cost=0.29..0.36 rows=1 width=8) (actual time=0.002..0.002 rows=1 loops=149)
12	Index Cond: (employee_id = b.employee_id)
13	-> Index Scan using buyer_pk on buyer (cost=0.29..0.46 rows=1 width=27) (actual time=0.003..0.003 rows=1 loops=149)
14	Index Cond: (buyer_id = employee.buyer_id)
15	Planning Time: 0.714 ms
16	Execution Time: 2.602 ms

Рис 25. План запроса

План запроса 4:

	QUERY PLAN text
1	Hash Join (cost=409.12..1048.52 rows=269 width=479) (actual time=2.805..6.410 rows=269 loops=1)
2	Hash Cond: (product.product_type_id = pt.product_type_id)
3	-> Hash Join (cost=380.67..1019.36 rows=269 width=477) (actual time=2.785..6.336 rows=269 loops=1)
4	Hash Cond: (buyer.buyer_id = review.buyer_id)
5	-> Seq Scan on buyer (cost=0.00..511.00 rows=25000 width=16) (actual time=0.009..1.883 rows=25000 loops=1)
6	-> Hash (cost=377.31..377.31 rows=269 width=465) (actual time=2.767..2.767 rows=269 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 140kB
8	-> Hash Join (cost=101.09..377.31 rows=269 width=465) (actual time=0.601..2.625 rows=269 loops=1)
9	Hash Cond: (product.product_id = review.product_id)
10	-> Seq Scan on product (cost=0.00..236.02 rows=5002 width=53) (actual time=0.007..1.601 rows=5002 loops=1)
11	-> Hash (cost=97.72..97.72 rows=269 width=416) (actual time=0.577..0.577 rows=269 loops=1)
12	Buckets: 1024 Batches: 1 Memory Usage: 126kB
13	-> Bitmap Heap Scan on review (cost=10.36..97.72 rows=269 width=416) (actual time=0.081..0.424 rows=269 loops=1)
14	Recheck Cond: (rating = 4)
15	Heap Blocks: exact=84
16	-> Bitmap Index Scan on i4 (cost=0.00..10.29 rows=269 width=0) (actual time=0.068..0.068 rows=269 loops=1)
17	Index Cond: (rating = 4)
18	-> Hash (cost=18.20..18.20 rows=820 width=10) (actual time=0.013..0.013 rows=8 loops=1)
19	Buckets: 1024 Batches: 1 Memory Usage: 9kB
20	-> Seq Scan on product_type pt (cost=0.00..18.20 rows=820 width=10) (actual time=0.006..0.008 rows=8 loops=1)
21	Planning Time: 2.538 ms
22	Execution Time: 6.531 ms

Рис 26. План запроса

План запроса 5:

	QUERY PLAN text
1	Hash Join (cost=312.42..679.94 rows=65 width=186) (actual time=2.753..4.506 rows=65 loops=1)
2	Hash Cond: (batch.manufacturer_id = manufacturer.manufacturer_id)
3	-> Nested Loop (cost=310.70..678.02 rows=65 width=182) (actual time=2.717..4.444 rows=65 loops=1)
4	-> Nested Loop (cost=310.41..647.84 rows=65 width=174) (actual time=2.702..4.234 rows=65 loops=1)
5	-> Nested Loop (cost=310.13..624.49 rows=65 width=170) (actual time=2.690..4.044 rows=65 loops=1)
6	-> Hash Join (cost=309.85..602.96 rows=65 width=170) (actual time=2.674..3.782 rows=65 loops=1)
7	Hash Cond: (pib.batch_id = batch.batch_id)
8	-> Hash Join (cost=100.35..393.29 rows=65 width=57) (actual time=0.401..1.486 rows=65 loops=1)
9	Hash Cond: (product.product_id = pib.product_id)
10	-> Seq Scan on product (cost=0.00..236.02 rows=5002 width=49) (actual time=0.008..0.819 rows=5002 loops=1)
11	-> Hash (cost=99.53..99.53 rows=65 width=16) (actual time=0.337..0.337 rows=65 loops=1)
12	Buckets: 1024 Batches: 1 Memory Usage: 12kB
13	-> Bitmap Heap Scan on product_in_batch pib (cost=4.79..99.53 rows=65 width=16) (actual time=0.087..0.318 rows=65 loops=1)
14	Recheck Cond: (quantity = 50)
15	Heap Blocks: exact=50
16	-> Bitmap Index Scan on i5 (cost=0.00..4.77 rows=65 width=0) (actual time=0.075..0.075 rows=65 loops=1)
17	Index Cond: (quantity = 50)
18	-> Hash (cost=147.00..147.00 rows=5000 width=121) (actual time=2.223..2.223 rows=5000 loops=1)
19	Buckets: 8192 Batches: 1 Memory Usage: 826kB
20	-> Seq Scan on batch (cost=0.00..147.00 rows=5000 width=121) (actual time=0.014..1.015 rows=5000 loops=1)
21	-> Index Only Scan using storage_pk on storage (cost=0.28..0.33 rows=1 width=4) (actual time=0.004..0.004 rows=1 loops=65)
22	Index Cond: (storage_id = batch.storage_id)
23	Heap Fetches: 65
24	-> Index Scan using employee_pk on employee (cost=0.29..0.36 rows=1 width=8) (actual time=0.002..0.002 rows=1 loops=65)
25	Index Cond: (employee_id = batch.employee_id)
26	-> Index Scan using buyer_pk on buyer (cost=0.29..0.46 rows=1 width=16) (actual time=0.003..0.003 rows=1 loops=65)
27	Index Cond: (buyer_id = employee.buyer_id)
28	-> Hash (cost=1.32..1.32 rows=32 width=12) (actual time=0.026..0.026 rows=32 loops=1)
29	Buckets: 1024 Batches: 1 Memory Usage: 10kB
30	-> Seq Scan on manufacturer (cost=0.00..1.32 rows=32 width=12) (actual time=0.010..0.014 rows=32 loops=1)
31	Planning Time: 4.925 ms
32	Execution Time: 4.751 ms

Рис 27. План запроса

Графики:

Для константного количества потоков (1):

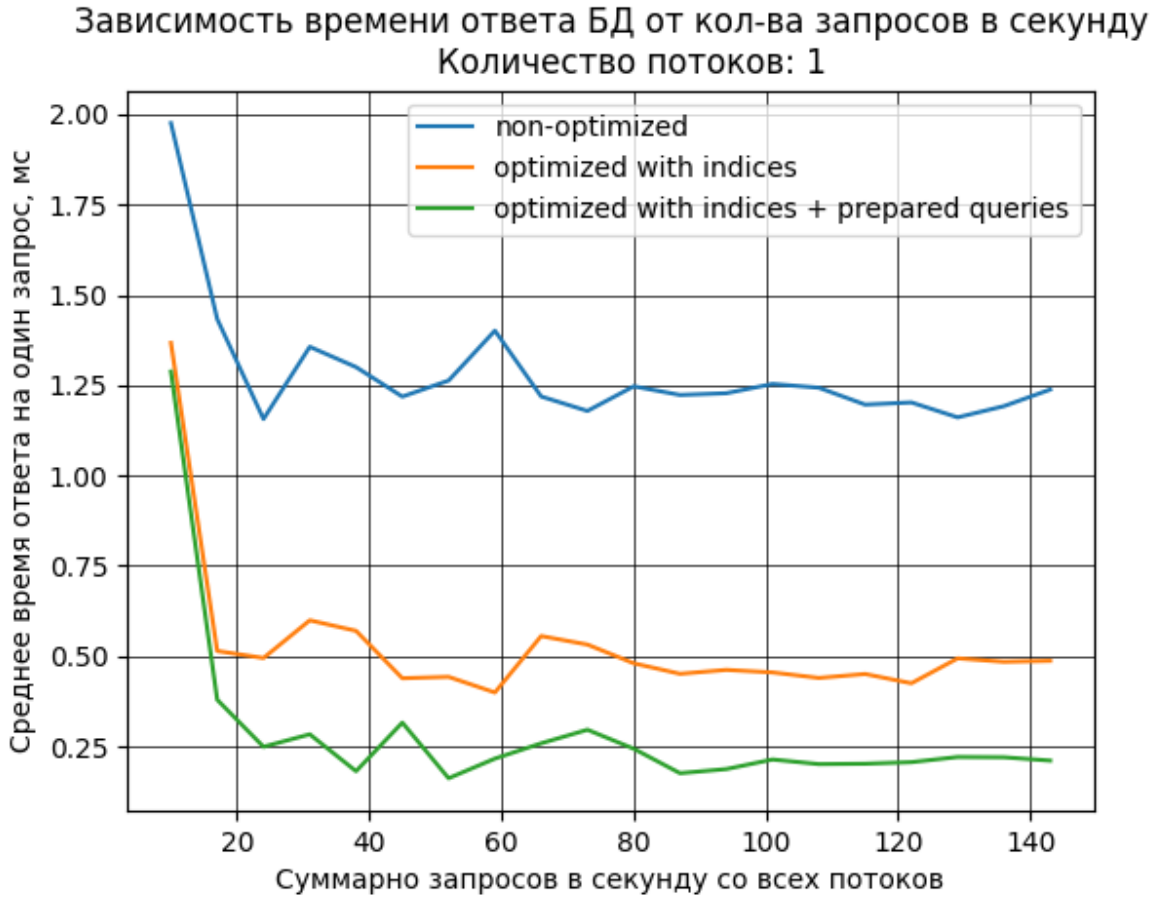


Рис 28. Эксперимент 1.

Для константного количества потоков (3):

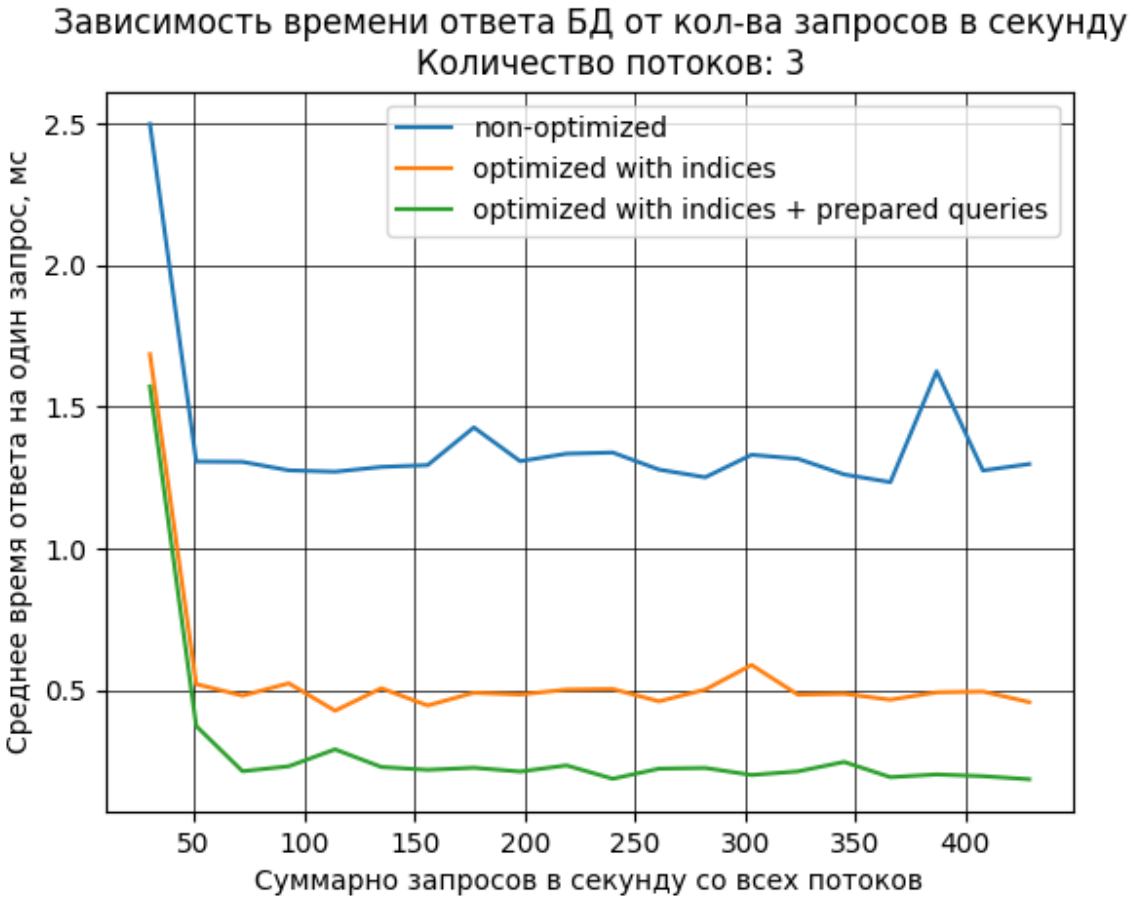


Рис 29. Эксперимент 2.

Для динамического количества потоков (от 1 до 10):

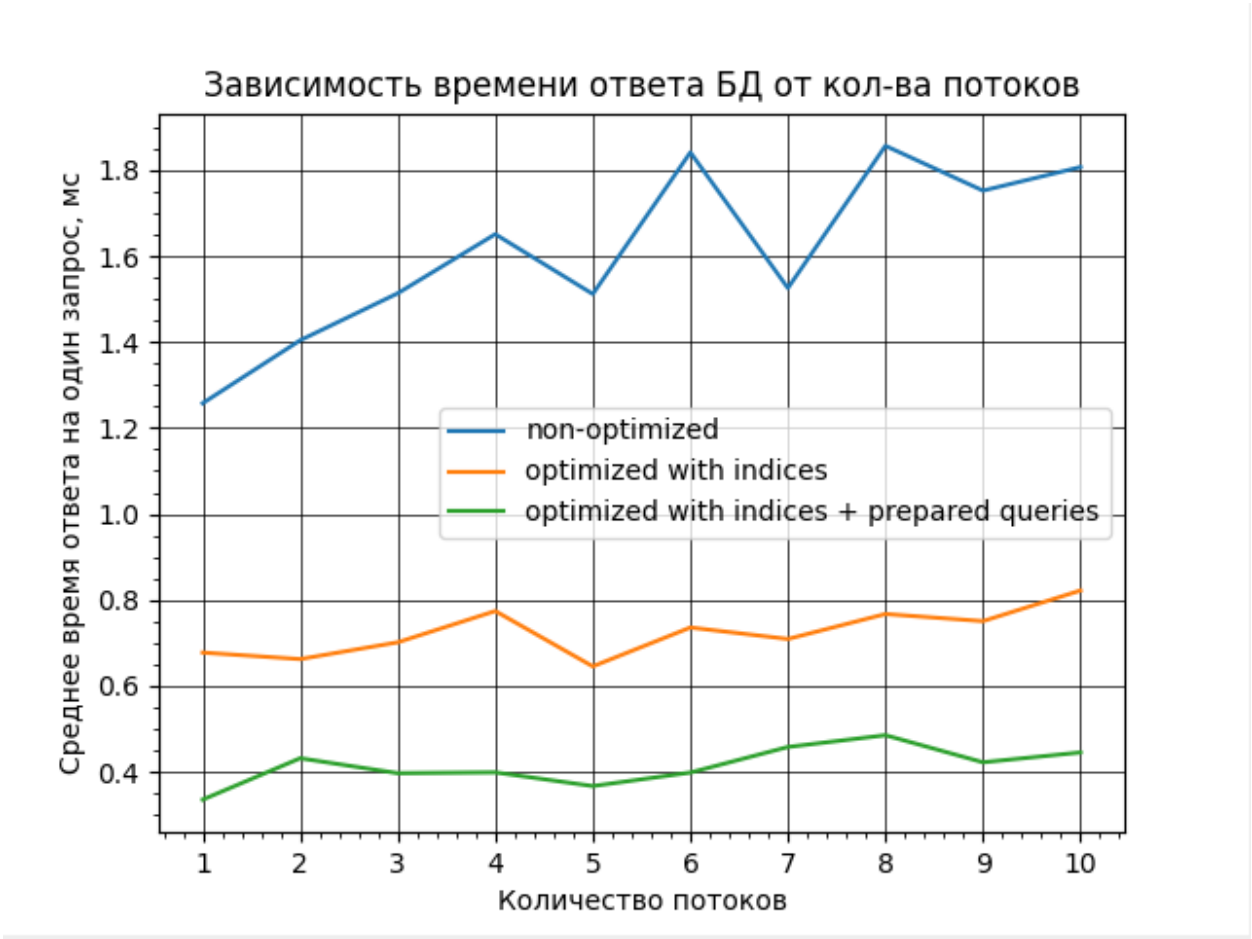


Рис 30. Эксперимент 3.

В ходе проведения экспериментов было обнаружено, что при очень больших количествах записей в базе данных (более 100 тысяч на таблицу) использование подготовленных операторов перестает давать прибавки в производительности. Это происходит потому, что на подготовку запроса тратится очень большое количество времени, большее чем требуется на исполнение запроса. Отсюда мы и не получаем прибавку в производительности.

С другой стороны, при относительно малом количестве записей в БД – меньше 1000 на таблицу – практически теряет смысл использование индексации, поскольку сканирование индексов только увеличивает время работы запроса, не помогая, в отличие от других случаев, ускорить фильтрацию большого объема данных.

Также, стоит отметить, что на результаты экспериментов влияет не только количество данных в базе и параметры запуска, но и разброс при случайном выборе выполняемого запроса, а также фоновые процессы на машине, на которой проводились эксперименты.

Итог.

Было разработано приложение, код находится по ссылке https://gitlab.icc.spbstu.ru/maynekeen/vape-shop_PostgreSQL/-/blob/master/lab4/main.py.
Для его работы используется модуль config.py для подключения к базе данных, такой же, как в лабораторной №2.
Приложение умеет работать в двух режимах: построение зависимости времени ответа БД от количества запросов в секунду (далее первый) и построение зависимости времени ответа БД от количества соединений, выполняющих запросы (далее второй). В каждом режиме сначала зависимость снимается на неоптимизированных запросах, затем создаются индексы, затем подготавливаются операторы. Все три зависимости выводятся на одном графике.
В первом режиме пользователю нужно задать количество потоков, осуществляющий запросы к базе данных, минимальное и максимальное количество запросов в секунду на один поток, а также время моделирования в секундах. Промежуток от минимального до максимального кол-ва запросов в секунду делится на количество секунд, далее потоки пытаются выполнить переданное им количество запросов, со временем наращивая нагрузку вплоть до максимального количества запросов. Выполнение заданного количества запросов не гарантируется и ограничено вычислительными мощностями машины. На каждом шаге производится подсчёт среднего времени ответа БД и общего количества запросов, осуществлённых со всех потоков.
Во втором режиме пользователю необходимо задать минимальное и максимальное кол-во потоков, а также кол-во запросов на каждый поток. На каждой итерации работы приложения количество соединений увеличивается на 1 и проходит все значения от минимального до максимального количества потоков. Каждый поток выполняется заданное количество запросов (без ограничений по времени). На каждом шаге производится подсчёт среднего времени ответа БД.