****

**­**

**Министр науки и высшего образования Российской̆**

**Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

Факультет информационных технологий и программирования

Рабочий протокол и отчет по лабораторной работе № 4

Маскировка и анонимизация данных

**Выполнил(а) студент группы** № **M34091**

Сидорцов Владимир Сергеевич

**Подпись:**

**Проверил:**

Санкт-Петербург

2023г.

1. **Цель работы.**

* Замаскировать поля с конфиденциальными данными.
* Провести анонимизацию данных.

1. **Задачи, решаемые при выполнении работы.**

* Создание контейнера
* Backup
* Restore
* Анонимизация данных
* Генерализация данных

1. **Объект исследования.**

* [postgresql anonymizer](https://postgresql-anonymizer.readthedocs.io/en/stable/INSTALL/)
* [docker](https://www.docker.com/get-started/)

1. **Выполнение работы.**

Для начала необходимо установить [postgresql anonymizer](https://postgresql-anonymizer.readthedocs.io/en/stable/INSTALL/). Сразу же сталкиваемся с проблемой: «WE DO NOT PROVIDE COMMUNITY SUPPORT FOR THIS EXTENSION ON WINDOWS.»

Следовательно, остаётся работать через [контейнер Docker](https://postgresql-anonymizer.readthedocs.io/en/stable/INSTALL/#install-with-docker), который будет работать как отдельная база данных.

Скачиваем образ контейнера последней стабильной (долгая поддержка) версии:

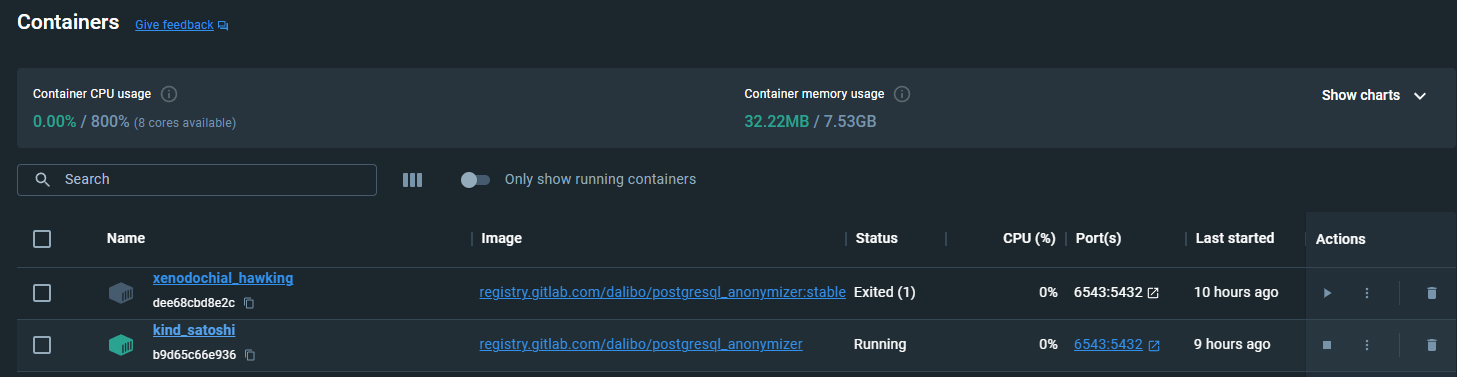
docker pull registry.gitlab.com/dalibo/postgresql\_anonymizer:stable

И запускаем этот контейнер, устанавливая необходимую конфигурацию:

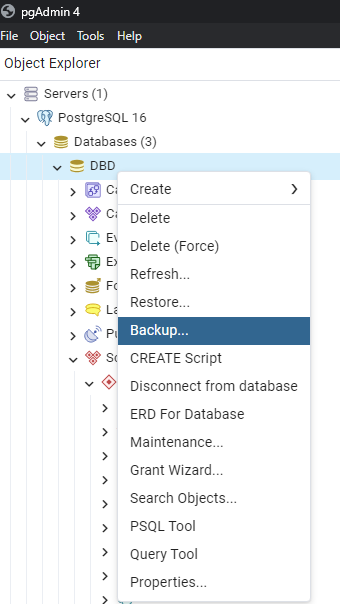
docker run -d -e POSTGRES\_PASSWORD=x -p 6543:5432 registry.gitlab.com/dalibo/postgresql\_anonymizer:stable

После чего мы теперь можем подключиться к ней как к обычной базе данных.  
Server: localhost  
Database: postgres  
Port: 6543  
Username: postgres  
Пароль пользователя postgres: x

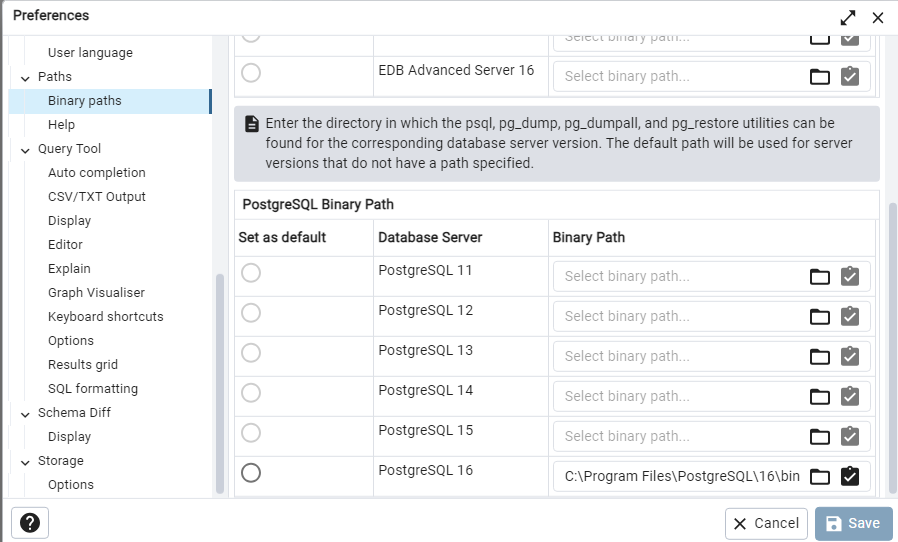
Видим его в Docker:



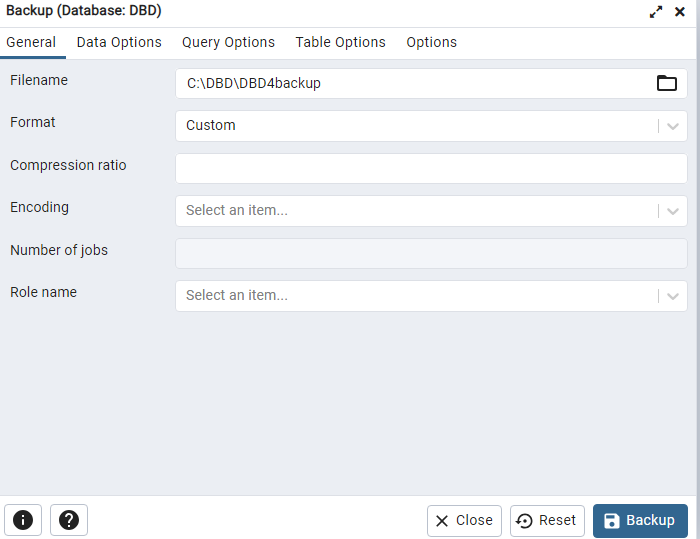
Теперь нам нужно перенести все существующие данные, ведь у нас теперь контейнер с отдельной бд. Для этого воспользуемся backup’ом базы данных:



(Если pgAdmin не видит путь до бинарников, то он попросит вас явно указать до них путь, ведь там находятся утилиты, одну из которых вы хотите вызвать):



Указав имя файла бэкапа бд, создаём его:



Мы выгрузили данные в отдельный backup, теперь нобходимо загрузить его в контейнер. Для этого воспользуемся утилитами docker, используя Powerchell на хосте:

PS C:\Windows\system32> docker cp C:\DBD\DBD4backup.sql xenodochial\_hawking:/home/lab4backup

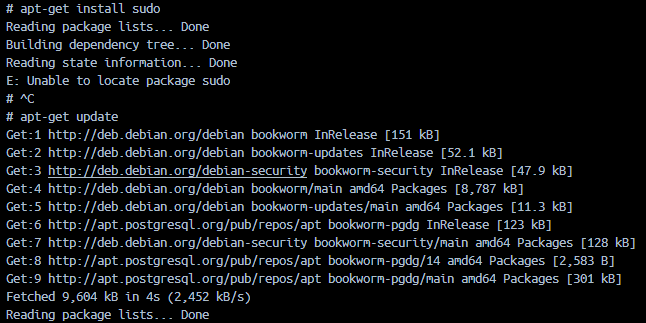
Successfully copied 12.3MB to kind\_satoshi:/home/lab4backup

Здесь мы сказали: скопируй файл бэкапа из C:\DBD\DBD4backup.sql в корневую директорию /home в файл /lab4backup контейнера под названием kind\_satoshi.

Теперь данные есть в файловой системе контейнера, необходимо только их загрузить в бд. Но… Самой [БД](https://www.postgresql.org/download/linux/debian/) еще в контейнере нет, так что необходимо ее [скачать](https://timeweb.cloud/tutorials/postgresql/obnovlenie-versii-postgresql):

Сейчас у нас в образе нет кэша пакетов, поэтому мы берем список пакетов последних версий в нашем дистрибутиве с помощью команды:  
apt-get update

apt-get upgrade при этом [запускать не следует](https://github.com/docker-library/postgres/issues/415#issuecomment-467997249).



Хотим выполнить [команду](#Скачивание_поста_для_дебиана), но у нас нет утилит sudo и lsb\_release, так что скачиваем их:  
apt-get -y install sudo  
apt-get -y install lsb-release

Теперь настраиваем репозиторий PostgreSQL для debian:  
sudo sh -c 'echo "deb https://apt.postgresql.org/pub/repos/apt $(lsb\_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'

Импортируем ключ подписи:   
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add –

Снова бновляем перечень пакетов в системе:  
sudo apt-get update

Устанавливаем нужную версию СУБД:   
sudo apt-get -y install postgresql

Отлично! Теперь у нас есть БД, мы готовы импортировать туда данные:

pg\_restore -U postgres -d postgres /home/lab4backup

Теперь необходимо создать роль, для которой нам нужно будет скрывать данные:

Активируем [механизм динамической маскировки](https://postgresql-anonymizer.readthedocs.io/en/stable/dynamic_masking/):

CREATE EXTENSION IF NOT EXISTS anon CASCADE;  
SELECT anon.*start\_dynamic\_masking*();

Создали роль зла:

CREATE ROLE evil LOGIN;  
SECURITY LABEL FOR anon ON ROLE evil IS 'MASKED';

И создадим для нее правила маскировки:

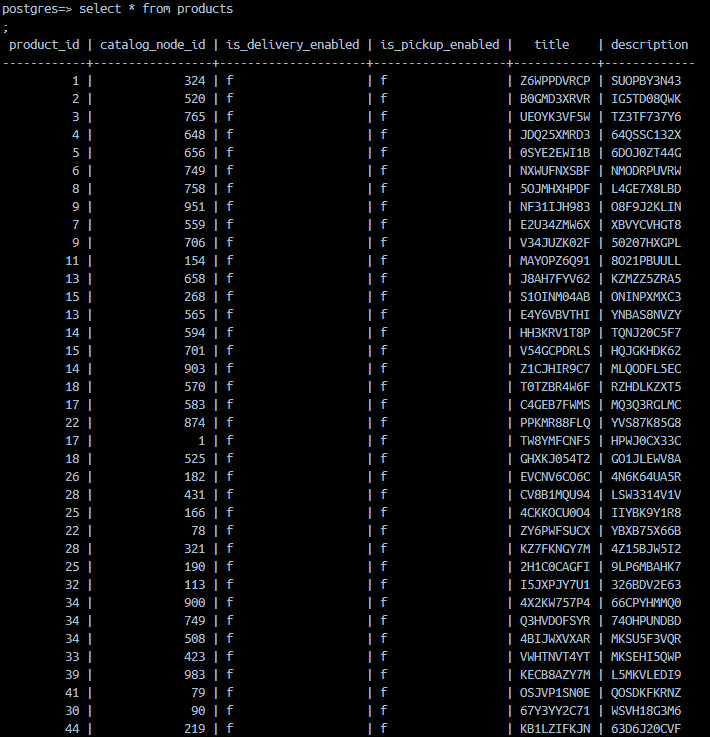
SECURITY LABEL FOR anon ON COLUMN products.title  
IS 'MASKED WITH FUNCTION anon.random\_string(10)';  
  
SECURITY LABEL FOR anon ON COLUMN products.description  
IS 'MASKED WITH FUNCTION anon.random\_string(10)';  
  
SECURITY LABEL FOR anon ON COLUMN products.is\_delivery\_enabled  
IS 'MASKED WITH VALUE ''false'' ';  
  
SECURITY LABEL FOR anon ON COLUMN products.is\_pickup\_enabled  
IS 'MASKED WITH VALUE ''false'' ';  
  
SECURITY LABEL FOR anon ON COLUMN products.product\_id  
IS 'MASKED WITH FUNCTION anon.noise(products.product\_id,0.2) ';  
  
SECURITY LABEL FOR anon ON COLUMN products.product\_id  
IS 'MASKED WITH FUNCTION anon.noise(products.product\_id,0.2) ';

Превосходно!  
Посмотрим результаты в базе данных:

Входим в нашу бд:  
psql -U postgres -h 0.0.0.0 -d postgres

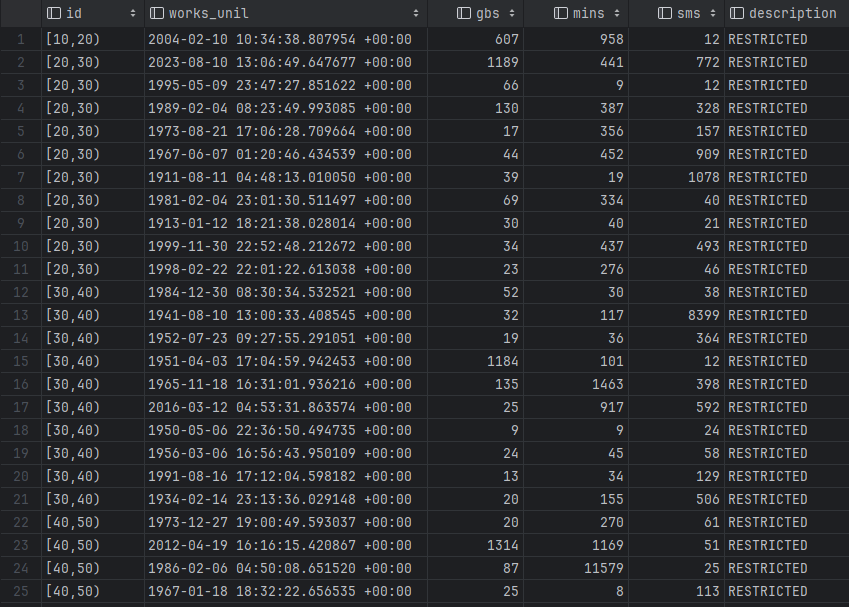
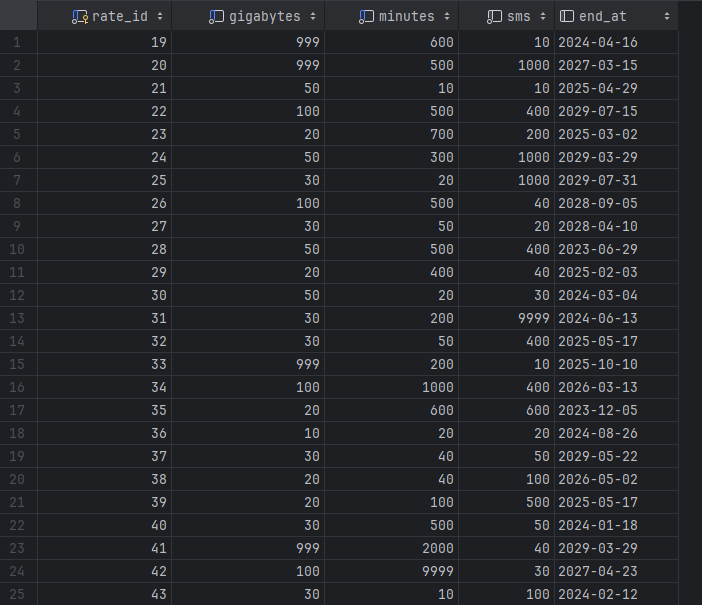
Входим внутри бд в роль зла:  
postgres=# \c – evil

Получаем все записи продуктов от этой роли:  
postgres=> select \* from products



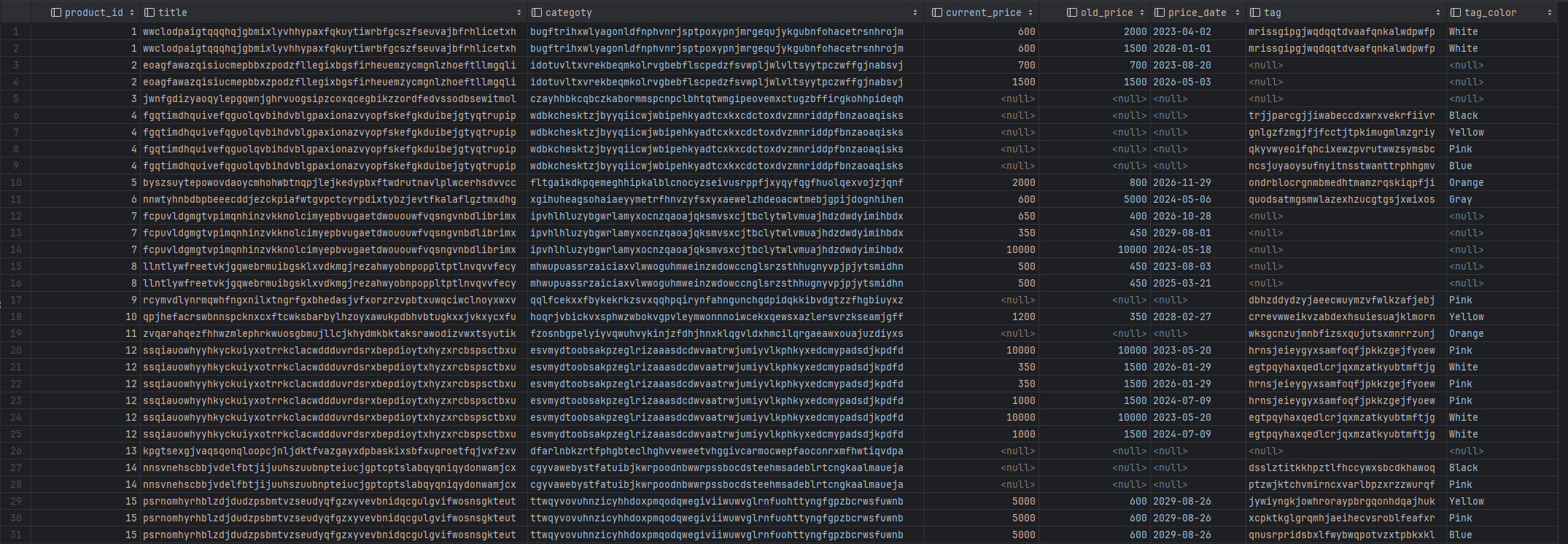
Теперь создадим материализованные представления, где используем различные [стратегии скрытия информации](https://postgresql-anonymizer.readthedocs.io/en/latest/masking_functions/):

CREATE MATERIALIZED VIEW evil\_rates AS  
 SELECT  
 anon.*generalize\_int4range*(r.rate\_id, 10) AS id,   
 anon.*random\_date*() as works\_unil,   
 anon.*noise*(r.gigabytes, 0.4) as gbs,  
 anon.*noise*(r.minutes, 0.6) as mins,  
 anon.*noise*(r.sms, 0.3) as sms,  
 -- Decay.  
 'RESTRICTED'::text AS description  
FROM postgres.public.rates AS r  
;



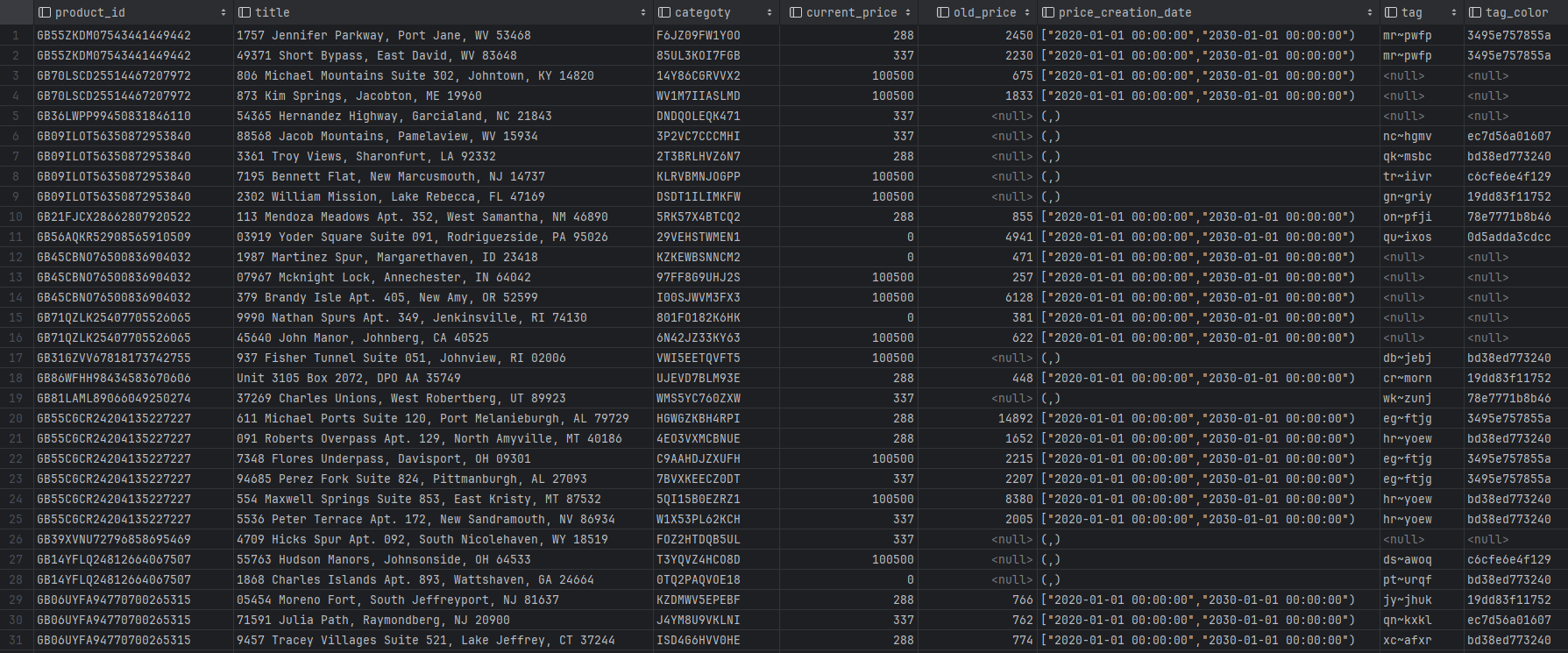
Создадим второе представление!  
Выведем интересующие данные:

SELECT  
 p.product\_id AS product\_id,  
 p.title AS title,  
 cn.title AS categoty,  
 pp.current\_price AS current\_price,  
 pp.old\_price AS old\_price,  
 pp.date AS price\_date,  
 t.title AS tag,  
 t.color AS tag\_color  
 FROM public.products AS p  
 LEFT JOIN public.product\_price pp on p.product\_id = pp.prodict\_id  
 LEFT JOIN public.catalog\_nodes cn on p.catalog\_node\_id = cn.catalog\_node\_id  
 LEFT JOIN public.tags\_products tp on p.product\_id = tp.product\_id  
 LEFT JOIN public.tags t on t.tag\_id = tp.tag\_id  
ORDER BY product\_id



Теперь создадим представление:

SELECT anon.*init*();  
CREATE MATERIALIZED VIEW evil\_products\_XD AS  
 SELECT  
 anon.*pseudo\_iban*(p.product\_id ,'salt') AS product\_id,  
 anon.*fake\_address*() AS title,  
 anon.*random\_string*(12) AS categoty,  
 anon.*random\_in*(ARRAY[288, 337, 100500, 0]) AS current\_price,  
 anon.*noise*(pp.old\_price,0.5) AS old\_price,  
 anon.*generalize\_tsrange*(pp.date,'decade') AS price\_creation\_date,  
 anon.*partial*(t.title,2,'~',4) AS tag,  
 pg\_catalog.*left*(anon.*hash*(t.color),12) AS tag\_color  
 FROM public.products AS p  
 LEFT JOIN public.product\_price pp on p.product\_id = pp.prodict\_id  
 LEFT JOIN public.catalog\_nodes cn on p.catalog\_node\_id = cn.catalog\_node\_id  
 LEFT JOIN public.tags\_products tp on p.product\_id = tp.product\_id  
 LEFT JOIN public.tags t on t.tag\_id = tp.tag\_id  
 ORDER BY p.product\_id  
;



1. **Выводы и анализ результатов работы.**

Научился устанавливать PostgreSQL Anonymizer  
Научился маскировать данные несколькими стратегиями.

1. **Дополнительные задания.**
2. **Выполнение дополнительных заданий.**
3. **Замечания преподавателя (исправления, вызванные замечаниями преподавателя, также помещают в этот пункт)**