

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего  
образования

Санкт-Петербургский национальный исследовательский университет ИТМО

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

**Лабораторная работа 4. Маскировка и анонимизация данных**

По дисциплине «Проектирование баз данных»

Выполнил:

студент группы №М3212

*Тимофеев Вячеслав* [REDACTED]

Проверила:

*Чеботарева* [REDACTED]



**УНИВЕРСИТЕТ ИТМО**

Санкт-Петербург  
2025

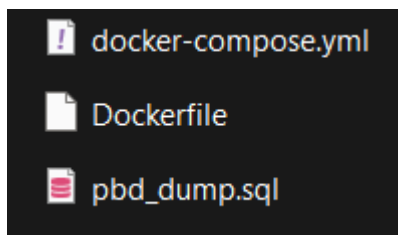
## Задачи:

1. Замаскировать поля с конфиденциальными данными.
2. Провести анонимизацию данных.

## Ход работы:

1. tl;dr: Для работы с PostgreSQL Anonymizer установил расширение ( [https://gitlab.com/dalibo/postgresql\\_anonymizer.git](https://gitlab.com/dalibo/postgresql_anonymizer.git) ) в ядре Linux-контейнера Docker.

Структура контейнера:



### Dockerfile

Устанавливаем зависимости для сборки расширений:

```
RUN apt-get update && apt-get install -y postgresql-server-dev-17 make gcc git
```

Клонируем репозиторий PostgreSQL Anonymizer от Neon Database:

```
RUN git clone https://github.com/neondatabase/postgresql_anonymizer.git /anon \
    && cd /anon && make && make install
```

### docker-compose.yml

Билдим контейнер в текущей директории (build: . ), заведомо закинув в неё Dockerfile;

Привязываем локальный порт 5433 с Windows на локальный порт 5432 поднятого внутри контейнера Linux + PostgreSQL;

Указываем volumes для хранения данных контейнера на хостовой машине (чтобы избежать потери при пересоздании контейнера);

Рестартим контейнер всегда при запуске, остановка при явном указании (restart: unless-stopped).

```

2 anonymizer_db:
3   build: .
4   container_name: anonymizer_db
5   ports:
6     - "5433:5432"
7   environment:
8     POSTGRES_USER: postgres
9     POSTGRES_PASSWORD: 
10    POSTGRES_DB: pbd
11   volumes:
12     - pg_anonymizer_data:/var/lib/postgresql/data
13   restart: unless-stopped

```

Билдим контейнер:

```

PS \pg-anonymizer> docker compose build
[+] Building 105.7s (8/8) FINISHED
docker:desktop-linux

```

(Далее для запуска: `docker compose up -d` )

Проверяем, что запросы перенаправляются внутрь контейнера на корретный порт (с PostgreSQL):

```

PS \pg-anonymizer> docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
171fd	24961d	"docker-entrypoint.s..."			0.0.0.0:5433->5432/tcp	anonymizer_db

Проверяем обмен пакетов между контейнером и хостом:

```

PS \pg-anonymizer> ping 192.

```

Обмен пакетами с 192. по с 32 байтами данньк:

Ответ от 192. : число байт=32 время<1мс TTL=128

Ответ от 192. : число байт=32 время<1мс TTL=128

Ответ от 192. : число байт=32 время<1мс TTL=128

Ответ от 192. : число байт=32 время<1мс TTL=128

Статистика Ping для 192. :

Пакетов: отправлено = 4, получено = 4, потеряно = 0 (0% потерь)

Приблизительное время приема-передачи в мс:

Минимальное = 0мсек, Максимальное = 0 мсек, Среднее = 0 мсек

Заходим внутрь контейнера и устанавливаем расширение **anon** в загруженные на этапе запуска сервера:

```

PS \pg-anonymizer> docker exec -it anonymizer_db bash
root@0ed9 :/# echo "shared_preload_libraries = 'anon'" >> /var/lib/postgresql/data/postgresql.conf

```

Подключаемся к базе с помощью psql (без явного пароля, так как на хосте в **pg\_hba.conf** вход по методу **scram-sha-256**) и проверяем наличие расширения в списке предварительных:

```

root@0ed9 :/# psql -U postgres -d pbd
psql (17.4 (Debian 17.4-1.pgdg120+2))
Type "help" for help.

pbd=# SHOW shared_preload_libraries;
      shared_preload_libraries
-----
anon
(1 row)

```

Перезагружаем контейнер:

```

pbd=# \q
root@0ed9 :/# exit
exit
PS \pg-anonymizer> docker restart anonymizer_db
anonymizer_db

```

Устанавливаем расширение и проверяем его в списке установленных:

```

pbd=# CREATE EXTENSION IF NOT EXISTS anon CASCADE;

CREATE EXTENSION
pbd=# \dx

```

List of installed extensions			
Name	Version	Schema	Description
anon	1.2.0	public	Data anonymization tools
pgcrypto	1.3	public	cryptographic functions
plpgsql	1.0	pg_catalog	PL/pgSQL procedural language

(3 rows)

(**pgcrypto** – вспомогательное расширение для работы функций генерирования случайных значений и хеширования)

Делаем дамп локальной базы данных, поднятой на Windows:

```

PS \pg-anonymizer> pg_dump -h localhost -p 5432 -U postgres -d pbd > pbd_dump.sql

```

Копируем дамп внутрь контейнера и восстанавливаем там:

```

PS \pg-anonymizer> docker cp pbd_dump.sql anonymizer_db:/pbd_dump.sql
Successfully copied 12 kB to anonymizer_db:/pbd_dump.sql
PS \pg-anonymizer> docker exec -it anonymizer_db psql -U postgres -d pbd -f /pbd_dump.sql

```

Создаю новый сервер в pgAdmin для работы с поднятой в контейнере бд (с портом 5433, через который пробрасываем запросы с Windows) с ролью **суперюзера** (postgres)):

## Register - Сервер

Общие Соединение Параметры SSH Tunnel Дополнительно

Имя/адрес сервера

localhost

Порт

5433

Служебная база данных

postgres

Имя пользователя

postgres

Проверим наличие расширения:

kbd/postgres@Docker PostgreSQL\* ×

kbd/postgres@Docker PostgreSQL

Запрос История запросов

1 **SELECT** \* **FROM** pg\_extension **WHERE** extname = 'anon';

Data Output Сообщения Notifications

	oid [PK] oid	extname name	extowner oid	extnamespace oid	extrelocatable boolean	extversion text	extconfig oid[]	extcondition text[]
1	16465	anon	10	2200	false	1.2.0	[null]	[null]

Инициализируем служебную схему и функции для работы расширения:

kbd/postgres@Docker PostgreSQL\* ×

kbd/postgres@Docker PostgreSQL

No limit

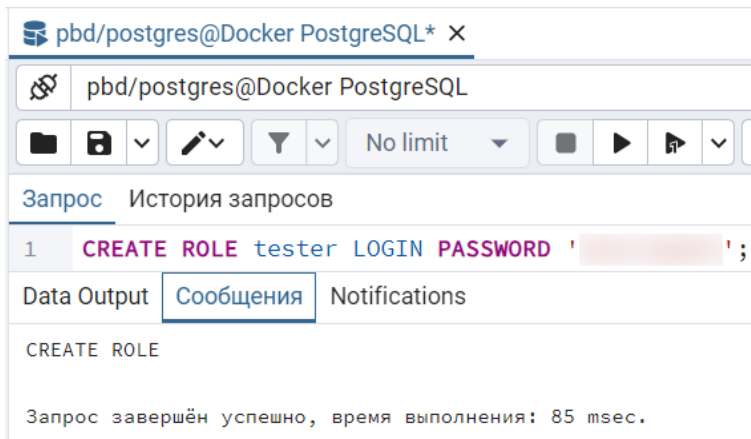
Запрос История запросов

1 **SELECT** anon.init()

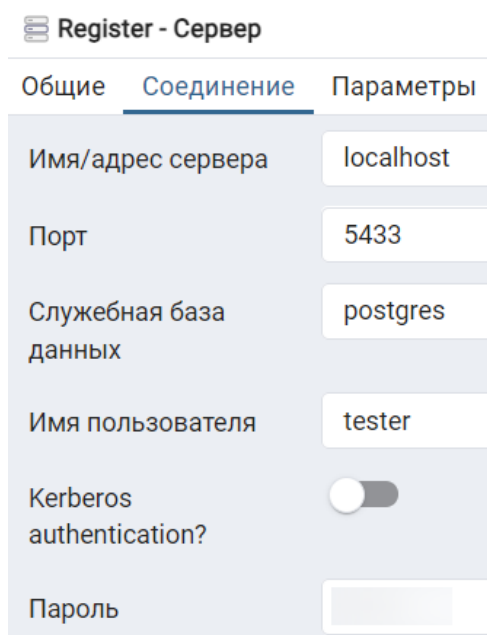
Data Output Сообщения Notifications

	init boolean
1	true

2. Создадим новую роль без прав суперюзера для тестирования маскировки данных (суперюзер видит данные не замаскированными, даже если включено маскирование):



Создаю новый сервер в pgAdmin для работы с поднятой в контейнере бд с ролью не суперюзера (tester):



Теперь имеем два сервера: один для модификаций и один для тестирования:

- > Docker PostgreSQL
- > [Redacted]
- > [No SuperUser] Docker PostgreSQL

**Выбранные поля для маскировки:**

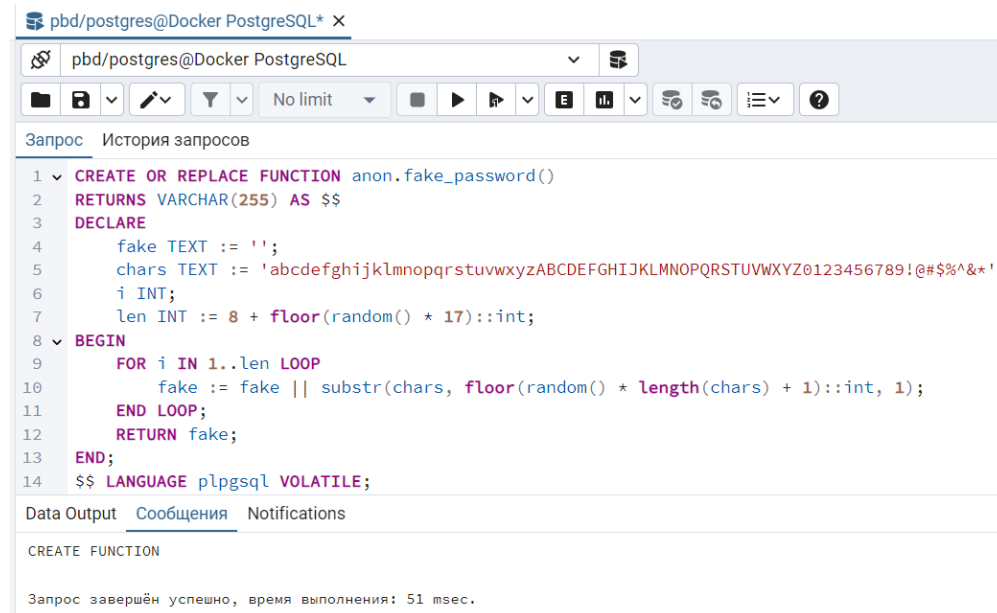
user:

name, surname, patronymic, email, phone\_number, registration\_date, password

company:

Inn, ogrn, kpp

Создадим функцию для генерации randomного пароля (длиной от 8 до 24 символов):

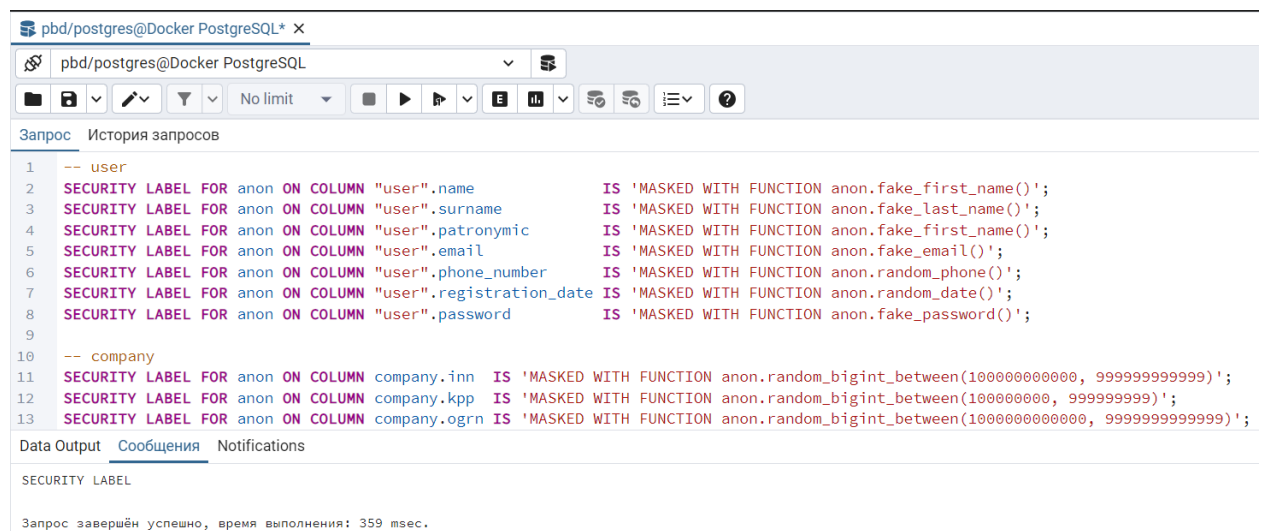


```
1 CREATE OR REPLACE FUNCTION anon.fake_password()
2 RETURNS VARCHAR(255) AS $$
3 DECLARE
4     fake TEXT := '';
5     chars TEXT := 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#%&*';
6     i INT;
7     len INT := 8 + floor(random() * 17)::int;
8 BEGIN
9     FOR i IN 1..len LOOP
10         fake := fake || substr(chars, floor(random() * length(chars) + 1)::int, 1);
11     END LOOP;
12     RETURN fake;
13 END;
14 $$ LANGUAGE plpgsql VOLATILE;
```

CREATE FUNCTION

Запрос завершён успешно, время выполнения: 51 мсек.

Создадим метки безопасности на выбранные поля:

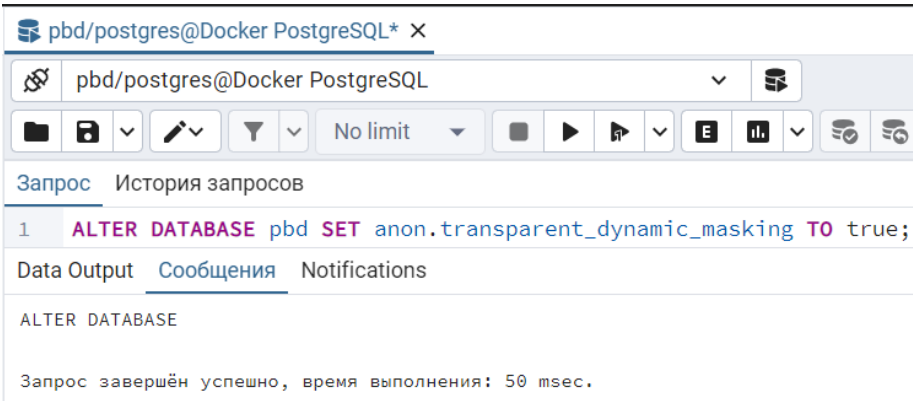


```
1 -- user
2 SECURITY LABEL FOR anon ON COLUMN "user".name IS 'MASKED WITH FUNCTION anon.fake_first_name()';
3 SECURITY LABEL FOR anon ON COLUMN "user".surname IS 'MASKED WITH FUNCTION anon.fake_last_name()';
4 SECURITY LABEL FOR anon ON COLUMN "user".patronymic IS 'MASKED WITH FUNCTION anon.fake_first_name()';
5 SECURITY LABEL FOR anon ON COLUMN "user".email IS 'MASKED WITH FUNCTION anon.fake_email()';
6 SECURITY LABEL FOR anon ON COLUMN "user".phone_number IS 'MASKED WITH FUNCTION anon.random_phone()';
7 SECURITY LABEL FOR anon ON COLUMN "user".registration_date IS 'MASKED WITH FUNCTION anon.random_date()';
8 SECURITY LABEL FOR anon ON COLUMN "user".password IS 'MASKED WITH FUNCTION anon.fake_password()';
9
10 -- company
11 SECURITY LABEL FOR anon ON COLUMN company.inn IS 'MASKED WITH FUNCTION anon.random_bigint_between(100000000000, 999999999999)';
12 SECURITY LABEL FOR anon ON COLUMN company.kpp IS 'MASKED WITH FUNCTION anon.random_bigint_between(100000000, 999999999)';
13 SECURITY LABEL FOR anon ON COLUMN company.ogrn IS 'MASKED WITH FUNCTION anon.random_bigint_between(10000000000000, 99999999999999)';
```

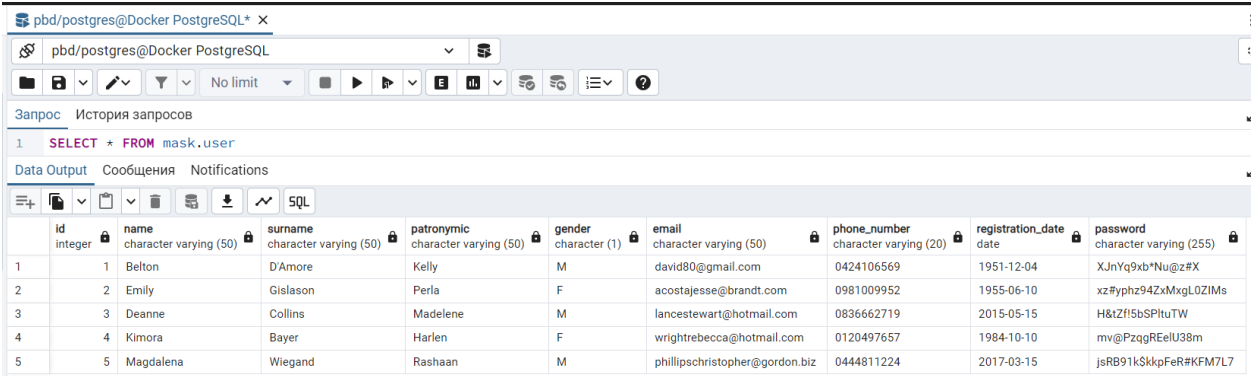
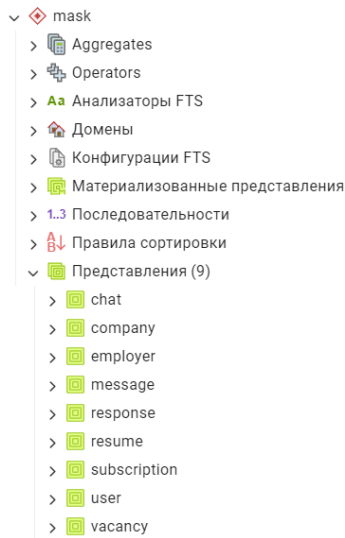
SECURITY LABEL

Запрос завершён успешно, время выполнения: 359 мсек.

Включим прозрачное маскирование через представления:



В схеме **mask** (которая появлялась при инициализации расширения) хранятся представления с установленными SECURITY LABEL:





kbd/postgres@Docker PostgreSQL ×

kbd/postgres@Docker PostgreSQL

Запрос История запросов

1 SELECT \* FROM public.user

Data Output Сообщения Notifications

	id [PK] integer	name character varying (50)	surname character varying (50)	patronymic character varying (50)	gender character (1)	email character varying (50)	phone_number character varying (20)	registration_date date	password character varying (255)
1	1	Иван	Петров	Александрович	M	ivan.petrov@mail.com	+79001234567	2023-01-15	hashed_password_1
2	2	Мария	Сидорова	Ивановна	F	maria.sidorova@mail.com	+79007654321	2023-02-20	hashed_password_2
3	3	Алексей	Кузнецов	Олегович	M	alex.kuznetsov@mail.com	+79001112233	2023-03-10	hashed_password_3
4	4	Ольга	Морозова	Петровна	F	olga.morozova@mail.com	+79005556677	2023-04-05	hashed_password_4
5	5	Дмитрий	Васильев	Сергеевич	M	dmitry.vasilyev@mail.com	+79009998877	2023-05-22	hashed_password_5

Выдадим созданной роли права на использование схемы **mask**, уберем права на SELECT из оригинальных таблиц:

kbd/postgres@Docker PostgreSQL ×

kbd/postgres@Docker PostgreSQL

Запрос История запросов

1 GRANT USAGE ON SCHEMA mask TO tester;

2 GRANT SELECT ON mask.user TO tester;

3 GRANT SELECT ON mask.company TO tester;

4

5 REVOKE SELECT ON public.user FROM tester;

6 REVOKE SELECT ON public.company FROM tester;

Data Output Сообщения Notifications

REVOKE

Запрос завершён успешно, время выполнения: 50 мсек.

Поменяем **search\_path** для роли не суперюзера так, чтобы запросы сначала шли в схему **mask**:

kbd/postgres@Docker PostgreSQL ×

kbd/postgres@Docker PostgreSQL

Запрос История запросов

1 ALTER ROLE tester SET search\_path TO mask, public;

Data Output Сообщения Notifications

ALTER ROLE

Запрос завершён успешно, время выполнения: 46 мсек.

(После чего нужно перезапустить локальную службу postgresql, либо просто перезайти в pgAdmin (или что используется), чтобы изменения **search\_path** вступили в силу)

Замаскированные данные от лица суперюзера:

**пbd/postgres@Docker PostgreSQL ×**

пbd/postgres@Docker PostgreSQL

No limit

Запрос История запросов

1 SELECT \* FROM company

Data Output Сообщения Notifications

	id [PK] integer	name character varying (100)	description text	industry character varying (50)	rating numeric (3,2)	inn character varying (12)	kpp character varying (9)	ogrn character varying (13)
1	1	ООО Техно	Компания в сфере IT	IT	4.80	123456789012	987654321	1234567890123
2	2	ЗАО Альфа	Финансовые услуги	Finance	4.50	223344556677	112233445	2233445566778
3	3	ОАО Рога и Копыта	Производство	Manufacturing	3.90	334455667788	223344556	3344556677889
4	4	АО Логистик	Транспорт и логистика	Logistics	4.20	445566778899	334455667	4455667788990
5	5	ООО ДизайнПро	Графический дизайн	Design	4.70	556677889900	445566778	5566778899001

pbd/tester@[No SuperUser] Docker PostgreSQL \* X  
 pbd/tester@[No SuperUser] Docker PostgreSQL  
 No limit  
[Запрос](#) История запросов  
 1 SELECT \* FROM "user"  
 Data Output Сообщения Notifications  

	<b>id</b> <small>integer</small>	<b>name</b> <small>character varying (50)</small>	<b>surname</b> <small>character varying (50)</small>	<b>patronymic</b> <small>character varying (50)</small>	<b>gender</b> <small>character (1)</small>	<b>email</b> <small>character varying (50)</small>	<b>phone_number</b> <small>character varying (20)</small>	<b>registration_date</b> <small>date</small>	<b>password</b> <small>character varying (255)</small>
1	1	Amelia	Nikolaus	Haley	M	scott77@yahoo.com	0193784125	1943-11-20	Uh\$MespdzUjVn
2	2	Wallace	Gutkowski	Bertrand	F	saundersandre@gmail.com	0371344856	2008-09-12	*HwDySxbQggq
3	3	Harrison	Hyatt	Ella	M	sjimenez@graham.org	0974507447	1996-11-11	C8*3ZEPct@&1q
4	4	Schley	Farrell	Dayana	F	erica28@gmail.com	0207983634	1914-01-11	rjvY\$FKw*9B5K
5	5	Kelis	Lannon	Roby	M	adamsterri@harris-benton.com	0619110478	1938-10-13	ZD9SDHFvbB1*zfs*cqmU8

**пbd/tester@[No SuperUser] Docker PostgreSQL ×**

пbd/tester@[No SuperUser] Docker PostgreSQL

Запрос История запросов

1 **SELECT \* FROM company**

Data Output Сообщения Notifications

SQL

	id integer	name character varying (100)	description text	industry character varying (50)	rating numeric (3,2)	inn character varying (12)	kpp character varying (9)	ogrn character varying (13)
1	1	ООО Техно	Компания в сфере IT	IT	4.80	705031591499	226244405	9810303866935
2	2	ЗАО Альфа	Финансовые услуги	Finance	4.50	282059797428	123062107	3899024547783
3	3	ОАО Рога и Копыта	Производство	Manufacturing	3.90	416041584200	389127732	1924713366532
4	4	АО Логистик	Транспорт и логистика	Logistics	4.20	356029521323	989004481	1374080542228
5	5	ООО ДизайнПро	Графический дизайн	Design	4.70	765461137084	527527128	1000437269257

3. Создадим материальное представление для таблиц **vacancy** (поля city, salary\_level) и **company** (поле industry), используя **Generalization**:

```
1 CREATE MATERIALIZED VIEW salary_generalization AS
2 SELECT
3     c.industry,
4     v.city,
5     anon.generalize_numrange(v.salary_level, 50000) AS salary_range,
6     COUNT(*) AS vacancy_count
7 FROM vacancy v
8 JOIN company c ON v.company_id = c.id
9 GROUP BY c.industry, v.city, salary_range;
```

Data Output   Сообщения   Notifications

SELECT 5

Запрос завершён успешно, время выполнения: 55 msec.

Данные из созданного представления **salary\_generalization**:

```
1 SELECT * FROM salary_generalization
```

Data Output   Сообщения   Notifications

	industry character varying (50)	city character varying (50)	salary_range numrange	vacancy_count bigint
1	Design	Казань	[100000,150000)	1
2	Logistics	Новосибирск	[100000,150000)	1
3	IT	Москва	[150000,200000)	1
4	Manufacturing	Екатеринбург	[50000,100000)	1
5	Finance	Санкт-Петербург	[100000,150000)	1

Применение – анализ распределения вакансий по городам, сфере и уровню зарплаты без точных чисел

Создадим материальное представление для таблиц **user** (поля email, name, surname), **resume** (поле title), **response** (поля id, response\_date) используя **Pseudonymization**:

kbd/postgres@Docker PostgreSQL ×

kbd/postgres@Docker PostgreSQL

Запрос История запросов

```
1 CREATE MATERIALIZED VIEW user_responses_pseudo AS
2 SELECT
3     r.id AS response_id,
4     anon.pseudo_email(u.email) AS pseudo_user_email,
5     anon.pseudo_first_name(u.name) AS pseudo_first_name,
6     anon.pseudo_last_name(u.surname) AS pseudo_last_name,
7     res.title AS resume_title,
8     r.response_date
9 FROM response r
10 JOIN resume res ON r.resume_id = res.id
11 JOIN "user" u ON res.user_id = u.id;
```

Data Output Сообщения Notifications

SELECT 5

Запрос завершён успешно, время выполнения: 86 мсек.

Данные из созданного представления **user\_responses\_pseudo**:

kbd/postgres@Docker PostgreSQL ×

kbd/postgres@Docker PostgreSQL

Запрос История запросов

1 SELECT \* FROM user\_responses\_pseudo

Data Output Сообщения Notifications

	response_id integer	pseudo_user_email text	pseudo_first_name text	pseudo_last_name text	resume_title character varying (100)	response_date date
1	1	deanna66@walters-oliver.net	Clay	Wolf	Разработчик Python	2023-10-16
2	2	ashleyblevins@hotmail.com	Benton	Armstrong	Аналитик данных	2023-10-17
3	3	nancyoneill@gmail.com	Jule	Nolan	Менеджер по продажам	2023-10-18
4	4	brettmacias@gonzalez-smith.com	Allena	Hilli	Логист	2023-10-19
5	5	irogers@johnson.info	Babette	Braun	Графический дизайнер	2023-10-20

Применение – анализ откликов пользователей на вакансии без раскрытия персональных данных

Создадим материальное представление для таблиц **user** (поля id, registration\_date), **resume** (поля id, user\_id), **response** (поля id, response\_date, resume\_id), **chat** (поля id, user\_id) используя **Generic Hashing** и **Adding Noise**:

pgsql/postgres@Docker PostgreSQL\* X

pgsql/postgres@Docker PostgreSQL

Запрос История запросов

```
1 CREATE MATERIALIZED VIEW user_behavior_hash_noise AS
2 SELECT
3     anon.hash(u.id::text) AS anon_user_id,
4     COUNT(DISTINCT r.id) AS total_responses,
5     COUNT(DISTINCT c.id) AS total_chats,
6     EXTRACT(DAY FROM (
7         anon.dnoise(MIN(r.response_date)::timestamp - u.registration_date::timestamp, INTERVAL '3 days')
8     )) AS days_to_first_response
9 FROM "user" u
10 LEFT JOIN resume res ON res.user_id = u.id
11 LEFT JOIN response r ON r.resume_id = res.id
12 LEFT JOIN chat c ON c.user_id = u.id
13 GROUP BY u.id, u.registration_date;
```

Data Output Сообщения Notifications

SELECT 5

Запрос завершён успешно, время выполнения: 61 мсек.

Данные из созданного представления **user\_behavior\_hash\_noise**:

pgsql/postgres@Docker PostgreSQL\* X

pgsql/postgres@Docker PostgreSQL

Запрос История запросов

1 SELECT \* FROM user\_behavior\_hash\_noise

Data Output Сообщения Notifications

	anon_user_id text	total_responses bigint	total_chats bigint	days_to_first_response numeric
1	6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b	1	0	276
2	d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13a...	1	0	238
3	4e07408562bedb8b60ce05c1decfe3ad16b72230967de01f640b7e4729b49...	1	0	222
4	4b227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdabf...	1	0	198
5	ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe3...	1	0	151

Применение – анализ поведения пользователей в процессе поиска работы без раскрытия личности (сколько: откликов, чатов с работодателями, дней до первого отклика с момента регистрации)

В итоге имеем три созданных материальных представления в схеме **public**:

- public
  - Aggregates
  - Operators
  - Анализаторы FTS
  - Домены
  - Конфигурации FTS
  - Материализованные представления (3)
    - salary\_generalization
    - user\_behavior\_hash\_noise
    - user\_responses\_pseudo

**Вывод:** в ходе лабораторной работы научился работать с расширением PostgreSQL Anonymizer. А именно: маскировать поля, проводить анонимизацию данных; создавать материальные представления, используя различные стратегии анонимизации. Убедился в корректности маскировки данных от лица обычного пользователя.