МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

“Санкт-Петербургский национальный исследовательский университет информационных технологий механики и оптики”

Мегафакультет: трансляционных информационных технологий

Факультет: информационных технологий и программирования

Лабораторная работа №4

По дисциплине: “Проектирование баз данных”

Тема: “Маскировка и анонимизация данных”

Выполнила студент группы №M3216:

Шевцов Роман Сергеевич

САНКТ-ПЕРЕРБУРГ

2025

Задачи:

1. Замаскировать поля с конфиденциальными данными.

2. Провести анонимизацию данных.

Порядок выполнения работы:

1. Установите расширение PostgreSQL Anonymizer

2. Выберите поля, которые необходимо замаскировать и модифицируйте существующие таблицы или представления.

Динамическая маскировка данных

3. Выберите данные, которые можно анализировать, скрыв, обобщив конфиденциальные данные. Создайте три Materialized Views используя:

1) Generalization – заменяет данные более широкими и менее точными значениями, диапазонами.

2) Используйте 2 стратегии анонимизации из списка:

- Destruction
- Adding Noise
- Randomization
- Faking
- Advanced Faking
- Pseudonymization
- Generic Hashing
- Partial scrambling

4. Предоставить отчёт, включить в него следующие данные:

- перечень таблиц и полей, задействованных в Materialized Views;
- данные из Materialized Views;
- код по маскировке и анонимизации.

На защите лабораторной необходимо будет продемонстрировать, как работает маскировка на ваших данных, Materialized Views, сами данные из БД, а также код.

Решение:

Выберем таблицу device:

	deviceid [PK] integer	userid integer	devicemodel text	purchasedate date	price numeric
1	1	101	iPhone 14 Pro	2023-02-15	1399.99
2	2	102	Samsung Galaxy S23	2022-12-01	1199.50
3	3	103	Xiaomi 12T	2023-03-10	649.00
4	4	104	Google Pixel 7	2023-01-25	799.99
5	5	105	OnePlus 11	2023-04-05	699.99

Создадим роль analyst чтобы проверить маскировку данных. От пользователя postgres мы можем видеть все поля в таблице, нам нужен пользователь с обычными правами(не правами админа) поэтому создаем.

Query | Query History

```
1  CREATE ROLE analyst LOGIN PASSWORD 'analyst123';
```

Выдаем ему доступ на таблицу:

Query | Query History

```
1  GRANT SELECT ON device TO analyst;
```

Включаем строковую безопасность:

Query | Query History

```
1  ALTER TABLE device ENABLE ROW LEVEL SECURITY;
```

Создаем политику доступа ко всем строкам для analyst.

Query | Query History

```
1  CREATE POLICY allow_analyst_select
2    ON device
3    FOR SELECT
4    TO analyst
5    USING (true);
6
```

Проверяем. Пока что analyst разрешено видеть все строки:

Query | Query History

```
1  SET ROLE analyst;
2  SELECT * FROM device;
```

Data Output | Messages | Notifications

	deviceid [PK] integer	userid integer	devicemodel text	purchasedate date	price numeric
1	1	101	iPhone 14 Pro	2023-02-15	1399.99
2	2	102	Samsung Galaxy S23	2022-12-01	1199.50
3	3	103	Xiaomi 12T	2023-03-10	649.00
4	4	104	Google Pixel 7	2023-01-25	799.99
5	5	105	OnePlus 11	2023-04-05	699.99

Ставим обратно роль админа:

Query | Query History

```
1  SET ROLE postgres;
```

Применяем маскировку:

Query Query History

```
1  CREATE EXTENSION IF NOT EXISTS anon CASCADE;
2  ALTER DATABASE postgres SET anon.transparent_dynamic_masking TO true;
3
4  SECURITY LABEL FOR anon ON COLUMN device.devicemodel
5  IS 'MASKED WITH FUNCTION anon.partial(devicemodel, 2, $$******$$, 2)';
6
```

Назначаем роль analyst как MASKED:

Query Query History

```
1  SECURITY LABEL FOR anon ON ROLE analyst IS 'MASKED';
```

Включаем маскировку в сессии:

Query Query History

```
1  SET anon.enable_masking = on;
```

Проверим что маскировка активна:

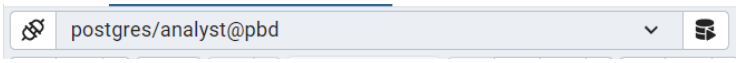
Query Query History

```
1  SHOW anon.enable_masking;
```

Data Output Messages Notifications

	anon.enable_masking text
1	on

Переключаемся на роль analyst:



Проверяем переключение:

Query Query History

```
1  SELECT current_user, session_user;
```

Data Output Messages Notifications

	current_user name	session_user name
1	analyst	analyst

Проверим что на колонках есть маскировки:

```
SELECT col.table_name, col.column_name, sec.label
FROM information_schema.columns col
JOIN pg_class cls ON col.table_name = cls.relname
JOIN pg_attribute attr ON attr.attrelid = cls.oid AND attr.attname = col.column_name
JOIN pg_seclabel sec ON sec.objoid = attr.attrelid AND sec.objsubid = attr.attnum
WHERE col.table_name = 'device';
```

	table_name name	column_name name	label text
1	device	devicemodel	MASKED WITH FUNCTION anon.fake_device_model()
2	device	purchasedate	MASKED WITH FUNCTION anon.random_date()
3	device	price	MASKED WITH FUNCTION anon.random_price()

Проверим что RLS включен:

```
SELECT relname, relrowsecurity
FROM pg_class
JOIN pg_namespace ON pg_class.relnamespace = pg_namespace.oid
WHERE relname = 'device';
```

	relname name	relrowsecurity boolean
1	device	true

Результат динамической маскировки данных от analyst:

postgres/analyst@pbd

```
SET anon.enable_masking = on;
SELECT * FROM device;
```

	deviceid [PK] integer	userid integer	devicemodel text	purchasedate date	price numeric
1	1	101	iP******ro	1900-10-04	1399.99
2	2	102	Sa******23	1900-05-04	1199.50
3	3	103	Xi******2T	1900-04-22	649.00
4	4	104	Go****** 7	1900-12-17	799.99
5	5	105	On******11	1900-01-22	699.99

Распишем Materialized Views:

Generalization View:

Query | Query History

```sql
CREATE MATERIALIZED VIEW mv_device_generalized AS
SELECT
  deviceid,
  userid,
  CASE
    WHEN devicemodel ILIKE '%iPhone%' THEN 'Apple'
    WHEN devicemodel ILIKE '%Samsung%' THEN 'Samsung'
    WHEN devicemodel ILIKE '%Xiaomi%' THEN 'Xiaomi'
    WHEN devicemodel ILIKE '%Pixel%' THEN 'Google'
    ELSE 'Other'
  END AS device_brand,
  DATE_TRUNC('year', purchasedate)::date AS purchase_year,
  CASE
    WHEN price < 700 THEN '<700'
    WHEN price < 1000 THEN '700-999'
    ELSE '1000+'
  END AS price_range
FROM device;

```

Проверка:

Query | Query History

```sql
SELECT * FROM mv_device_generalized;
```

Data Output | Messages | Notifications

	deviceid integer	userid integer	device_brand text	purchase_year date	price_range text
1	1	101	Apple	2023-01-01	1000+
2	2	102	Samsung	2022-01-01	1000+
3	3	103	Xiaomi	2023-01-01	<700
4	4	104	Google	2023-01-01	700-999
5	5	105	Other	2023-01-01	<700

Destruction:

Query | Query History

```sql
CREATE MATERIALIZED VIEW mv_device_destruction AS
SELECT
  deviceid,
  NULL AS userid, -- удаляем userID
  devicemodel,
  purchasedate,
  price
FROM device;

```

Проверка:

```
SELECT * FROM mv_device_destruction;
```

Data Output | Messages | Notifications

	deviceid integer	userid text	devicemodel text	purchasedate date	price numeric
1	1	[null]	iPhone 14 Pro	2023-02-15	1399.99
2	2	[null]	Samsung Galaxy S23	2022-12-01	1199.50
3	3	[null]	Xiaomi 12T	2023-03-10	649.00
4	4	[null]	Google Pixel 7	2023-01-25	799.99
5	5	[null]	OnePlus 11	2023-04-05	699.99

Randomization:

Save File Ctrl S

```
E MATERIALIZED VIEW mv_device_randomized AS
SELECT
  deviceid,
  userid,
  devicemodel,
  purchasedate,
  price + (random() * 100 - 50) AS noisy_price
FROM device;
```

Проверка:

```
SELECT * FROM mv_device_randomized;
```

Data Output | Messages | Notifications

	deviceid integer	userid integer	devicemodel text	purchasedate date	noisy_price double precision
1	1	101	iPhone 14 Pro	2023-02-15	1402.3791758537182
2	2	102	Samsung Galaxy S23	2022-12-01	1172.422927576941
3	3	103	Xiaomi 12T	2023-03-10	616.9092622184278
4	4	104	Google Pixel 7	2023-01-25	791.5599520562182
5	5	105	OnePlus 11	2023-04-05	682.3491419873586

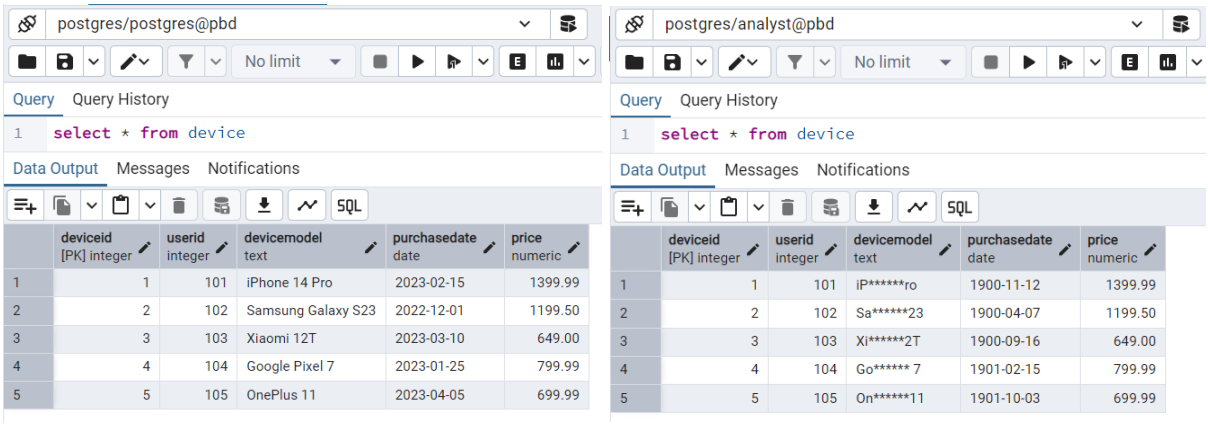
Результаты лабораторной работы:
1) Установка PostgreSQL Anonymizer через Docker. Подтверждение:

```
CREATE EXTENSION IF NOT EXISTS anon CASCADE;
SELECT anon.init();
```

Data Output | Messages | Notifications

	init boolean
1	true

2) Динамическая маскировка конфиденциальных полей. Подтверждение от двух пользователей с разными правами:



postgres/postgres@pbd

No limit

Query | Query History

```
1  select * from device
```

Data Output | Messages | Notifications

SQL

	deviceid [PK] integer	userid integer	devicemodel text	purchasedate date	price numeric
1	1	101	iPhone 14 Pro	2023-02-15	1399.99
2	2	102	Samsung Galaxy S23	2022-12-01	1199.50
3	3	103	Xiaomi 12T	2023-03-10	649.00
4	4	104	Google Pixel 7	2023-01-25	799.99
5	5	105	OnePlus 11	2023-04-05	699.99

postgres/analyst@pbd

No limit

Query | Query History

```
1  select * from device
```

Data Output | Messages | Notifications

SQL

	deviceid [PK] integer	userid integer	devicemodel text	purchasedate date	price numeric
1	1	101	iP******ro	1900-11-12	1399.99
2	2	102	Sa******23	1900-04-07	1199.50
3	3	103	Xi******2T	1900-09-16	649.00
4	4	104	Go****** 7	1901-02-15	799.99
5	5	105	On******11	1901-10-03	699.99

3) Materialized Views с разными стратегиями анонимизации

- mv_device_generalized (Generalization)

  Заменил точные данные на обобщённые:

  devicemodel → brand (Apple/Samsung)

  purchasedate → год

  price → диапазон (1000+, 700–999).

- mv_device_randomized (Adding Noise)

  Добавил случайный шум к price (±5-10%).

- mv_device_destruction (Destruction)

  Полное удаление userid (замена на NULL).