

ЗАЩИЩЕННАЯ СИСТЕМА УПРАВЛЕНИЯ  
БАЗАМИ ДАННЫХ «ЈАТОВА»

Руководство по настройке. Часть 33.  
Маскирование данных.  
Компонент «ja\_Anonymizer»

643.72410666.00067-07 98 01-33

Листов 52

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

## АННОТАЦИЯ

Компонент «ja\_Anonymizer» входит в состав СУБД «Jatoba» и обеспечивает защиту данных за счёт их маскировки или обезличивания. Это позволяет скрывать конфиденциальную коммерческую информацию и персональные данные пользователей.

Настоящее руководство предназначено для администраторов СУБД.



Все примеры в данном документе приведены для СУБД «Jatoba» версии ядра 6.x.

Версия компонента — 2.1

Степени важности примечаний, применяемые в документе:



**Важная информация** – указания, требующие особого внимания



**Дополнительная информация** – указания, позволяющие упростить работу с изделием



**Важная информация**

Для сертифицированной версии СУБД «Jatoba» поддерживается работа только на ОС, указанных в формуляре на поставку!

## СОДЕРЖАНИЕ

1. Назначение компонента.....	5
1.1. Условия применения.....	5
1.2. Ограничения при работе с компонентом.....	5
2. Установка и настройка.....	6
2.1. Установка пакета «ja_anonymizer» в ОС GNU/Linux.....	6
2.2. Настройка конфигурационного файла СУБД.....	7
2.3. Установка расширения компонента .....	8
2.3.1. Установка расширения в БД .....	8
2.3.2. Установка расширения в СУБД (глобально).....	10
3. Функциональные возможности компонента.....	12
3.1. Функции маскирования данных .....	13
3.1.1. Полная маскировка (замена исходных данных фиксированным значением).....	13
3.1.2. Частичная маскировка данных.....	13
3.1.3. Искажение исходных данных .....	14
3.1.4. Генерация случайных данных.....	15
3.2. Псевдонимизация.....	17
3.3. Хэширование исходных данных.....	19
3.4. Обобщение исходных данных .....	20
3.5. Базовая фальсификация исходных данных .....	21
3.6. Расширенная фальсификация исходных данных .....	22
3.7. Маскирование исходных данных с условием .....	23
3.8. Пользовательские функции и наборы поддельных данных.....	23
3.8.1. Пользовательские наборы данных .....	23
3.8.2. Пользовательские функции маскирования данных.....	24
4. Пример использования .....	28
4.1. Статическое маскирование данных .....	28
4.2. Динамическое маскирование данных.....	32
4.2.1. Правило маскирования данных пользователя с маской .....	33
4.2.2. Замена данных запроса фиксированным значением .....	34
4.2.3. Замена символами исходных данных .....	35
4.2.4. Замена исходных данных на случайное правдоподобное значение .....	35
4.2.5. Замена исходных данных на псевдоним.....	36
4.2.6. Результат применения динамического маскирования.....	36
4.2.7. Отключение правила динамической маскировки.....	37
4.2.8. Отключение правила динамической маскировки для пользователя.....	38
4.3. Дамп с масками данными .....	38
4.3.1. Создания пользователя с маской для работы с дампом .....	38

4.3.2. Запуск создания дампа с маскированными данными .....	39
4.3.3. Проверка маскировки данных в дампе .....	40
4.4. Представления с маскированными данными .....	40
4.5. Маскирующие обертки данных .....	43
4.5.1. Подготовка к работе с внешним источником данных .....	43
4.5.2. Создание внешней таблицы .....	44
4.5.3. Определение правил маскирования данных внешнего источника .....	45
4.5.4. Проверка маскирования данных из внешнего источника .....	46
5. Удаление расширения .....	48
6. Возможные ошибки .....	49
6.1. Ошибка при выгрузке дампа с маскированными данными из БД в режиме «только чтение» .....	49
Перечень сокращений .....	51

## 1. НАЗНАЧЕНИЕ КОМПОНЕНТА

Компонент «ja\_Anonymizer» — это расширение СУБД «Jatoba», предоставляющее механизм маскирования или обезличивания данных.

Такая функциональная возможность позволяет объявлять правила маскирования, используя язык описания данных (DDL), и определять свою стратегию маскировки данных внутри самого определения таблицы БД.

### 1.1. Условия применения

Компонент «ja\_Anonymizer» может использоваться с СУБД «Jatoba» версий 6.x и выше, под управлением операционных систем GNU/Linux.

Компонент выполнен в форме расширения СУБД и не имеет ограничений по совместимости с другими компонентами.

### 1.2. Ограничения при работе с компонентом

Имеет ряд особенностей по функциональным возможностям:

- скорость выполнения статической маскировки зависит от объема БД и количества созданных правил маскировки;
- динамическая маскировка данных применяется только для одной схемы, по умолчанию — public. В случае необходимости схема может быть изменена пользователем;
- динамическая маскировка данных может существенно замедлить работу с некоторыми запросами, особенно при попытке соединить две таблицы по замаскированному ключу, используя хеширование или псевдомаскировку.

## 2. УСТАНОВКА И НАСТРОЙКА

Установка компонента должна производиться от имени пользователя, обладающего административными привилегиями в системе.

Установка компонента под управлением ОС GNU/Linux приведено ниже.

### 2.1. Установка пакета «ja\_anonymizer» в ОС GNU/Linux

Компонент устанавливается в составе СУБД «Jatoba». Его возможно установить при первичной установке либо при последующей эксплуатации СУБД.

Установку пакета компонента возможно провести двумя способами:

- 1) установка из локального репозитория (CDROM) – производится из файлов, записанных на компакт-диск или скопированных с него;
- 2) установка непосредственно из deb/rpm-файлов – производится опционально, по усмотрению пользователя.

Компонент выполнен в виде отдельного deb или rpm-пакета. Установка компонента осуществляется средствами пакетного менеджера ОС. Для разных типов пакетных менеджеров команда установки немного отличается. Ниже приведены основные типы:

— для систем на основе пакетного менеджера APT (к таким системам относятся все ОС семейства Debian, использующие deb-пакеты) команда установки следующая:

```
# apt-get install jatoba<ver>-ja_anonymizer
```

— для систем на основе пакетных менеджеров YUM/DNF (к таким системам относятся все ОС семейства RedHat и вышедшие из нее, использующие rpm-пакеты) команда установки, следующая:

```
# yum install jatoba<ver>-ja_anonymizer
```

Отдельного уточнения требуют операционные системы ALT Linux и openSUSE.

— ALT Linux использует пакетный менеджер APT, но распространяется в виде rpm-пакетов и для нее команда установки выглядит аналогично Debian:

```
# apt-get install jatoba<ver>-ja_anonymizer
```

Установка компонента в составе других версий СУБД «Jatoba» осуществляется аналогично. Отличие будет только в номере версии СУБД, в составе которой он распространяется.

Удаление модуля также осуществляется средствами пакетного менеджера ОС. Вместо команды `install` нужно использовать соответствующую данному пакетному менеджеру команду удаления (`remove`, `purge`, `erase` и т.п.).

Для получения детальной информации по пакетному менеджеру рекомендуется обратиться к документации по ОС.

В результате установки пакета в директории:

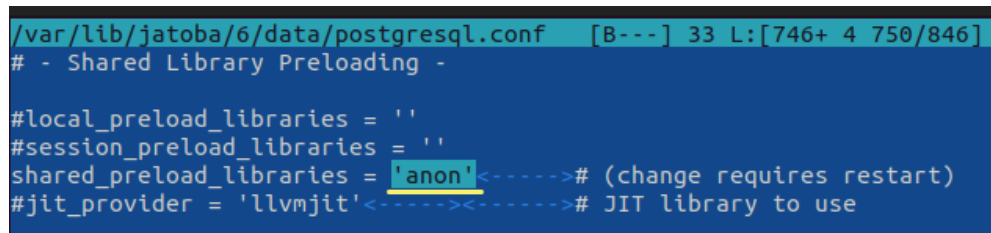
- `/usr/jatoba-6/lib/` будет создана библиотека `anon.so`;
- `/usr/jatoba-6/share/extension/` будет создан управляющий файл расширения `anon.control`;
- `/usr/jatoba-6/share/extension/anon/` будет создан набор csv-файлов.

## 2.2. Настройка конфигурационного файла СУБД

Чтобы активировать функции компонента, необходимо выполнить следующие действия:

- 1) Добавьте имя библиотеки в переменную `shared_preload_libraries` в файле `postgresql.conf`:

```
shared_preload_libraries = 'anon'
```



```
/var/lib/jatoba/6/data/postgresql.conf [B---] 33 L:[746+ 4 750/846]
# - Shared Library Preloading -
#local_preload_libraries = ''
#session_preload_libraries = ''
shared_preload_libraries = 'anon'<-----># (change requires restart)
#jit_provider = 'llvmjit'<-----># JIT library to use
```

Рисунок 2.1 – Настройки библиотеки компонента в файле `postgresql.conf`

- 2) Перезагрузить СУБД, чтобы активировать внесённые изменения:

```
# systemctl restart jatoba-6
```

- 3) Проверить статус работы СУБД после перезагрузки:

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```
# systemctl status jatoba-6
```

Дополнительно можно проверить правильность загрузки библиотеки расширения выполнив следующий запрос:

```
SHOW shared_preload_libraries;
```

```

root@node1: /home/admin1
root@node1:/home/admin1# su postgres
postgres@node1:/home/admin1$ psql
Password for user postgres:
psql (16.6)
Type "help" for help.

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

postgres=#
  
```

Рисунок 2.2 – Информация о загруженной в СУБД библиотеке компонента

### 2.3. Установка расширения компонента


После перезагрузки СУБД и загрузки библиотеки компонента появляется возможность установки расширения «ja\_anonymizer».

Установка расширения возможна двумя способами:

- Для БД, в которой требуется маскирование данных;
- Глобально для СУБД.

Установка расширения выполняется от имени и с правами привилегированного пользователя СУБД.

#### 2.3.1. Установка расширения в БД

 Здесь и далее расширение устанавливается в БД «anon\_db» в качестве примера.

Последовательность действий при установке расширения в БД следующая:

- 1) Создать БД, в которую будет устанавливаться расширение компонента:

```
CREATE DATABASE anon_db;
\connect anon_db
```

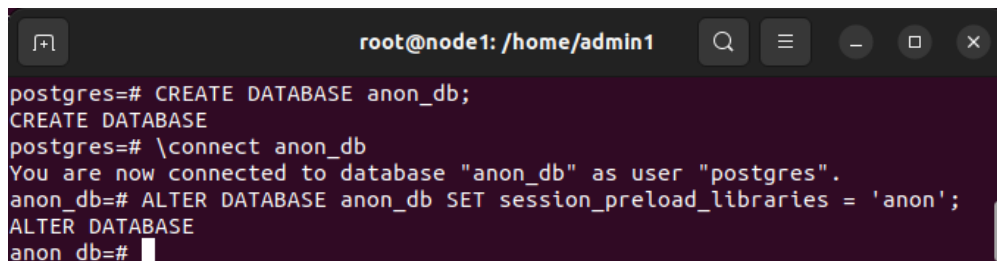


2) Загрузку компонента в БД рекомендуется выполнять следующим образом:

```
ALTER DATABASE anon_db SET session_preload_libraries = 'anon';  
\connect anon_db
```

Данный способ имеет ряд преимуществ:

- Будет выполнен сброс pg\_dump с помощью опции -c, В этом случае дамп БД будет самодостаточным.
- Распространяется на резервный экземпляр БД с помощью потоковой репликации. Это означает, что возможно использование функции анонимизации на копии БД, доступном только для чтения (при условии, что расширение так же установлено на резервном экземпляре БД).

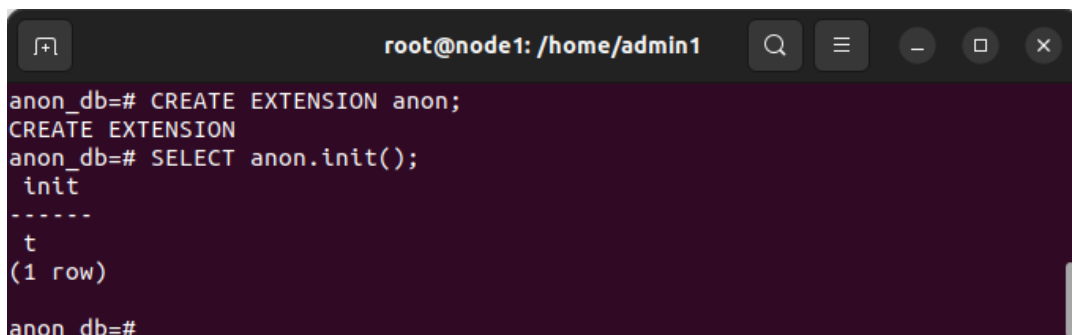


```
root@node1: /home/admin1  
postgres=# CREATE DATABASE anon_db;  
CREATE DATABASE  
postgres=# \connect anon_db  
You are now connected to database "anon_db" as user "postgres".  
anon_db=# ALTER DATABASE anon_db SET session_preload_libraries = 'anon';  
ALTER DATABASE  
anon_db=#
```

Рисунок 2.3 – Создание БД anon\_db и регистрация загрузки библиотеки компонента

3) Установка и инициализация расширения:

```
CREATE EXTENSION anon CASCADE;  
SELECT anon.init();
```



```
anon_db=# CREATE EXTENSION anon;  
CREATE EXTENSION  
anon_db=# SELECT anon.init();  
init  
-----  
t  
(1 row)  
anon_db=#
```

Рисунок 2.4 – Установка в БД расширения компонента и инициализация

4) Просмотр установленных расширений в БД:

```
\dx
```

```
root@node1: /home/admin1
anon_db=# \dx
                                List of installed extensions
  Name    | Version | Schema  | Description
-----+-----+-----+-----
anon      | 2.1.1   | public  | Anonymization & Data Masking for PostgreSQL
plpgsql   | 1.0     | pg_catalog | PL/pgSQL procedural language
(2 rows)
anon_db=#
```

Рисунок 2.5 – Просмотр установленных расширений

Проверка загруженных расширений в БД:

```
SHOW session_preload_libraries;
SELECT * FROM pg_extension WHERE extname= 'anon';
SELECT anon.is_initialized();
```

```
root@node1: /home/admin1
anon_db=# SELECT * FROM pg_extension WHERE extname= 'anon';
 oid | extname | extowner | extnamespace | extrelocatable | extversion | extconfig | extcondition
-----+-----+-----+-----+-----+-----+-----+-----
17643 | anon    | 10      | 2200         | f               | 2.1.1     |           |
(1 row)
anon_db=#
```

Рисунок 2.6 – Просмотр установленных расширений

### 2.3.2. Установка расширения в СУБД (глобально)

Вторым способом является установка расширения глобально для СУБД.

Последовательность действий при установке расширения в СУБД следующая:

- 1) Загрузку компонента в БД рекомендуется выполнять следующим образом:

```
ALTER SYSTEM SET shared_preload_libraries = 'anon';
```

- 2) Выполнить перезагрузку СУБД:

```
# systemctl restart jatoba-6
```

- 3) Проверить статус службы СУБД:

```
# systemctl status jatoba-6
```

4) Далее выполнить установку и проверку расширения согласно пунктам 3)-4) из п.п. 2.3.1.

### 3. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ КОМПОНЕНТА

Компонент предоставляет следующие функциональные возможности:

- статическое маскирование (Static Masking): безвозвратно замаскировать все персональные данные, хранящиеся в БД, заменив их другими значениями, сохранив структуру для дальнейшего анализа;
- динамическое маскирование (Dynamic Masking): включить «прозрачную» маскировку для определённых (MASKED) пользователей, чтобы они не имели возможности доступа к маскируемым реальным персональным данным;
- анонимные дампы (Anonymous Dumps): экспортировать замаскированную версию данных в внешний SQL-файл. SQL-файл с замаскированными данными возможно использовать для тестов или передачи другим пользователям;
- маскирующие представления (Masking Views): создание специальных представлений (views), в которых конфиденциальные поля уже замаскированы;
- маскирующие обертки данных (Masking Data Wrappers): применение правил маскировки к данным, поступающим в БД из внешних источников.

Кроме указанных пяти способов, также существуют конкретные функции маскировки, используемые для определенных полей таблиц, такие как:

- рандомизация;
- фальсификация;
- частичное скрывание;
- перестановка;
- искажение.

Рандомизация заменяет конфиденциальные данные случайными, но правдоподобными значениями. Цель состоит в том, чтобы исключить возможность какой-либо идентификации по записи данных, оставляя её пригодной для тестирования, анализа и обработки данных.

Перестановка перемешивает значения в рамках столбца. Исходные данные могут быть восстановлены, если алгоритм перестановки будет расшифрован.

Частичное скрывание заменяет часть значения, а часть данных оставляет нетронутыми. Например: номер кредитной карты может быть заменён на «40XX XXXX XXXX XX96»

### 3.1. Функции маскирования данных

#### 3.1.1. Полная маскировка (замена исходных данных фиксированным значением)

При полной маскировке исходные данные могут быть полностью заменены на определенный фиксированный набор символов.

Замена на фиксированное значение описана в п.п. 4.2.2 данного руководства.

Полная маскировка данных возможна при использовании синтаксической конструкции:

```
MASKED WITH VALUE $$текст$$
```

Или

```
$$MASKED WITH VALUE 'текст'$$
```



В данном случае символы экранирования (\$\$) находятся снаружи от маскирующего значения.

#### Равнозначные примеры:

```
MASKED WITH VALUE $$CONFIDENTIAL$$
```

```
$$MASKED WITH VALUE 'CONFIDENTIAL'$$
```

#### 3.1.2. Частичная маскировка данных

Частичная маскировка данных заключается в том, что часть оригинального значения остается видимой, а другая часть скрывается.

Таблица 3.1 – Функции частичной маскировки данных

№	Функция	Описание	Пример
1	anon.partial(original_value,n1,text,n2)	Маскирование части символов, original_value -	anon.partial(postcode,1,\$\$****\$\$,1) → заменить значение столбца

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

		значение или имя столбца таблицы, которое будет искажено, n1 и n2 - количество не маскируемых символов в начале и конце строки, text - набор символов для подмены значения	postcode на ****, оставив видимыми первый и последний символы, например так 6****5
2	anon.partial_email(mail_value)	Функция для маскировки адреса электронной почты, mail_value - адрес электронной почты	anon.partial_email(email) → заменяет значение столбца email на *****, оставляя видимыми по 2 первых символа, например так da*****@gm*****.com

### 3.1.3. Искажение исходных данных

Искажение исходных данных возможно применять к числовым значениям или датам в определенных пределах.

Например, можно скрыть день рождения, заменив его случайной датой в пределах  $\pm 1$  месяц. Такое искажение скроет персональные данные, но практически не повлияет на аналитические отчеты (например анализ возрастных групп покупателей).

Таблица 3.2 – Функции искажения исходных данных

№	Функция	Описание	Пример
1	anon.noise(original_value, ratio)	Искажение числовых значений, original_value - значение или имя столбца таблицы, которое будет искажено, ratio - коэффициент искажения	anon.noise(salary, 0.33) → изменить значения столбца salary на $\pm 33\%$
2	anon.dnoise(original_value, interval)	Искажение даты, original_value - значение или имя столбца таблицы, которое будет искажено, interval - интервал сдвига	anon.dnoise(date, '1 month') → изменить значения столбца date на $\pm 1$ месяц



Функции маскирования «noise()» уязвимы для повторных атак, особенно при использовании «динамического маскирования». Пользователь под маской может угадать исходное значение, запросив его замаскированное значение

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

несколько раз, а затем просто использовать «AVG()» функцию для получения приближенного значения.

Вкратце, эти функции лучше всего подходят для «анонимных дампов» и «статического маскирования». Их следует избегать при использовании «динамического маскирования».

### 3.1.4. Генерация случайных данных

Генерация абсолютно случайных данных.

Выделяются следующие группы генерации случайных данных:

- Простая генерация;
- Значение в определенном интервале;
- Из заданного массива;
- Из перечисляемого типа ENUM;
- Значение из диапазона (RANGE);
- Из последовательности (sequence).

Таблица 3.3 – Функции генерации случайных данных

№	Функция	Описание	Пример
<b>Простая генерация</b>			
1	anon.random_date()	Возвращает случайную дату	anon.random_date() → случайная дата
2	anon.random_string(n)	Случайная строка, n - количество символов	anon.random_string(10) → случайная строка из 10 символов
3	anon.random_zip()	Случайный 5-значный цифровой код	anon.random_zip() → 5-значный цифровой код
4	anon.random_phone(p)	Случайный 9-значный телефонный номер, p - префикс номера	anon.random_phone('+79') → номер, начинающийся на +79
5	anon.random_hash(seed)	Случайный хеш строки, seed - срока	anon.random_hash('3Se6j') → случайный хеш
<b>Генерация значения из определенного интервала</b>			
№ изменения: _____		Подпись отв. лица: _____	Дата внесения изм: _____

6	anon.random_date_between(d1,d2)	Случайная дата в интервале между d1 и d2	anon.random_date_between('2024-01-01','2024-12-31') → случайная дата 2024 года
7	anon.random_int_between(i1,i2)	Случайное число INT в интервале между i1 и i2	anon.random_int_between(1,10) → случайное число от 1 до 10
8	anon.random_bigint_between(b1,b2)	Случайное число BIGINT в интервале между b1 и b2	anon.random_bigint_between(10,1000) → случайное число от 10 до 1000
<b>Генерация из массива</b>			
9	anon.random_in(ARRAY[n1,n2,...,nX]))	Случайный элемент из массива от n1 до nX	anon.random_in(ARRAY['red','green','blue']) → случайный цвет из 3 элементов массива
<b>Случайный выбор из перечисляемого типа ENUM</b>			
10	anon.random_in_enum(var_of_an_enum_type)	Случайный элемент из перечисляемого типа ENUM	CREATE TYPE card AS ENUM ('visa', 'mastercard', 'unionpay'); anon.random_in_enum(NULL::CARD) → случайный выбор из типа CARD
<b>Выбор случайного значения из диапазона (RANGE)</b>			
11	anon.random_in_int4range('[n1,n2]')	Случайное число INT от n1 (включительно) до n2 (исключается)	anon.random_in_int4range('[5,6)') → вернется число INT равное 5
12	anon.random_in_int8range('(n1,n2]')	Случайное число BIGINT от n1 (исключается) до n2 (включительно)	anon.random_in_int8range('(6,7]') → вернется число BIGINT равное 7
13	anon.random_in_numrange('[n1,n2]')	Случайное число NUMERIC между n1 и n2	anon.random_in_numrange('[0.1,0.9]') → число NUMERIC между 0.1 и 0.9
14	anon.random_in_daterange('[d1,d2]')	Случайная дата от d1 (включительно) до d2 (исключается)	anon.random_in_daterange('[2001-01-01, 2002-01-01)') → дата из 2001 года
15	anon.random_in_tsrange('[t1,t2]')	Случайная дата в формате TIMESTAMP между t1 и t2	anon.random_in_tsrange('[2022-10-01,2022-10-31]') → случайный TIMESTAMP в пределах октября 2022
16	anon.random_in_tstzrange('[t1,t2]')	Случайная дата в формате TIMESTAMP между t1 и t2 с часовым поясом (TIMEZONE)	anon.random_in_tstzrange('[2022-10-01,2022-10-31]') → случайный TIMESTAMP с часовым поясом



Генерация последовательных значений (sequence)			
17	<code>anon.random_id()</code>	Случайное уникальное значение типа BIGINT, каждый вызов значение увеличивается, подобно <code>nextval()</code>	<code>anon.random_id()</code> → случайный последовательный BIGINT
18	<code>anon.random_id_int()</code>	Случайное уникальное значение типа INT, каждый вызов значение увеличивается, подобно <code>nextval()</code>	<code>anon.random_id_int()</code> → случайный последовательный INT
19	<code>anon.random_id_small_int()</code>	Случайное уникальное значение типа SMALLINT, каждый вызов значение увеличивается, подобно <code>nextval()</code>	<code>anon.random_id_small_int()</code> → случайный последовательный SMALLINT



Символы '[' и ']' – означают включение значения в диапазон. Символы '(' и ')' – означают исключение значения в диапазона.

Например, `(anon.random_in_int4range('[n1,n2]'))` – случайное число INT от n1 (включительно) до n2 (исключается).



Невозможно получить случайное значение из диапазона с бесконечной границей. Например:

```
anon.random_in_int4range('[2022,)' )
```

возвращается NULL.



Каждый вызов функций генерации последовательных значений будет возвращать увеличенное значение, во многом похожее на функцию `[nextval()]`.

В любой момент вы можете сбросить текущее значение последовательности, заменив его новым значением. Например:

```
SELECT pg_catalog.setval('anon.random_id_seq', 42);
```

### 3.2. Псевдонимизация

Псевдонимизация создает реалистичные значения (псевдонимы), связанные с исходным значением.

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

Данная функция всегда будут возвращать одно и то же значение. В этом состоит основное отличие псевдонимизации от фальсификации.

При вызове любой из функций псевдонимизации возможно использование аргумента salt. Аргумент salt предназначен для увеличения сложности генерируемых данных и минимизации атак по словарю. Если соль (salt) не указана в аргументе, вместо неё используется значение общего параметра anon.salt.

Таблица 3.4 – Функции псевдонимизации исходных данных

№	Функция	Описание	Пример
1	anon.pseudo_first_name(seed,salt)	Псевдоним имени, seed - имя столбца таблицы, salt - необязательный аргумент	anon.pseudo_first_name(fname) → псевдоним имени для столбца fname
2	anon.pseudo_last_name(seed,salt)	Псевдоним фамилии, seed - имя столбца таблицы, salt - необязательный аргумент	anon.pseudo_last_name(lname) → псевдоним фамилии для столбца lname
3	anon.pseudo_email(seed,salt)	Псевдоним электронной почты, seed - имя столбца таблицы, salt - необязательный аргумент	anon.pseudo_email(email) → случайный действительный электронный адрес для столбца email
4	anon.pseudo_city(seed,salt)	Псевдоним города, seed - имя столбца таблицы, salt - необязательный аргумент	anon.pseudo_city(work_city) → случайное название города для столбца work_city
5	anon.pseudo_country(seed,salt)	Псевдоним страны, seed - имя столбца таблицы, salt - необязательный аргумент	anon.pseudo_country(user_country) → случайное название страны для столбца user_country
6	anon.pseudo_company(seed,salt)	Псевдоним компании, seed - имя столбца таблицы, salt - необязательный аргумент	anon.pseudo_company(cname) → случайное название компании для столбца cname
7	anon.pseudo_iban(seed,salt)	Случайный действительный номер IBAN (международный номер банковского счёта), seed - имя столбца таблицы, salt - необязательный аргумент	anon.pseudo_iban(iban_cust) → случайный действительный номер IBAN для столбца iban_cust
8	anon.pseudo_siret(seed,salt)	Случайный действительный 14-значный номер SIRET, seed - имя столбца таблицы,	anon.pseudo_siret(siret_code) → случайный действительный

		salt - необязательный аргумент	идентификатор SIRET для столбца siret_code
9	anon.pseudo_shift()	Сдвиг используя секретное значение (anon.shift) для псевдонимизации. Секретное значение может быть инициализировано случайным образом с помощью anon.set_shift() или определено с помощью anon.set_shift(INT).	anon.pseudo_shift(id) → сдвиг идентификатора первичного ключа
10	anon.pseudo_xor()	Выполнение операции «исключающее ИЛИ» используя секретное значение (anon.shift) для псевдонимизации. Секретное значение может быть инициализировано случайным образом с помощью anon.set_shift() или определено с помощью anon.set_shift(INT).	anon.pseudo_xor(id) → выполнение операции «исключающее ИЛИ» над значением идентификатора первичного ключа



Этот набор данных представлен на английском языке и очень мал (1000 значений для каждой категории). Если вы хотите использовать локализованные данные или загрузить определённый набор данных, ознакомьтесь с разделом 3.8 Пользовательские функции и наборы поддельных данных.

### 3.3. Хэширование исходных данных

Хэширование исходных данных не является допустимым методом маскирования. Тем не менее на практике иногда необходимо сгенерировать детерминированный хэш исходных данных.

Например, в случае если таблицы связаны через пару первичного и внешнего ключа. В таком случае рекомендуется использовать функцию anon.hash(), а не функцию anon.digest(), так как, в этом случае, salt не будет отображаться в правиле маскирования.

Таблица 3.5 – Функции хэширования исходных данных

№	Функция	Описание	Пример
---	---------	----------	--------

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

1	<code>anon.hash(value)</code>	Текстовый хеш значения <code>value</code> , при этом будут использованы общие параметры <code>anon.salt</code> и <code>anon.algorithm</code>	<code>anon.hash('Abc123')</code> → хеш значения 'Abc123'
2	<code>anon.digest(value,salt,algorithm)</code>	Текстовый хеш значения <code>value</code> , с учетом переданных аргументов: <code>salt</code> - секретная соль, <code>algorithm</code> - алгоритм хеширования	<code>anon.digest('Abc123','abcd','sha224')</code> → хеш значения 'Abc123', по алгоритму sha224

Для того чтобы определить общие параметры функций хеширования необходимо использовать следующие запросы:

```
ALTER DATABASE anon_db SET anon.salt TO '[anon.salt]';
```

Где `anon.salt` – случайные символы, например `xsfnjefnjsnfjsnf`.

```
ALTER DATABASE anon_db SET anon.algorithm TO '[anon.algorithm]';
```

Где `algorithm` – используемый алгоритм хеширования, например `md5`, `sha1`, `sha224`, `sha256`, `sha384`, `sha512`.



Функции хеширования не будут работать, если входные данные содержат неэкранированный символ (особенно одинарный обратный слеш). В большинстве случаев это признак ошибки в приложении, как правило, когда входные данные не проходят надлежащую очистку.

### 3.4. Обобщение исходных данных

Обобщение исходных данных заключается в замене исходного значения диапазоном, содержащим это значение. То есть, информацию: «Максиму 42 года», можно обобщить как: «Максиму от 40 до 50 лет».

Таблица 3.6 – Функции обобщения исходных данных

№	Функция	Описание	Пример
1	<code>anon.generalize_tsrange(date,step)</code>	Функция обобщения даты <code>date</code> , возвращает диапазон дат, определенный аргументом <code>step</code> - это период: день, месяц, год и т.д.	<code>anon.generalize_tsrange('1952-07-17','decade')</code> → этот пример вернет диапазон "1950-01-01 00:00:00","1960-01-01 00:00:00"
№ изменения: _____		Подпись отв. лица: _____	Дата внесения изм: _____

2	<code>anon.generalize_int4range(value, step)</code>	Функция вернет диапазон типа <code>integer</code> , <code>value</code> - числовое значение, <code>step</code> - размер диапазона	<code>anon.generalize_int4range(55, '10')</code> → этот пример вернет диапазон 50,60
3	<code>anon.generalize_int8range(value, step)</code>	Функция вернет диапазон типа <code>bigint</code> , <code>value</code> - числовое значение, <code>step</code> - размер диапазона	<code>anon.generalize_int8range(5005, 200)</code> → этот пример вернет диапазон 5000,5200
4	<code>anon.generalize_numrange(value, step)</code>	Функция вернет диапазон типа <code>numeric</code> , <code>value</code> - числовое значение, <code>step</code> - размер диапазона	<code>anon.generalize_numrange(5.47, 2)</code> → этот пример вернет диапазон 4,6

### 3.5. Базовая фальсификация исходных данных

Фальсификация - замена данных случайными, но правдоподобными значениями.

В отличие от псевдонимизации, каждый запрос данных будет возвращать новое случайное значение. Цель состоит в том, чтобы избежать какой-либо идентификации по записи данных, оставляя её при этом пригодной для тестирования, анализа и обработки данных.

Кроме фальсификации таких данных, как имя, фамилия, название компании и так далее, компонент, позволяет фальсифицировать текстовые данные подходящим по размеру случайным текстом.

Таблица 3.7 – Функции базовой фальсификации исходных данных

№	Функция	Описание	Пример
1	<code>anon.fake_address()</code>	Функция вернет случайный почтовый адрес	<code>anon.fake_address()</code> → случайный адрес, например: 743 Wade Point Suite 171, Lake Connie, MT 92286
2	<code>anon.fake_city()</code>	Функция вернет случайное название города	<code>anon.fake_city()</code> → случайное название города
3	<code>anon.fake_country()</code>	Функция вернет случайное название страны	<code>anon.fake_country()</code> → случайное название страны
4	<code>anon.fake_company()</code>	Функция вернет случайное название компании	<code>anon.fake_company()</code> → случайное название компании

5	<code>anon.fake_email()</code>	Функция вернет случайный адрес электронной почты	<code>anon.fake_email()</code> → случайный адрес электронной почты
6	<code>anon.fake_first_name()</code>	Функция вернет случайное имя	<code>anon.fake_first_name()</code> → случайное имя
7	<code>anon.fake_last_name()</code>	Функция вернет случайную фамилию	<code>anon.fake_last_name()</code> → случайная фамилия
8	<code>anon.fake_postcode()</code>	Функция вернет случайный 5-значный почтовый код	<code>anon.fake_postcode()</code> → 5-значный почтовый код
9	<code>anon.fake_iban()</code>	Случайный действительный номер IBAN (международный номер банковского счёта)	<code>anon.fake_iban()</code> → случайный действительный номер IBAN
10	<code>anon.fake_siret()</code>	Случайный действительный 14-значный номер <u>SIRET</u>	<code>anon.fake_siret()</code> → случайный действительный номер SIRET
11	<code>anon.lorem_ipsum(paragraphs := value)</code>	Функция вернет несколько параграфов случайного текста, количество определяется <i>value</i>	<code>anon.lorem_ipsum(paragraphs := 4)</code> → 4 параграфа случайного текста
12	<code>anon.lorem_ipsum(words := value)</code>	Функция вернет несколько случайных слов, количество определяется <i>value</i>	<code>anon.lorem_ipsum(words := 20)</code> → 20 случайных слов
13	<code>anon.lorem_ipsum(characters := value)</code>	Функция вернет случайную строку, длина которой определяется <i>value</i>	<code>anon.lorem_ipsum(characters := anon.length(table.column))</code> → вернет то же количество символов, что и в исходном столбце

### 3.6. Расширенная фальсификация исходных данных

Расширенная фальсификация основана на функции `dummy_*`, которая насчитывает более 70 параметров, таких как: случайный город, IP-адрес, фамилия, имя и т.д.

Функция `dummy_*` позиционируется как более продвинутая, по сравнению с `fake_*`.

Локализация в компоненте доступна только для функций `dummy_*`. Это достигается добавлением слова `_locale` к названию функции (например `dummy_last_name_locale('fr_FR')` вернет французскую фамилию).

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

На данный момент доступны следующие варианты локализации фальсификации данных: en\_US (по умолчанию), ar\_SA, fr\_FR, ja\_JP, pt\_BR, zh\_CN, zh\_TW.

### 3.7. Маскирование исходных данных с условием

В некоторых ситуациях может потребоваться применить маскирующий фильтр только для определенных значений или для ограниченного числа строк в таблице.

Например, если надо замаскировать только строки, содержащие значения, при этом строки, содержащие NULL, не маскировать. Для этого предназначена функция anon.ternary, которая работает как классическое условие: CASE WHEN x THEN y ELSE z.

Таблица 3.8 – Функция маскирования исходных данных с условием

№	Функция	Описание	Пример
1	anon.ternary(condition,value1,value2)	Если выполняется условие condition, то выбирается значение value1, иначе value2	anon.ternary(score IS NULL, NULL, anon.random_int_between(0,100)) → изменить значения столбца на случайное число (от 0 до 100), если его значение не NULL.



Условное маскирование может создать частично детерминированную «связь» между исходными и замаскированными данными. Эта связь может быть использована для извлечения персонифицированных данных из замаскированных.

### 3.8. Пользовательские функции и наборы поддельных данных

При работе с пользовательскими функциями необходимо создать пользователя с маской. Для этого необходимо выполнить следующий запрос:

```
CREATE ROLE maskuser LOGIN;
```

#### 3.8.1. Пользовательские наборы данных

По умолчанию расширение поставляется с небольшим набором поддельных данных на английском языке (address.csv, company.csv, email.csv, iban.csv, identifiers\_category.csv, lorem\_ipsum.csv, siret.csv, city.csv, country.csv, first\_name.csv, identifier.csv, last\_name.csv, postcode.csv).

Набор файлов поддельных данных на английском языке расположен в каталоге /usr/jatoba-6/share/extension/anon/.

Для того чтобы функция anon.fake\_\* работала с пользовательскими данными, необходимо подготовить собственные наборы по образцу из файлов, затем их можно импортировать из файлов в формате CSV с помощью запроса:

```
SELECT anon.init('/path/to/custom_csv_files/');
```

Где /path/to/custom\_csv\_files/ - путь к каталогу с набором пользовательских данных.

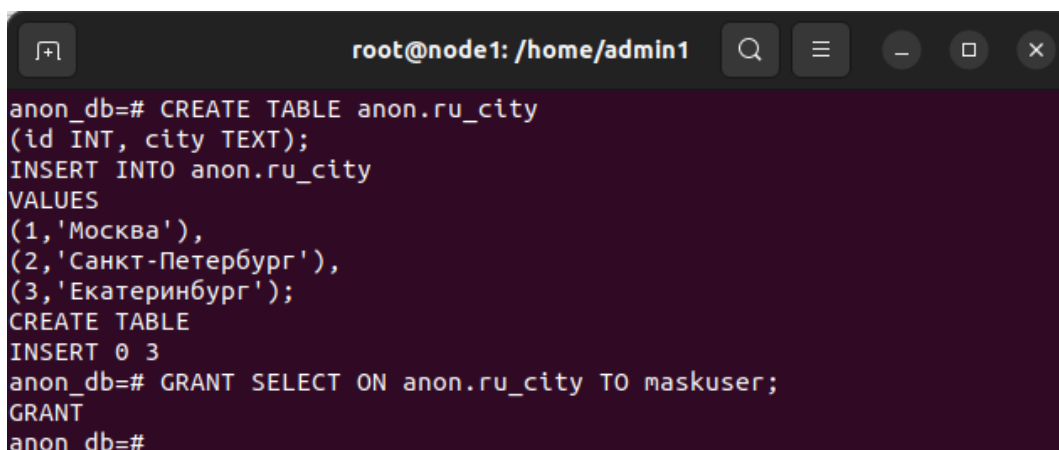
### 3.8.2. Пользовательские функции маскирования данных

Кроме поставляемых с расширением функций, есть возможность создавать собственные функции для маскирования данных.

Для пользовательской функции потребуется создать таблицу, которая будет служить справочником значений:

```
CREATE TABLE anon.ru_city  
(id INT, city TEXT);  
INSERT INTO anon.ru_city  
VALUES  
(1, 'Москва'),  
(2, 'Санкт-Петербург'),  
(3, 'Екатеринбург');  
GRANT SELECT ON anon.ru_city TO maskuser;  
GRANT SELECT ON public.users TO maskuser;
```



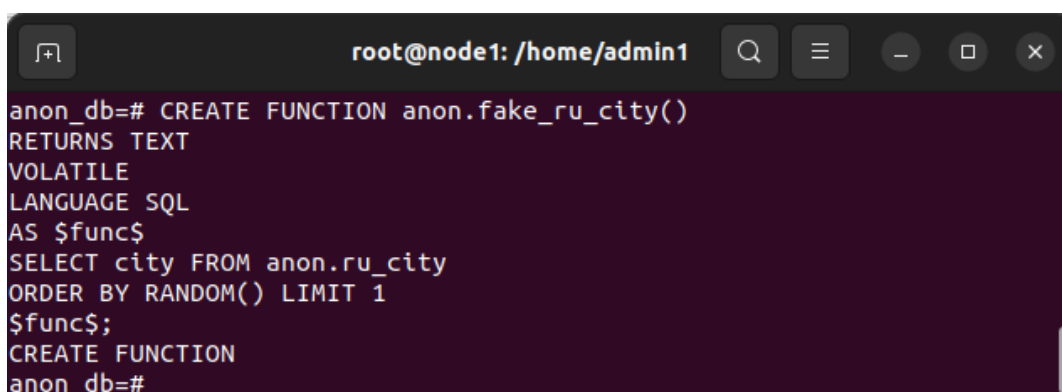


```
root@node1: /home/admin1
anon_db=# CREATE TABLE anon.ru_city
(id INT, city TEXT);
INSERT INTO anon.ru_city
VALUES
(1, 'Москва'),
(2, 'Санкт-Петербург'),
(3, 'Екатеринбург');
CREATE TABLE
INSERT 0 3
anon_db=# GRANT SELECT ON anon.ru_city TO maskuser;
GRANT
anon_db=#
```

Рисунок 3.1 – Создание таблицы-справочника значений

В данном руководстве в качестве примера предлагается создать функцию маскирования исходных данных, которая будет возвращать названия российских городов и применить созданную функцию для динамического маскирования таблицы:

```
CREATE FUNCTION anon.fake_ru_city()
RETURNS TEXT
VOLATILE
LANGUAGE SQL
AS $func$
SELECT city FROM anon.ru_city
ORDER BY RANDOM() LIMIT 1
$func$;
```



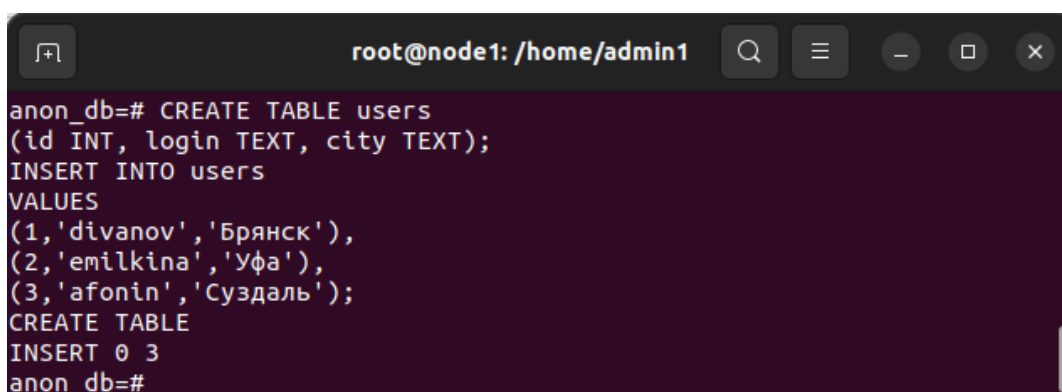
```
root@node1: /home/admin1
anon_db=# CREATE FUNCTION anon.fake_ru_city()
RETURNS TEXT
VOLATILE
LANGUAGE SQL
AS $func$
SELECT city FROM anon.ru_city
ORDER BY RANDOM() LIMIT 1
$func$;
CREATE FUNCTION
anon_db=#
```

Рисунок 3.2 – Создание функции anon.fake\_ru\_city

Создание таблицы, для которой будет применяться пользовательская функция anon.fake\_ru\_city:

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```
CREATE TABLE users
(id INT, login TEXT, city TEXT);
INSERT INTO users
VALUES
(1, 'divanov', 'Брянск'),
(2, 'emilkina', 'Уфа'),
(3, 'afonin', 'Суздаль');
```

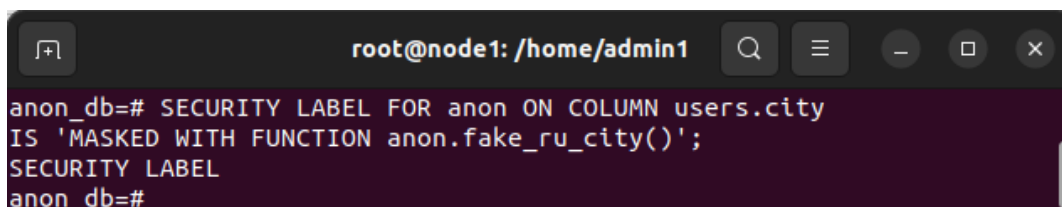


```
root@node1: /home/admin1
anon_db=# CREATE TABLE users
(id INT, login TEXT, city TEXT);
INSERT INTO users
VALUES
(1, 'divanov', 'Брянск'),
(2, 'emilkina', 'Уфа'),
(3, 'afonin', 'Суздаль');
CREATE TABLE
INSERT 0 3
anon_db=#
```

Рисунок 3.3 – Создание таблицы с исходными данными для маскирования

Далее необходимо применить правило маскирования к столбцу `users.city`, которое будет использовать пользовательскую функцию `anon.fake_ru_city`:

```
SECURITY LABEL FOR anon ON COLUMN users.city
IS 'MASKED WITH FUNCTION anon.fake_ru_city()';
SELECT anon.anonymize_column('users', 'city');
```



```
root@node1: /home/admin1
anon_db=# SECURITY LABEL FOR anon ON COLUMN users.city
IS 'MASKED WITH FUNCTION anon.fake_ru_city()';
SECURITY LABEL
anon_db=#
```

Рисунок 3.4 – Применение правил маскирования данных к таблице

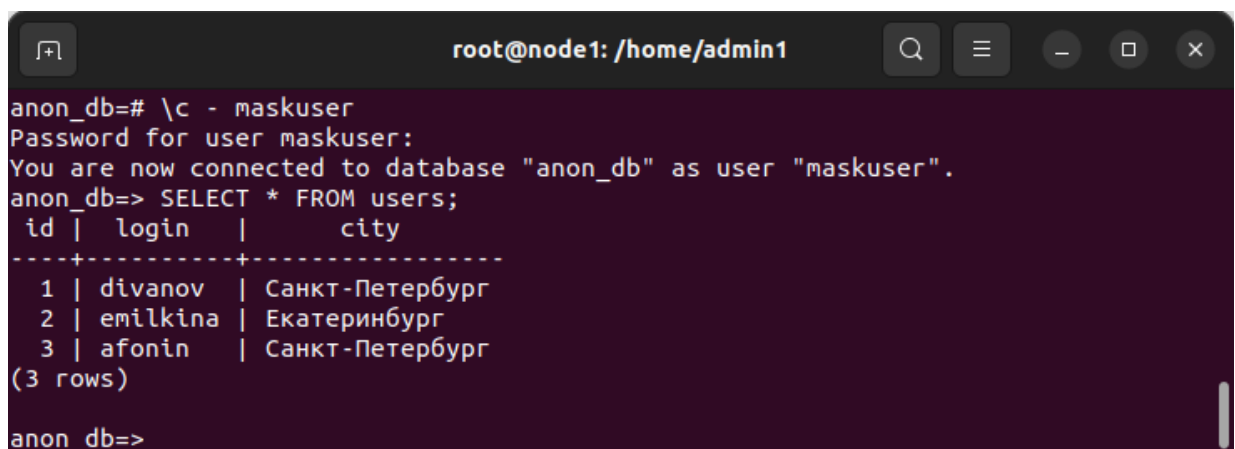
После активации правила маскирования необходимо убедиться в том, что данные из исходной таблицы `users` успешно маскируются:

```
\c - maskuser
You are now connected to database "anon_db" as user "maskuser".
```

```
SELECT * FROM users;

id | login | city
----+-----+-----
1 | divanov | Екатеринбург
2 | emilkina | Санкт-Петербург
3 | afonin | Москва
```

Как можно увидеть, данные в столбце `users.city` выводятся данные, которые маскируют исходные, подменяя значения из таблицы-справочника `anon.ru_city`.



```
root@node1: /home/admin1
anon_db=# \c - maskuser
Password for user maskuser:
You are now connected to database "anon_db" as user "maskuser".
anon_db=> SELECT * FROM users;
id | login | city
----+-----+-----
1 | divanov | Санкт-Петербург
2 | emilkina | Екатеринбург
3 | afonin | Санкт-Петербург
(3 rows)
anon_db=>
```

Рисунок 3.5 – Проверка маскирования исходных данных при использовании пользовательской функции `anon.fake_ru_city`

## 4. ПРИМЕР ИСПОЛЬЗОВАНИЯ

### 4.1. Статическое маскирование данных

Статический метод подменяет данные другими значениями.



Статический метод маскировки безвозвратно изменяет данные в таблицах или всей БД.

Статическое маскирование данных в случае, если для столбца таблицы есть ограничение внешнего ключа (например, `references city(name)`, где `city` - список городов) может привести к ошибке, если исходное значение заменяется из маскирующего набора данных, которого нет во внешнем ключе. Это следствие нарушения ограничений внешнего ключа. Если маскирующее значение совпадает с одним из элементов внешнего ключа, то статическое маскирование не приводит к возникновению ошибок.

В качестве примера будет представлен пример с несколькими функциями маскировки:

```
CREATE TABLE employees (  
    id SERIAL,  
    firstname TEXT,  
    lastname TEXT,  
    company TEXT,  
    postcode TEXT  
);  
  
INSERT INTO employees  
VALUES  
(111, 'Maria', 'Belova', 'Bank of Saratov', '405657'),  
(222, 'Pavel', 'Petrov', 'Head and Hands', '601245');
```

```
root@node1: /home/admin1
anon_db=# CREATE TABLE employees (
  id SERIAL,
  firstname TEXT,
  lastname TEXT,
  company TEXT,
  postcode TEXT
);
CREATE TABLE
anon_db=# INSERT INTO employees
VALUES
(111,'Maria','Belova','Bank of Saratov','405657'),
(222,'Pavel','Petrov','Head and Hands','601245');
INSERT 0 2
anon_db=#
```

Рисунок 4.1 – Создание таблицы с исходными данными

Таблица employees будет содержать информацию, представленную на рисунке 4.2.

```
root@node1: /home/admin1
anon_db=# SELECT FROM employees;
--
(2 rows)

anon_db=# SELECT * FROM employees;
 id | firstname | lastname |   company   | postcode
-----+-----+-----+-----+-----
 111 | Maria    | Belova  | Bank of Saratov | 405657
 222 | Pavel    | Petrov  | Head and Hands  | 601245
(2 rows)
anon_db=#
```

Рисунок 4.2 – Исходные данные в таблице employees

В качестве маскируемых данных будет выбран столбец postcode. Для этого определяется правило статической маскировки следующего вида:

```
SECURITY LABEL FOR anon ON COLUMN employees.postcode
IS 'MASKED WITH FUNCTION anon.partial(postcode,1,$$****$$,1)';
```

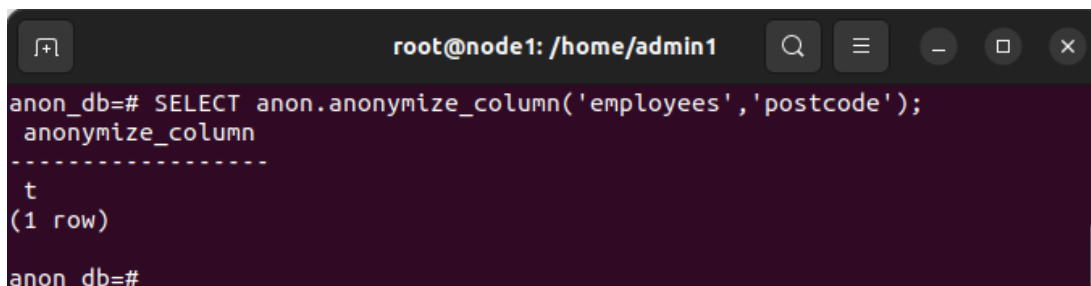
```
root@node1: /home/admin1
anon_db=# SECURITY LABEL FOR anon ON COLUMN employees.postcode
IS 'MASKED WITH FUNCTION anon.partial(postcode,1,$$****$$,1)';
SECURITY LABEL
anon_db=#
```

Рисунок 4.3 – Регистрация правила статического маскирования исходных данных таблицы employees

Данное правило заменяет знаками звездочка (\*) символы между первым и последним знаком в столбце postcode.

После определения правила статической маскировки она применяется к столбцу postcode с помощью следующего запроса:

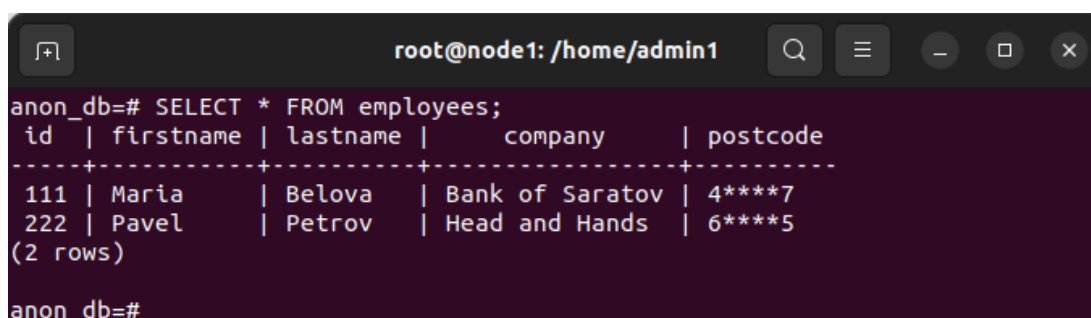
```
SELECT anon.anonymize_column('employees','postcode');
```



```
root@node1: /home/admin1
anon_db=# SELECT anon.anonymize_column('employees','postcode');
anon_db=#
anonize_column
-----
t
(1 row)
anon_db=#
```

Рисунок 4.4 – Применение правила anon.anonymize\_column статического маскирования исходных данных таблицы employees к столбцу postcode

С целью проверки статической маскировки можно повторно выполнить запрос на чтение данных из таблицы employees:



```
root@node1: /home/admin1
anon_db=# SELECT * FROM employees;
anon_db=#
id | firstname | lastname | company | postcode
-----+-----+-----+-----+-----
111 | Maria    | Belova  | Bank of Saratov | 4****7
222 | Pavel    | Petrov  | Head and Hands  | 6****5
(2 rows)
```

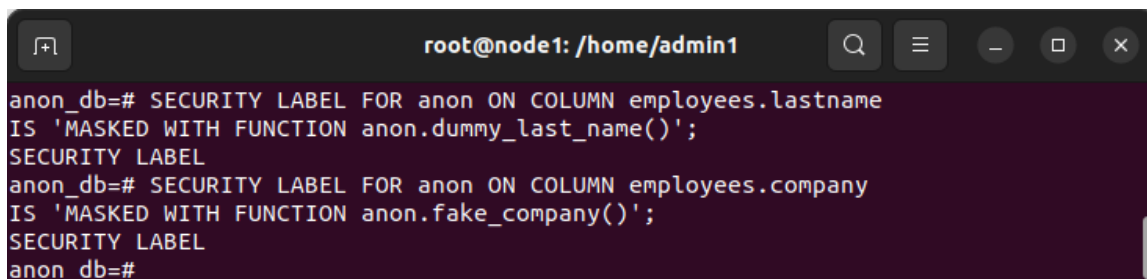
Рисунок 4.5 – Данные таблицы employees после использования статического маскирования к столбцу postcode

Как видно из результата выполнения запроса все значения в столбце postcode заменены на маскированные данные вида «х\*\*\*\*х» в соответствии с определенным ранее правилом статической маскировки.

Дополнительно можно выполнить статическую маскировку других столбцов таблицы employees определяя новые правила, например маскировка фамилии или названия компании (столбцы lastname и company):

```
SECURITY LABEL FOR anon ON COLUMN employees.lastname
IS 'MASKED WITH FUNCTION anon.dummy_last_name()';
SECURITY LABEL FOR anon ON COLUMN employees.company
```

```
IS 'MASKED WITH FUNCTION anon.fake_company()';
```

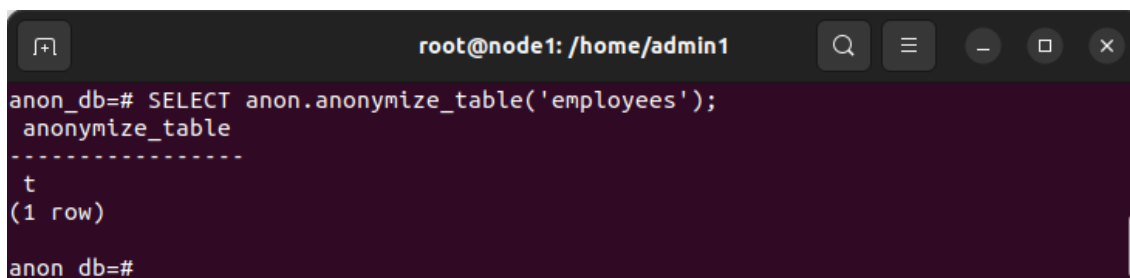


```
root@node1: /home/admin1
anon_db=# SECURITY LABEL FOR anon ON COLUMN employees.lastname
IS 'MASKED WITH FUNCTION anon.dummy_last_name()';
SECURITY LABEL
anon_db=# SECURITY LABEL FOR anon ON COLUMN employees.company
IS 'MASKED WITH FUNCTION anon.fake_company()';
SECURITY LABEL
anon_db=#
```

Рисунок 4.6 –Дополнительные правила маскирования исходных данных к таблице employees

После этого необходимо выполнить запросы на применение указанных правил к данным, содержащимся в таблице employees:

```
SELECT anon.anonymize_table('employees');
```



```
root@node1: /home/admin1
anon_db=# SELECT anon.anonymize_table('employees');
anonymize_table
-----
t
(1 row)
anon_db=#
```

Рисунок 4.7 –Дополнительные правила маскирования исходных данных к таблице employees

Либо статическая маскировка данных во всех таблицах БД в соответствии с определяемыми правилами:

```
SELECT anon.anonymize_database();
```

С целью проверки статической маскировки можно повторно выполнить запрос на чтение данных из таблицы employees:

```
SELECT * FROM employees;
```

Как видно и из рисунка 4.8, в таблице employees столбцы lastname, company содержат новые данные.

```

root@node1: /home/admin1
anon_db=# SELECT * FROM employees;
id | firstname | lastname | company | postcode
----+-----+-----+-----+-----
111 | Maria    | Kirlin  | Burch LLC | 4****7
222 | Pavel    | Bins    | Davis-Anderson | 6****5
(2 rows)
anon_db=#
  
```

Рисунок 4.8 – Данные таблицы employees после использования статического маскирования к столбцам lastname, company, postcode

## 4.2. Динамическое маскирование данных

При динамическом маскировании, данные для обычного пользователя представляются в исходном виде, а для пользователя с маской, соответственно в замаскированном.

Динамическое маскирование можно использовать кластере компонента «jaDog» при чтении с синхронной реплики.

Динамическое маскирование данных для секционированных таблиц устанавливается для самой секционированной таблицы, но не для секций этой таблицы.

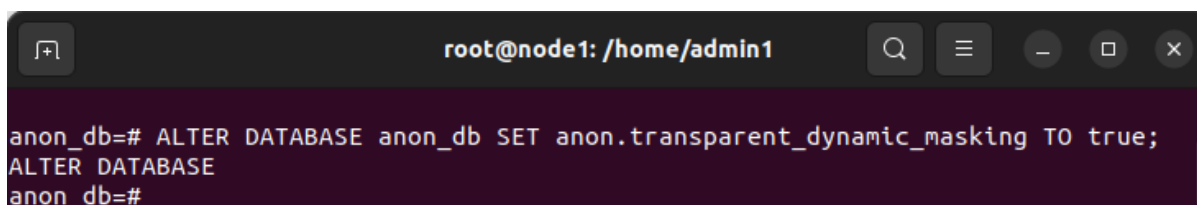
Динамическое маскирование в запросах вида `SELECT * FROM table WHERE colname = 'значение'`, в случае если colname имеет динамическое маскирование. То есть столбец может быть как проиндексирован, так и нет. Если в запросе, где условием выборки будет значение из замаскированного столбца, подставляется существующее значение из оригинальной таблицы, то результатом будет пустая выборка, так как оригинальные значения подменяются маскированными. Если же подставить значение из набора данных, которыми маскируется столбец, то получим ненулевую выборку данных. Индексация столбца с маскируемыми данными на конечный результат не влияет.

В данном подразделе будет рассмотрено несколько примеров использования динамической маскировки данных.

Включение динамического маскирования данных в общем случае производится с помощью выполнения следующего запроса:


```
ALTER DATABASE anon_db SET anon.transparent_dynamic_masking TO true;
```





```
root@node1: /home/admin1
anon_db=# ALTER DATABASE anon_db SET anon.transparent_dynamic_masking TO true;
ALTER DATABASE
anon_db=#
```

Рисунок 4.9 – Включение динамического маскирования исходных данных

 При использовании динамической маскировки исходных данных **не** используются следующие запросы:

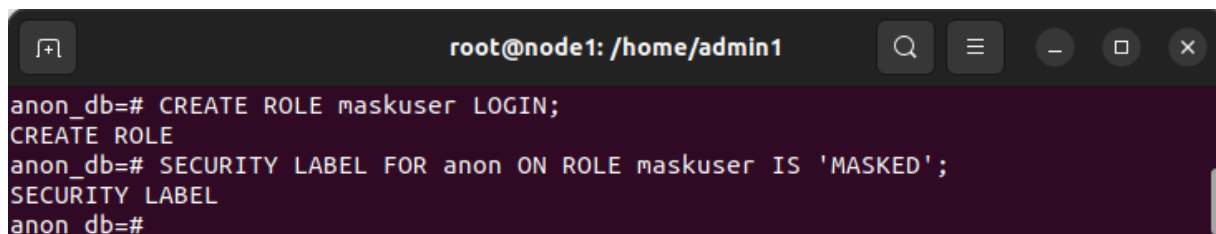
```
SELECT anon.anonymize_table('table_name');
SELECT anon.anonymize_database('database_name');
SELECT anon.anonymize_column('column_name', 'column_name');
```

#### 4.2.1. Правило маскирования данных пользователя с маской

Пользователь с маской создается так же как это указано в п.п. 3.8.

Применить правило маскирования данных к созданному пользователю;

```
SECURITY LABEL FOR anon ON ROLE maskuser IS 'MASKED';
```

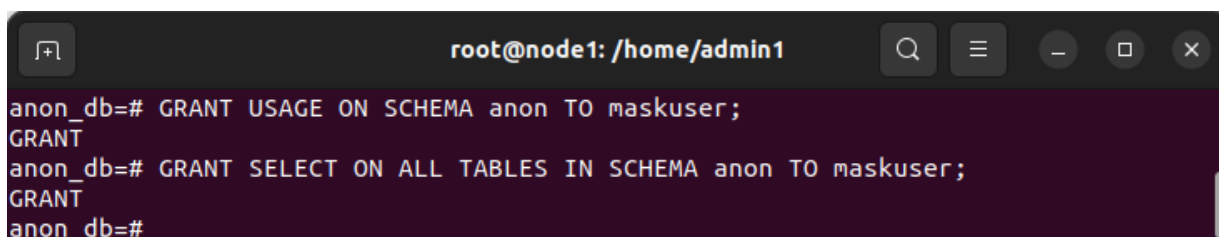


```
root@node1: /home/admin1
anon_db=# CREATE ROLE maskuser LOGIN;
CREATE ROLE
anon_db=# SECURITY LABEL FOR anon ON ROLE maskuser IS 'MASKED';
SECURITY LABEL
anon_db=#
```

Рисунок 4.10 – Создание пользователя с маской и применение для него правил маскирования исходных данных

Определить права доступа пользователя с маской к схеме БД и таблицам:

```
GRANT USAGE ON SCHEMA anon TO maskuser;
GRANT USAGE ON SCHEMA public TO maskuser;
GRANT SELECT ON ALL TABLES IN SCHEMA anon TO maskuser;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO maskuser;
```



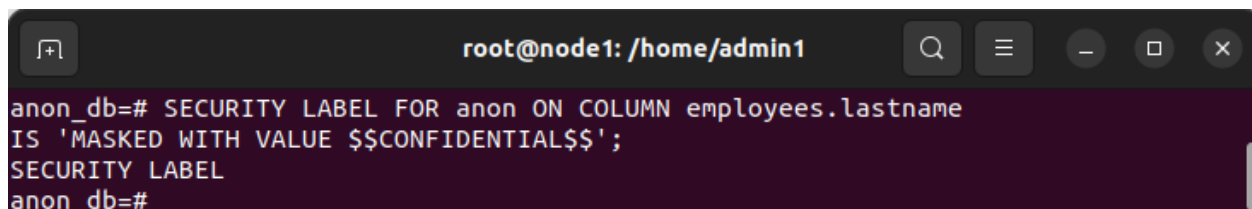
```
root@node1: /home/admin1
anon_db=# GRANT USAGE ON SCHEMA anon TO maskuser;
GRANT
anon_db=# GRANT SELECT ON ALL TABLES IN SCHEMA anon TO maskuser;
GRANT
anon_db=#
```

Рисунок 4.11 – Определить права доступа пользователя с маской к схеме БД и таблицам

#### 4.2.2. Замена данных запроса фиксированным значением

Правило замены данных запроса фиксированным значением активируется следующим образом:

```
SECURITY LABEL FOR anon ON COLUMN employees.lastname
IS 'MASKED WITH VALUE $$CONFIDENTIAL$$';
```



```
root@node1: /home/admin1
anon_db=# SECURITY LABEL FOR anon ON COLUMN employees.lastname
IS 'MASKED WITH VALUE $$CONFIDENTIAL$$';
SECURITY LABEL
anon_db=#
```

Рисунок 4.12 – Регистрация правила динамического маскирования исходных данных таблицы employees в столбце lastname

Другой вариант экранирования фиксированного значения выглядит следующим образом:

```
SECURITY LABEL FOR anon ON COLUMN employees.lastname
IS $$MASKED WITH VALUE 'CONFIDENTIAL'$$;
```

При выполнении запроса от имени пользователя maskuser в столбце lastname будет отображаться замаскированное значение «CONFIDENTIAL».

```

root@node1: /home/admin1
anon_db=# \c - maskuser
Password for user maskuser:
You are now connected to database "anon_db" as user "maskuser".
anon_db=> SELECT * FROM employees;
 id | firstname | lastname | company | postcode
-----+-----+-----+-----+-----
 111 | Maria     | CONFIDENTIAL | Poole, Brown and Robinson | 4****7
 222 | Pavel     | CONFIDENTIAL | Pratt-Gray | 6****5
(2 rows)
anon_db=>

```

Рисунок 4.13 – Данные таблицы employees после использования динамического маскирования к столбцам lastname при выполнении запроса от имени пользователя maskuser

#### 4.2.3. Замена символами исходных данных

Правило замены исходных данных запроса на символы активируется с помощью выполнения следующего SQL-запроса:

```

SECURITY LABEL FOR anon ON COLUMN employees.postcode
IS 'MASKED WITH FUNCTION anon.partial(postcode,1,$$****$$,1)';

```

```

root@node1: /home/admin1
anon_db=# SECURITY LABEL FOR anon ON COLUMN employees.postcode
IS 'MASKED WITH FUNCTION anon.partial(postcode,1,$$****$$,1)';
SECURITY LABEL
anon_db=#

```

Рисунок 4.14 – Регистрация правила динамического маскирования исходных данных таблицы employees в столбце postcode

#### 4.2.4. Замена исходных данных на случайное правдоподобное значение

В данном правиле маскирования используется функция dummy\_\*.

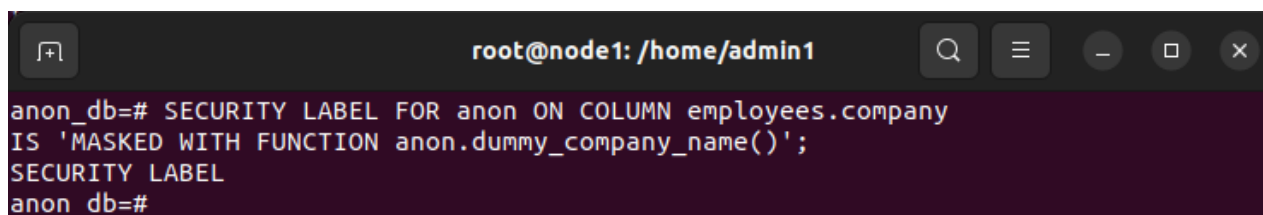
Функция dummy\_\* при каждом запросе данных подставляет разные правдоподобные значения.

Правило для замены данных запроса на случайное правдоподобное значение активируется с помощью выполнения следующим образом:

```

SECURITY LABEL FOR anon ON COLUMN employees.company
IS 'MASKED WITH FUNCTION anon.dummy_company_name()';

```



```
root@node1: /home/admin1
anon_db=# SECURITY LABEL FOR anon ON COLUMN employees.company
IS 'MASKED WITH FUNCTION anon.dummy_company_name()';
SECURITY LABEL
anon_db=#
```

Рисунок 4.15 – Регистрация правила динамического маскирования исходных данных таблицы employees в столбце company

#### 4.2.5. Замена исходных данных на псевдоним

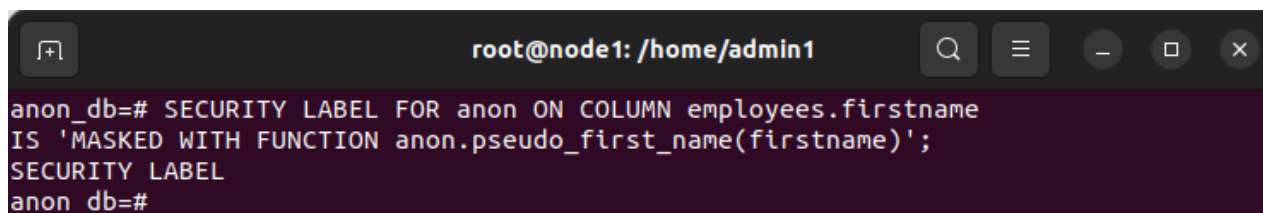
В данном правиле маскирования используется функция `pseudo_*`.

Функция `pseudo_*` подменяет исходные данные на псевдоним.

Псевдоним остается неизменным для каждого запроса данных. Иными словами, псевдонимизация — это способ защиты конфиденциальной информации, при котором, в отличие от замаскированных данных (функции `dummy_*` и `fake_*`), псевдонимизированные данные (`pseudo_*`) в некоторой степени «связаны» с реальными данными в БД.

Правило для замены данных запроса на псевдонимизированные данные активируется с помощью выполнения следующим образом:

```
SECURITY LABEL FOR anon ON COLUMN employees.firstname
IS 'MASKED WITH FUNCTION anon.pseudo_first_name(firstname)';
```



```
root@node1: /home/admin1
anon_db=# SECURITY LABEL FOR anon ON COLUMN employees.firstname
IS 'MASKED WITH FUNCTION anon.pseudo_first_name(firstname)';
SECURITY LABEL
anon_db=#
```

Рисунок 4.16 – Регистрация правила динамического маскирования исходных данных таблицы employees в столбце firstname

#### 4.2.6. Результат применения динамического маскирования

После определения одного или нескольких правил динамического маскирования данных таблица, к которой имеет доступ пользователь без маски, будет иметь вид, представленный на рисунке 4.17.

```

root@node1: /home/admin1
anon_db=# SELECT * FROM employees;
 id | firstname | lastname | company | postcode
-----+-----+-----+-----+-----
 111 | Maria     | Belova   | Bank of Saratov | 405657
 222 | Pavel     | Petrov   | Head and Hands  | 601245
(2 rows)
anon_db=#

```

Рисунок 4.17 – Исходные данные в таблице employees

В тоже время, для пользователя с маской эта таблица будет иметь вид, содержащий замаскированные данные (см. рисунок 4.18).

```

root@node1: /home/admin1
anon_db=# \c - maskuser
Password for user maskuser:
You are now connected to database "anon_db" as user "maskuser".
anon_db=> SELECT * FROM employees;
 id | firstname | lastname | company | postcode
-----+-----+-----+-----+-----
 111 | Abigail   | CONFIDENTIAL | Goldner and Grant LLC | 4****7
 222 | Steve     | CONFIDENTIAL | Weber and Hansen Inc  | 6****5
(2 rows)
anon_db=>

```

Рисунок 4.18 – Данные таблицы employees после использования динамического маскирования столбцов lastname, company и postcode при выполнении запроса от имени пользователя maskuser

При выполнении пользователем с маской повторного запроса результат будет содержать новые замаскированные данные (столбец company) и те же самые псевдонимизированные данные (столбец firstname):

```

SELECT * FROM employees;
 id | firstname | lastname | company | postcode
-----+-----+-----+-----+-----
 111 | Abigail   | CONFIDENTIAL | Erdman and Sons | 4****7
 222 | Steve     | CONFIDENTIAL | Zulauf and Abbott LLC | 6****5

```

#### 4.2.7. Отключение правила динамической маскировки

Для того чтобы отключить правило динамической маскировки необходимо выполнить следующий запрос:

```
SECURITY LABEL FOR anon ON COLUMN employees.lastname IS NULL;
```

#### 4.2.8. Отключение правила динамической маскировки для пользователя

Для того чтобы отключить правило динамической маскировки для пользователя с маской необходимо выполнить следующий запрос:

```
SECURITY LABEL FOR anon ON ROLE maskuser IS NULL;
```

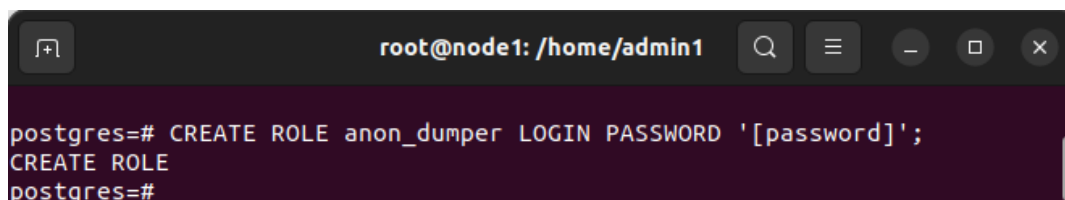
#### 4.3. Дамп с маскированными данными

Дамп с маскированными данными является удобным средством передачи копии реальной БД для разработки, тестирования или внешним организациям, без риска раскрытия персональных данных, коммерческой тайны и других критических данных.

##### 4.3.1. Создания пользователя с маской для работы с дампом

Создадим пользователя, который будет использовать функционал дампа с маскированными данными:

```
CREATE ROLE anon_dumper LOGIN PASSWORD '[password]';
```

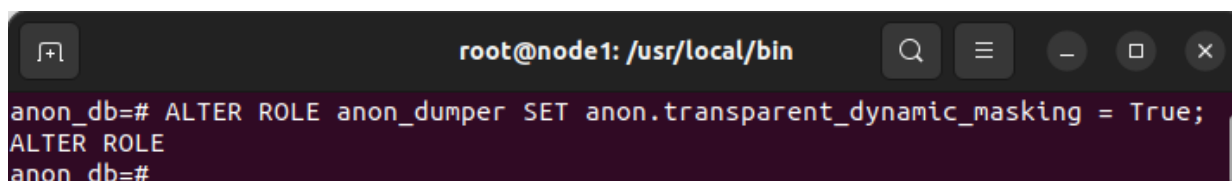


The screenshot shows a terminal window with the prompt 'root@node1: /home/admin1'. The user has entered the command 'CREATE ROLE anon\_dumper LOGIN PASSWORD '[password]';' and the PostgreSQL prompt 'postgres=#' is visible on the next line.

Рисунок 4.19 – Создание пользователя с маской для работы с дампом

Включение динамического маскирования данных с помощью выполнения следующего запроса:

```
ALTER ROLE anon_dumper SET anon.transparent_dynamic_masking = True;
```



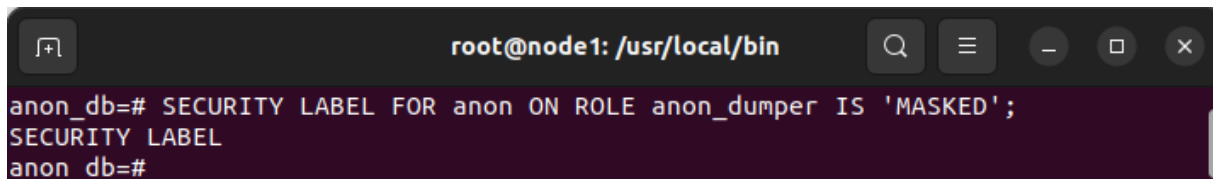
The screenshot shows a terminal window with the prompt 'root@node1: /usr/local/bin'. The user has entered the command 'ALTER ROLE anon\_dumper SET anon.transparent\_dynamic\_masking = True;' and the PostgreSQL prompt 'anon\_db=#' is visible on the next line.

Рисунок 4.20 – Включение динамического маскирования данных пользователя с маской для работы с дампом

Включение правил маскирования данных для пользователя:

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```
SECURITY LABEL FOR anon ON ROLE anon_dumper IS 'MASKED';
```

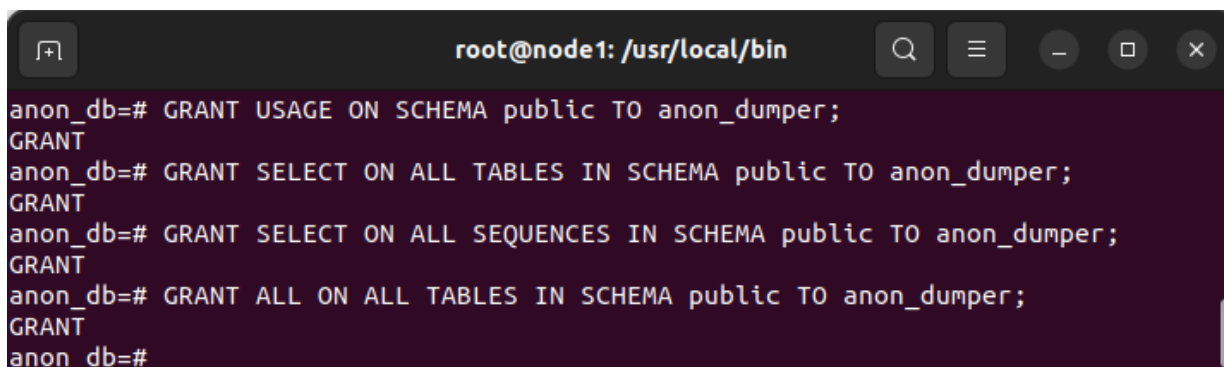


```
root@node1: /usr/local/bin
anon_db=# SECURITY LABEL FOR anon ON ROLE anon_dumper IS 'MASKED';
SECURITY LABEL
anon_db=#
```

Рисунок 4.21 – Включение правил маскирования данных для пользователя с маской для работы с дампом

Для предоставления доступа к схеме данных и таблицам необходимо выполнить следующие запросы:

```
GRANT USAGE ON SCHEMA public TO anon_dumper;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO anon_dumper;
GRANT SELECT ON ALL SEQUENCES IN SCHEMA public TO anon_dumper;
GRANT ALL ON ALL TABLES IN SCHEMA public TO anon_dumper;
```



```
root@node1: /usr/local/bin
anon_db=# GRANT USAGE ON SCHEMA public TO anon_dumper;
GRANT
anon_db=# GRANT SELECT ON ALL TABLES IN SCHEMA public TO anon_dumper;
GRANT
anon_db=# GRANT SELECT ON ALL SEQUENCES IN SCHEMA public TO anon_dumper;
GRANT
anon_db=# GRANT ALL ON ALL TABLES IN SCHEMA public TO anon_dumper;
GRANT
anon_db=#
```

Рисунок 4.22 – SQL-запросы для предоставления доступа к схеме данных и таблицам

#### 4.3.2. Запуск создания дампа с маскированными данными

Для того чтобы создать дамп, содержащий маскированные данные необходимо выполнить команду:

```
postgres@node1:~$ pg_dump [db_name] --user anon_dumper --no-
security-labels --file=test_anonymized.sql
```



Команда `pg_dump` выполняется от имени пользователя `postgres`.

В случае возникновения ошибок при создании дампа с маскированными данными необходимо обратиться к п.п. 6.1.

### 4.3.3. Проверка маскировки данных в дампе

После создания дампа необходимо убедиться в том, что данные в нем замаскированы (столбцы `firstname`, `company` и `postcode`):

```
COPY public.employees (id, firstname, lastname, company,
postcode) FROM stdin;
111      Abigail Belova Jaskolski and Sons 4****7
222      Steve   Petrov  Christiansen and Heathcote Group 6****5
```

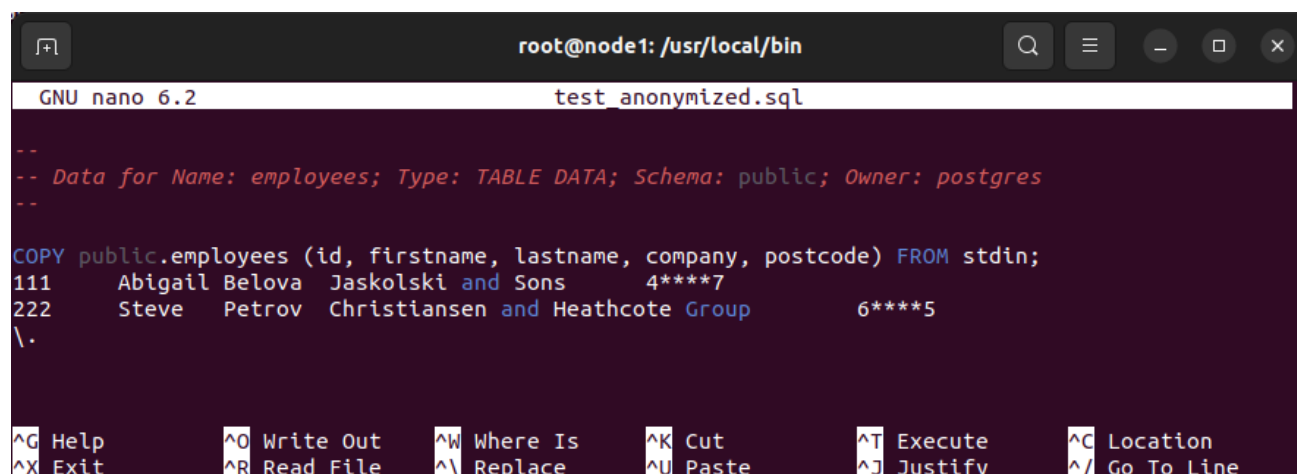


Рисунок 4.23 – Содержимое таблицы `employees` с замаскированными данными (столбцы `firstname`, `company` и `postcode`)

### 4.4. Представления с маскированными данными

Для описания работы с маскированным представлением необходимо создать таблицу `customer`:

```
CREATE TABLE customer (
    id SERIAL,
    full_name TEXT,
    birth DATE,
    employer TEXT,
    postcode TEXT,
    fk_shop INTEGER
```



);

Заполнить таблицу customer данными:

```
INSERT INTO customer
VALUES
(911,'Sergey Svetlakov','1977-12-12','STS', '750010',12),
(312,'Mikhail Porechenkov','1969-03-02','NTV', '620086',423);
```

```

root@node1: /usr/local/bin
anon_db=# CREATE TABLE customer(
  id SERIAL,
  full_name TEXT,
  birth DATE,
  employer TEXT,
  postcode TEXT,
  fk_shop INTEGER
);
CREATE TABLE
anon_db=# INSERT INTO customer
VALUES
(911,'Sergey Svetlakov','1977-12-12','STS', '750010',12),
(312,'Mikhail Porechenkov','1969-03-02','NTV', '620086',423);
INSERT 0 2
anon_db=#
  
```

Рисунок 4.24 – Создание таблицы customer и заполнение данными

Убедится в корректности внесенных данных:

```
SELECT * FROM customer;
```

id	full_name	birth	employer	postcode	fk_shop
911	Sergey Svetlakov	1977-12-12	STS	750010	12
312	Mikhail Porechenkov	1969-03-02	NTV	620086	423

```

root@node1: /usr/local/bin
anon_db=# SELECT * FROM customer;
 id |      full_name      |      birth      | employer | postcode | fk_shop
-----+-----+-----+-----+-----+-----
 911 | Sergey Svetlakov    | 1977-12-12     | STS      | 750010   |      12
 312 | Mikhail Porechenkov | 1969-03-02     | NTV      | 620086   |      423
(2 rows)
anon_db=#
  
```

Рисунок 4.25 – Содержимое таблицы customer

Создание представления с маскированием данных осуществляется при помощи следующего запроса:

```
CREATE MATERIALIZED VIEW masked_customer AS
SELECT
  id,
  'CONFIDENTIAL'::TEXT AS full_name,
  anon.generalize_daterange(birth,'decade') AS birth,
  employer,
  anon.partial(postcode,2,$$****$$,0) AS postcode,
  fk_shop
FROM customer;
```

The screenshot shows a terminal window with the prompt 'root@node1: /usr/local/bin'. The user has entered the SQL command to create the 'masked\_customer' view. The command is displayed in a monospaced font on a dark background. The command is: 'anon\_db=# CREATE MATERIALIZED VIEW masked\_customer AS SELECT id, 'CONFIDENTIAL'::TEXT AS full\_name, anon.generalize\_daterange(birth,'decade') AS birth, employer, anon.partial(postcode,2,\$\$\*\*\*\*\$\$,0) AS postcode, fk\_shop FROM customer;'. Below the command, the prompt 'anon\_db=#' is visible again, indicating the command has been executed.

Рисунок 4.26 – Создание представления masked\_customer

Вывод информации указанного представления masked\_customer будет содержать информацию, скрывающую исходные данные из таблицы customer:

```
SELECT * FROM masked_customer;
```

id	full_name	birth	employer	postcode	fk_shop
911	CONFIDENTIAL	[1970-01-01,1980-01-01)	STS	75****	12
312	CONFIDENTIAL	[1960-01-01,1970-01-01)	NTV	62****	423

```

root@node1: /usr/local/bin
anon_db=# SELECT * FROM masked_customer;
 id | full_name | birth | employer | postcode | fk_shop
-----+-----+-----+-----+-----+-----
 911 | CONFIDENTIAL | [1970-01-01,1980-01-01) | STS | 75**** | 12
 312 | CONFIDENTIAL | [1960-01-01,1970-01-01) | NTV | 62**** | 423
(2 rows)
anon_db=#

```

Рисунок 4.27 – Замаскированное содержимое представления masked\_customer

## 4.5. Маскирующие обертки данных

Принцип маскирующей обертки данных заключается в том, что СУБД используется в качестве «маскирующего прокси» с любым типом внешнего источника данных.

Таким образом возможно применять правила маскирования к данным, хранящимся в CSV-файлах, в другой СУБД, в хранилище NoSQL, в каталоге LDAP и так далее.

В приводимом примере источником данных будет файл журнала:

```

cat /tmp/app.log
2025-03-24 08:25:32,sarah,10.0.0.45,view_dashboard
2025-03-24 09:15:00,mike,172.16.0.89,update_profile
2025-03-24 09:30:45,emma,192.168.2.200,download_report

```

Здесь /tmp/app.log – путь и название файла журнала с данными.

### 4.5.1. Подготовка к работе с внешним источником данных

Для загрузки данных из внешнего источника необходимо воспользоваться расширением file\_fdw:

```

CREATE EXTENSION IF NOT EXISTS file_fdw;
CREATE SERVER external_files FOREIGN DATA WRAPPER file_fdw;

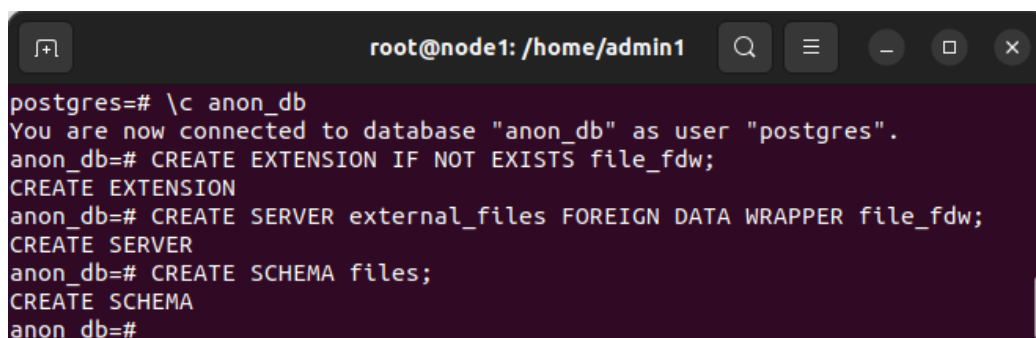
```

Создать новую схему данных для работы с внешним источником. В примере таблица будет называться files:

```

CREATE SCHEMA files;

```



```
root@node1: /home/admin1
postgres=# \c anon_db
You are now connected to database "anon_db" as user "postgres".
anon_db=# CREATE EXTENSION IF NOT EXISTS file_fdw;
CREATE EXTENSION
anon_db=# CREATE SERVER external_files FOREIGN DATA WRAPPER file_fdw;
CREATE SERVER
anon_db=# CREATE SCHEMA files;
CREATE SCHEMA
anon_db=#
```

Рисунок 4.28 – Создание и настройка расширения file\_fdw, создание схемы данных files

#### 4.5.2. Создание внешней таблицы

Столбцы внешней таблицы соответствуют содержимому файла журнала.

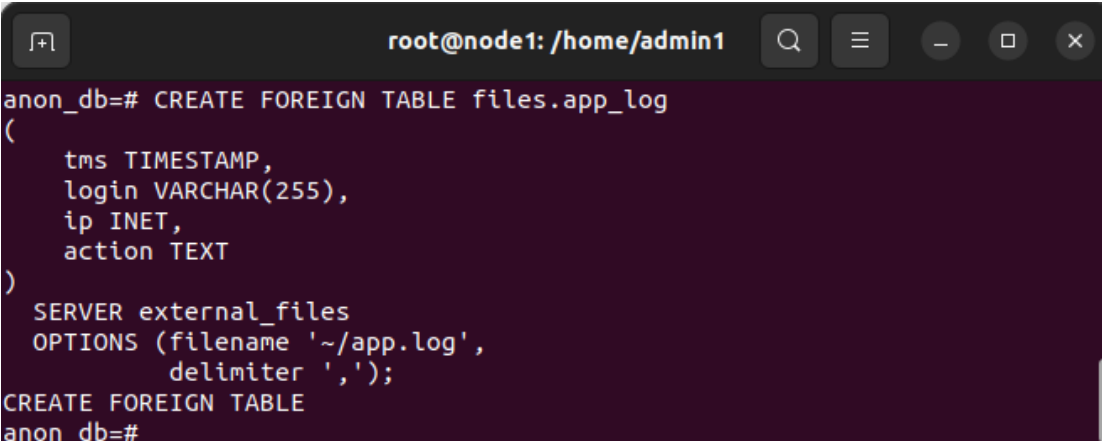
Для работы с внешним источником данных необходимо создать таблицу и указать источник внешних данных:

```
CREATE FOREIGN TABLE files.app_log
(
    tms TIMESTAMP,
    login VARCHAR(255),
    ip INET,
    action TEXT
)
SERVER external_files
OPTIONS (filename '/tmp/app.log',
        delimiter ',');
```

Опция filename указывает путь и название внешнего источника данных для маскирования.



К файлу внешнего источника данных должен быть обеспечен доступ для пользователя postgres.

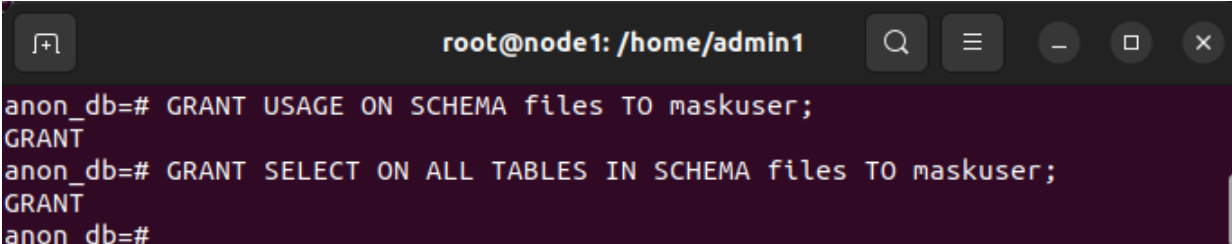


```
root@node1: /home/admin1
anon_db=# CREATE FOREIGN TABLE files.app_log
(
    tms TIMESTAMP,
    login VARCHAR(255),
    ip INET,
    action TEXT
)
SERVER external_files
OPTIONS (filename '~/app.log',
        delimiter ',');
CREATE FOREIGN TABLE
anon_db=#
```

Рисунок 4.29 – Создание внешней таблицы files.app\_log

Для предоставления доступа к схеме данных и таблицам необходимо выполнить следующие запросы:

```
GRANT USAGE ON SCHEMA files TO maskuser;
GRANT SELECT ON ALL TABLES IN SCHEMA files TO maskuser;
```



```
root@node1: /home/admin1
anon_db=# GRANT USAGE ON SCHEMA files TO maskuser;
GRANT
anon_db=# GRANT SELECT ON ALL TABLES IN SCHEMA files TO maskuser;
GRANT
anon_db=#
```

Рисунок 4.30 – Предоставление доступа к схеме данных и внешней таблице files.app\_log

#### 4.5.3. Определение правил маскирования данных внешнего источника

В соответствии с тем к каким полям необходимо применить маскирование определяются правила маскировки:

```
SECURITY LABEL FOR anon ON COLUMN files.app_log.login
IS 'MASKED WITH VALUE $$CONFIDENTIAL$$';
SECURITY LABEL FOR anon ON COLUMN files.app_log.ip
IS 'MASKED WITH FUNCTION anon.dummy_ipv4()';
```

```
root@node1: /home/admin1
anon_db=# SECURITY LABEL FOR anon ON COLUMN files.app_log.login
IS 'MASKED WITH VALUE $$CONFIDENTIAL$$';
SECURITY LABEL
anon_db=# SECURITY LABEL FOR anon ON COLUMN files.app_log.ip
IS 'MASKED WITH FUNCTION anon.dummy_ipv4()';
SECURITY LABEL
anon_db=#
```

Рисунок 4.31 – Определение правил маскирования данных внешней таблицы files.app\_log  
Альтернативный вариант экранирования значения CONFIDENTIAL:

```
SECURITY LABEL FOR anon ON COLUMN files.app_log.login
IS $$MASKED WITH VALUE 'CONFIDENTIAL'$$;
```

#### 4.5.4. Проверка маскирования данных из внешнего источника

Для проверки маскирования данных из внешнего источника необходимо выполнить запрос к таблице:

```
\c - maskuser
You are now connected to database "anon_db" as user "maskuser".
SELECT * FROM files.app_log;

      tms          |      login      |      ip          |      action
-----+-----+-----+-----
2025-03-24 08:25:32 | CONFIDENTIAL    | 152.90.250.121   | view_dashboard
2025-03-24 09:15:00 | CONFIDENTIAL    | 177.29.143.147   | update_profile
2025-03-24 09:30:45 | CONFIDENTIAL    | 204.141.38.236   | download_report
```

В результате запроса данные, которые были определены ранее правилами маскирования искажены или содержат запись о конфиденциальности.

```
root@node1: /home/admin1
anon_db=# \c - maskuser
Password for user maskuser:
You are now connected to database "anon_db" as user "maskuser".
anon_db=> SELECT * FROM files.app_log;
      tms      |      login      |      ip      |      action
-----+-----+-----+-----
2025-03-24 08:25:32 | CONFIDENTIAL | 184.20.121.95 | view_dashboard
2025-03-24 09:15:00 | CONFIDENTIAL | 112.175.35.111 | update_profile
2025-03-24 09:30:45 | CONFIDENTIAL | 215.119.80.232 | download_report
(3 rows)
anon_db=>
```

Рисунок 4.32 –Маскированные данные внешней таблицы files.app\_log при просмотре от имени пользователя с маской

## 5. УДАЛЕНИЕ РАСШИРЕНИЯ

Расширение компонента и библиотека удаляются с помощью следующих SQL-команд:

```
postgres=# DROP EXTENSION anon CASCADE;  
postgres=# ALTER DATABASE anon_db RESET  
session_preload_libraries;
```

После этого необходимо убедиться в том, что расширение компонента успешно удалено из СУБД при помощи команды:

```
\dx
```

В конфигурационном файле СУБД postgresql.conf закомментировать (при помощи знака #) или удалить из параметра shared\_preload\_libraries название библиотеки компонента:

```
#shared_preload_libraries = 'anon'
```

Перезагрузить СУБД и убедиться в корректном статусе работы службы:

```
# systemctl restart jatoba-6  
# systemctl status jatoba-6
```



## 6. ВОЗМОЖНЫЕ ОШИБКИ

### 6.1. Ошибка при выгрузке дампа с маскированными данными из БД в режиме «только чтение»

Предварительные условия:

- Установленные СУБД «Jatoba» и расширение ja\_anonymizer в БД anon\_db;
- Создан пользователь с маской (здесь и далее см. п.п. 4.3.1):

```
CREATE ROLE anon_dumper LOGIN PASSWORD '[password]';
```

- Включено динамическое маскирование данных роли:

```
ALTER ROLE anon_dumper SET anon.transparent_dynamic_masking = true;
```

- Применено правило маскирования данных к созданному пользователю:

```
SECURITY LABEL FOR anon ON ROLE anon_dumper IS 'MASKED';
```

- Создана таблица и активировано правило маскирования данных таблицы с генерацией последовательных значений (sequence):

```
SECURITY LABEL FOR anon ON COLUMN customer.id IS $$MASKED WITH  
FUNCTION anon.random_id_int()$$;
```

После создания таблицы, содержащей данные для маскирования, при выгрузке в дампы может возникать ошибка следующего вида:

```
./pg_dump anon_db --user anon_dumper --no-security-labels --  
file=masked_data.sql  
  
pg_dump: error: Dumping the contents of table "customer"  
failed: PQgetResult() failed.  
  
pg_dump: detail: Error message from server: ОШИБКА: в  
транзакции в режиме "только чтение" нельзя выполнить nextval()  
  
CONTEXT: SQL-функция "random_id_int", оператор 1  
  
pg_dump: detail: Command was: COPY public.customer (id,  
full_name, birth, city, postcode, fk_shop) TO stdout
```

Возникновение данной ошибки связано с установленным для БД режимом «только чтение» (read-only). В связи с этим, при выполнении выгрузки дампа из БД функции маскирования данных, которые выполняют изменение пользовательских и/или служебных данных (например, счетчики), не будут доступны к использованию.

## ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

CSV	–	Текстовый формат, предназначенный для представления табличных данных
LDAP	–	Протокол прикладного уровня для доступа к службе каталогов
NoSQL	–	Обозначение класса систем управления базами данных (СУБД) и существенно отличающихся от традиционных реляционных СУБД
SQL	–	Structured Query Language
БД	–	База данных
ОС	–	Операционная система
СУБД	–	Система управления базами данных

## Лист регистрации изменений

Дата внесения изм: