

ЗАЩИЩЕННАЯ СИСТЕМА УПРАВЛЕНИЯ  
БАЗАМИ ДАННЫХ «ЈАТОВА»

Руководство по настройке. Часть 23.  
Поддержка лексографического идентификатора.  
Компонент «pg\_ulid»

643.72410666.00067-07 98 01-23

Листов 22

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

## АННОТАЦИЯ

В документе приведены сведения, необходимые для установки и эксплуатации компонента «pg\_ulid» (далее по тексту – «компонент» или «pg\_ulid»), предназначенного для поддержки типа данных ULID.

Настоящее руководство предназначено для администраторов СУБД.



Все примеры в данном документе приведены для СУБД «Jatoba» версии ядра 5.x, для других версий все шаги выполняются аналогично, разница состоит в именах директорий.

Например, СУБД «Jatoba» версии 6.x по умолчанию устанавливается в директорию:

- ОС Windows – «C:\Program Files\GIS\Jatoba\6\bin»;
- ОС Linux – «/usr/jatoba-6/bin».

Версия компонента — 0.0.1-15



### **Важная информация**

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

Степени важности примечаний, применяемые в документе:



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



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

## СОДЕРЖАНИЕ

1. Назначение компонента.....	4
1.1. Условия применения.....	4
2. Установка и настройка.....	5
2.1. Установка компонента «pg_ulid» на ОС Windows.....	5
2.2. Установка компонента «pg_ulid» в ОС GNU/Linux.....	5
2.3. Настройка конфигурационного файла «postgresql.conf» .....	6
2.4. Установка расширения «ulid».....	7
3. Функциональные возможности компонента.....	8
3.1.1. Функция gen_ulid() .....	9
3.1.2. Конструкции приведения типа ulid к тексту и наоборот.....	9
3.1.3. Конструкции приведения типа ulid к штампу времени.....	10
3.1.4. Использование нового типа данных «ulid» в таблицах пользователя .....	11
3.1.5. Сравнение двух ulid-значений.....	13
3.1.6. Использование нового типа данных в индексах пользователя .....	15
4. Удаление компонента .....	19
Термины и определения .....	20
Перечень сокращений.....	21

## 1. НАЗНАЧЕНИЕ КОМПОНЕНТА

Компонент «pg\_ulid» предназначен для поддержки в СУБД типа данных ULID (128 бит) с полноценным использованием в таблицах, запросах и индексах.

Компонент обладает функцией монотонности в случае генерации большого количество случайных значений в рамках одной миллисекунды, в формате до 80 бит.

### Например

```
ulid() -- 01BX5ZZKBKACTAV9WEVGEMMVR Y  
ulid() -- 01BX5ZZKBKACTAV9WEVGEMMVR Z  
ulid() -- 01BX5ZZKBKACTAV9WEVGEMMV S0
```

В таком случае временная часть ULID будет постоянной, а случайная часть будет генерироваться в виде последовательности значений. Если же генерация значений происходит в разные миллисекунды реального времени, то случайная часть будет именно случайной. Свойство порядка будет обеспечивать временная часть.

### 1.1. Условия применения

Компонент «pg\_ulid» может использоваться с СУБД «Jatoba» версий 5.x и выше, под управлением операционных систем Windows и GNU/Linux.



В текущей реализации не поддерживается управление компонентом «pg\_ulid» из компонента пользовательского веб-интерфейса для администраторов «Jatoba data safe» Ограничений по совместимости с другими компонентами нет.

## 2. УСТАНОВКА И НАСТРОЙКА

Установка компонента должна производиться от имени пользователя, обладающего административными привилегиями в системе. Данный компонент штатным образом может быть установлен только с СУБД «Jatoba» (см. документ «Защищенная система управления базами данных «Jatoba». Руководство по установке).

### 2.1. Установка компонента «pg\_ulid» на ОС Windows

Компонент устанавливается в составе СУБД «Jatoba» под управлением ОС Windows при первичной установке.

Компонент будет установлен при «Полной» или «Выборочной установке».

### 2.2. Установка компонента «pg\_ulid» в ОС GNU/Linux

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

Установку 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 jatoba5-pg-ulid
```

– 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 jatoba5-pg_ulid
```

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 jatoba5-pg_ulid
```

– openSUSE также распространяется в виде rpm-пакетов, но использует собственный пакетный менеджер zypper, для нее команда установки выглядит следующим образом:

```
zypper install jatoba5-pg_ulid
```

Установка компонента в составе других версий СУБД «Jatoba» осуществляется аналогично. Отличие будет только в номере версии СУБД, в составе которой он распространяется. Например, jatoba6-pg\_ulid и т.п.

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.

### 2.3. Настройка конфигурационного файла «postgresql.conf»

В разделе «Shared Library Preloading», для последующей загрузки расширения, установить параметр:

```
shared_preload_libraries = 'ulid'
```



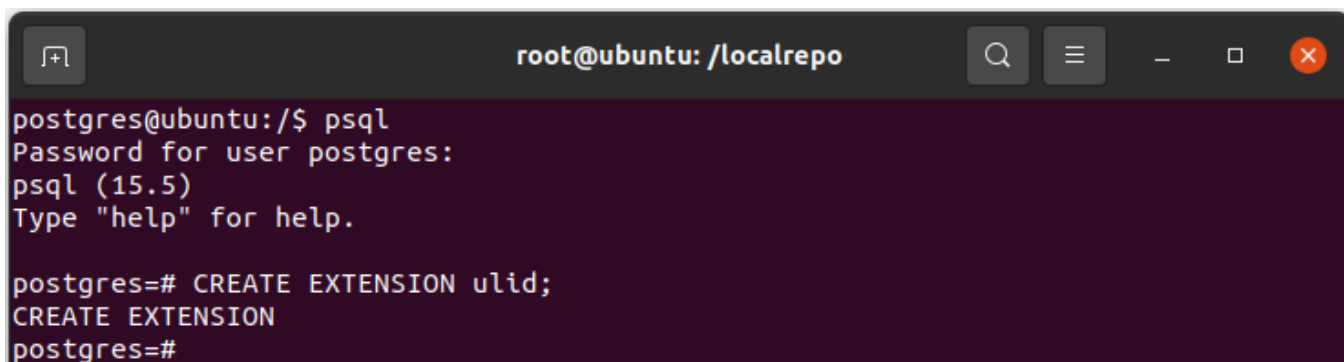
Рисунок 2.1 – Параметр загрузки расширения

Для применения параметров потребуется перезапустить СУБД.

## 2.4. Установка расширения «ulid»

После перезагрузки СУБД станет доступной установка расширения «ulid».

```
CREATE EXTENSION ulid;
```



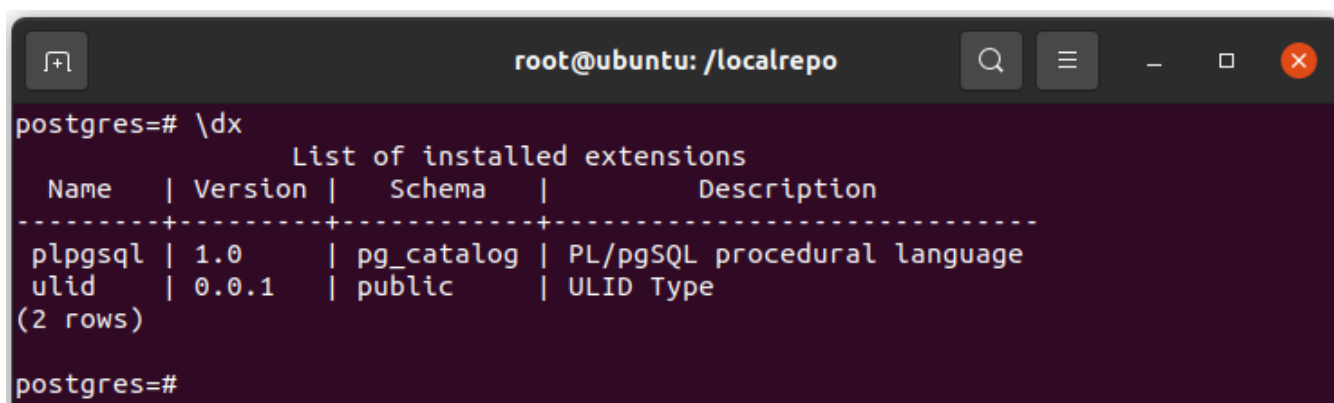
```
root@ubuntu: /localrepo

postgres@ubuntu:/$ psql
Password for user postgres:
psql (15.5)
Type "help" for help.

postgres=# CREATE EXTENSION ulid;
CREATE EXTENSION
postgres=#
```

Рисунок 2.2 – Команда установки расширения «ulid»

В результате выполненных действий установится расширение «ulid».



```
root@ubuntu: /localrepo

postgres=# \dx
              List of installed extensions
  Name      | Version | Schema  | Description
-----+-----+-----+-----
 plpgsql    | 1.0     | pg_catalog | PL/pgSQL procedural language
 ulid       | 0.0.1   | public   | ULID Type
(2 rows)

postgres=#
```

Рисунок 2.3 – Вывод установленных расширений

### 3. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ КОМПОНЕНТА

После установки расширения в составе выбранной базы данных пользователю становятся доступны следующие объекты БД, вносимые расширением:

1) Тип данных ulid

Размер поля 128 бит; временная часть = старшие 48 бит; случайная часть = младшие 80 бит.

Временная часть отвечает за порядок следования случайных значений.

Случайная часть за генерацию уникальных значений (вероятность коллизии  $10^{-24}$ ).

2) Операторы и функции для работы с типом данных ulid:

- функция `gen_ulid()` (см .п. 3.1.1);
- конструкции приведения типа ulid к тексту и наоборот (см .п. 3.1.2)
- конструкции приведения типа ulid к штампу времени (см. п. 3.1.3);
- сравнение двух ulid-значений (см. п. 3.1.5);

3) Использование нового типа данных в таблицах пользователя (см. п. 3.1.4);

4) Использование нового типа данных в индексах пользователя в том числе и в составных индексах указанных типов (см. п. 3.1.6);

5) Использование нового типа данных в представлениях и материализованных представлениях в качестве полей результата запроса

6) Использование нового типа данных в триггерных процедурах (поддерживается язык PL/pgSQL) в качестве элемента, который можно вставлять/изменять/удалять или читать значение этого элемента в составе кода триггерной процедуры.



Функция `gen ulid()` возвращает случайный идентификатор.

Может применяться в SQL запросах в выражениях там, где допустим вызов функции.

```
SELECT gen_ulid();

SELECT ... FROM ... WHERE ... gen ulid() ...
```

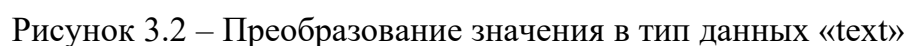


Конструкции приведения типа «ulid» к тексту и наоборот имеет синтаксис SQL-команды:

```
SELECT ulid_field::text ....
SELECT 'XXXXXXXX...':ulid ...
```

Преобразовать значение в текст:

```
SELECT ('7ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ'::ulid)::text;
```



### Преобразовать значение в тип данных «ulid»:

```
SELECT ('7ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ'::ulid);
```

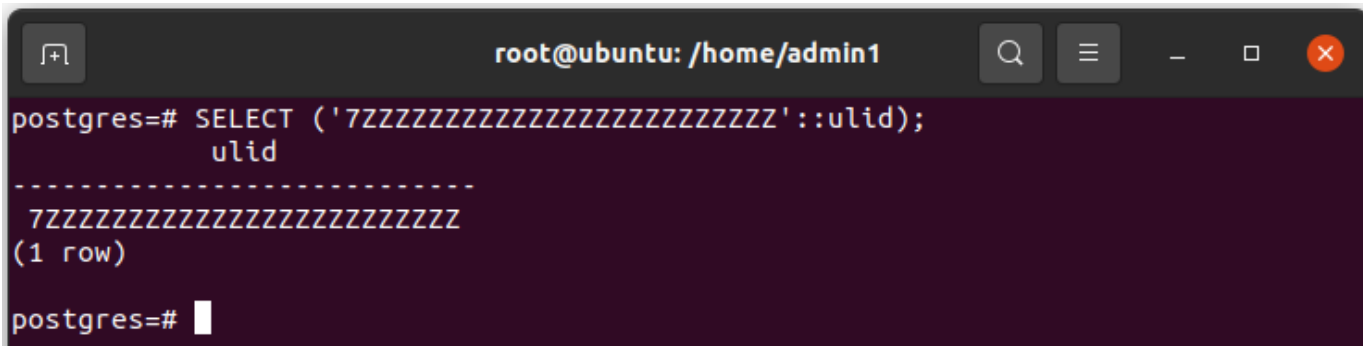


Рисунок 3.3 - Преобразование значение в тип данных «ulid»

### 3.1.3. Конструкции приведения типа ulid к штампу времени

```
SELECT ulid_field::timestamp ...
SELECT ts field::ulid ...
```

## Например

Выполнить преобразование в timestamp:

```
SELECT ('7ZZZZZZZZZZZZZZZZZZZZZZZZZZZZ':::ulid)::timestamp;  
SELECT ('7ZZZZZZZZZZZZZZZZZZZZZZZZZZZZ':::ulid)::timestamp;  
SELECT ('01H55TNAQ96WPSWE6WZRCH9G0C':::ulid)::timestamp;
```

```
postgres=# SELECT ('7ZZZZZZZZZZZZZZZZZZZZZZZZZZ'::ulid)::timestamp;
          timestamp
-----
10889-08-02 05:31:50.655
(1 row)

postgres=# SELECT ('7ZZZZZZZZZZZZZZZZZZZZZZZZZZ'::ulid)::timestamp;
          timestamp
-----
10889-08-02 05:31:50.655
(1 row)

postgres=# SELECT ('01H55TNAQ96WPSWE6WZRCH9G0C'::ulid)::timestamp;
          timestamp
-----
2023-07-12 19:53:43.401
(1 row)

postgres=#
```

Выполнить преобразование в ulid:

```
SELECT ('2023-07-12 19:53:43.401'::timestamp::ulid);
```

```
root@ubuntu: /home/admin1
postgres=# SELECT('2023-07-12 19:53:43.401'::timestamp::ulid);
          ulid
-----
 01H55TNAQ9W28C6P7TN8EV0E20
(1 row)

postgres=#
```

Рисунок 3.5 – Преобразование значения из формата timestamp в формат ulid

### 3.1.4. Использование нового типа данных «ulid» в таблицах пользователя

Компонент «pg\_ulid» вводит новый тип данных «ulid» в таблицах пользователя. При этом используется синтаксис SQL-команды:

```
CREATE TABLE t ( id ulid ... );
```

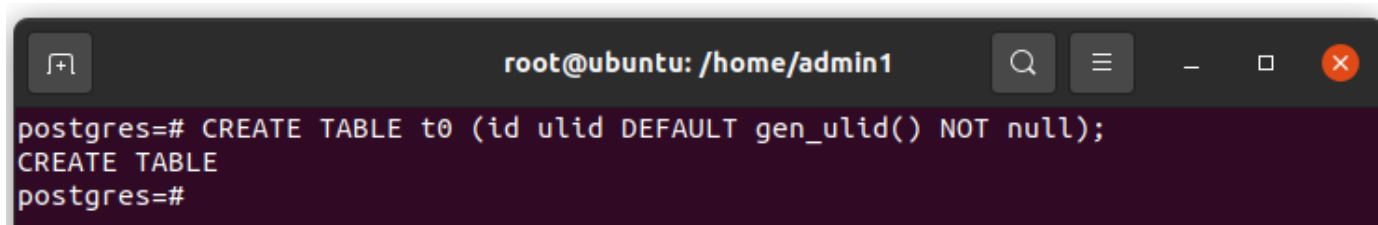
Допустимы и другие конструкции создания и изменений полей через оператора ALTER TABLE.

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

## Например

Создать таблицу:

```
# CREATE TABLE t0 (id ulid DEFAULT gen_ulid() NOT null);
```

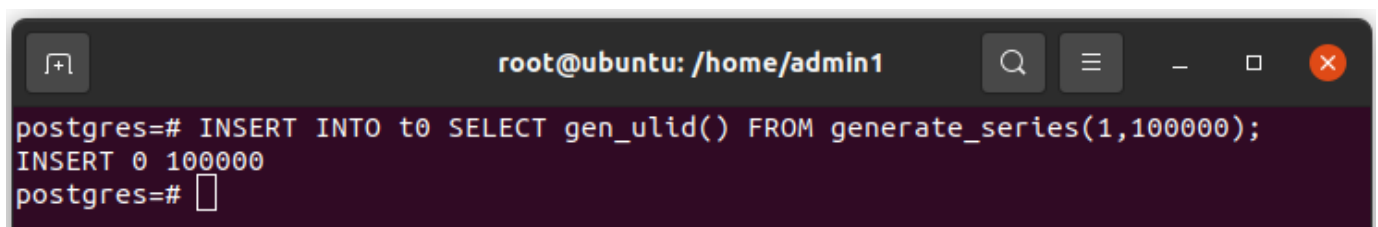


A terminal window titled 'root@ubuntu: /home/admin1' with search, menu, and window control icons. The command 'CREATE TABLE t0 (id ulid DEFAULT gen\_ulid() NOT null);' is entered and executed successfully, returning 'postgres=#'.

Рисунок 3.6 – Создание таблицы с типом данных «ulid»

Вставить произвольный ulid:

```
# INSERT INTO t0 SELECT gen_ulid() FROM  
generate_series(1,100000);
```

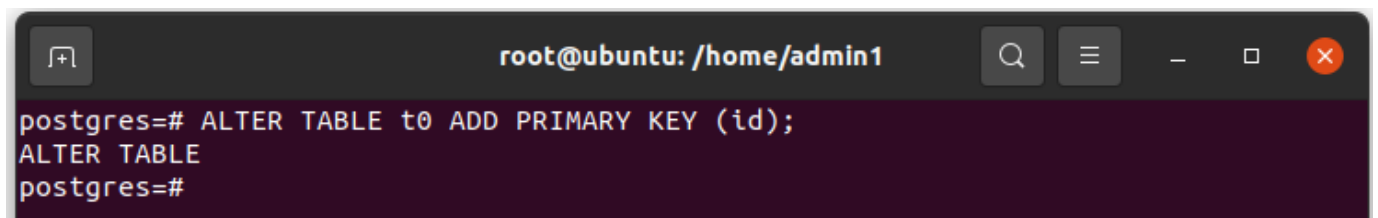


A terminal window titled 'root@ubuntu: /home/admin1' with search, menu, and window control icons. The command 'INSERT INTO t0 SELECT gen\_ulid() FROM generate\_series(1,100000);' is entered and executed successfully, returning 'postgres=#'.

Рисунок 3.7 – Вставка произвольных значений

Указать первичный ключ:

```
ALTER TABLE t0 ADD PRIMARY KEY (id);
```



A terminal window titled 'root@ubuntu: /home/admin1' with search, menu, and window control icons. The command 'ALTER TABLE t0 ADD PRIMARY KEY (id);' is entered and executed successfully, returning 'postgres=#'.

Рисунок 3.8 – Установка первичного ключа

Вывести содержание таблицы:

```
SELECT id FROM public.t0;
```

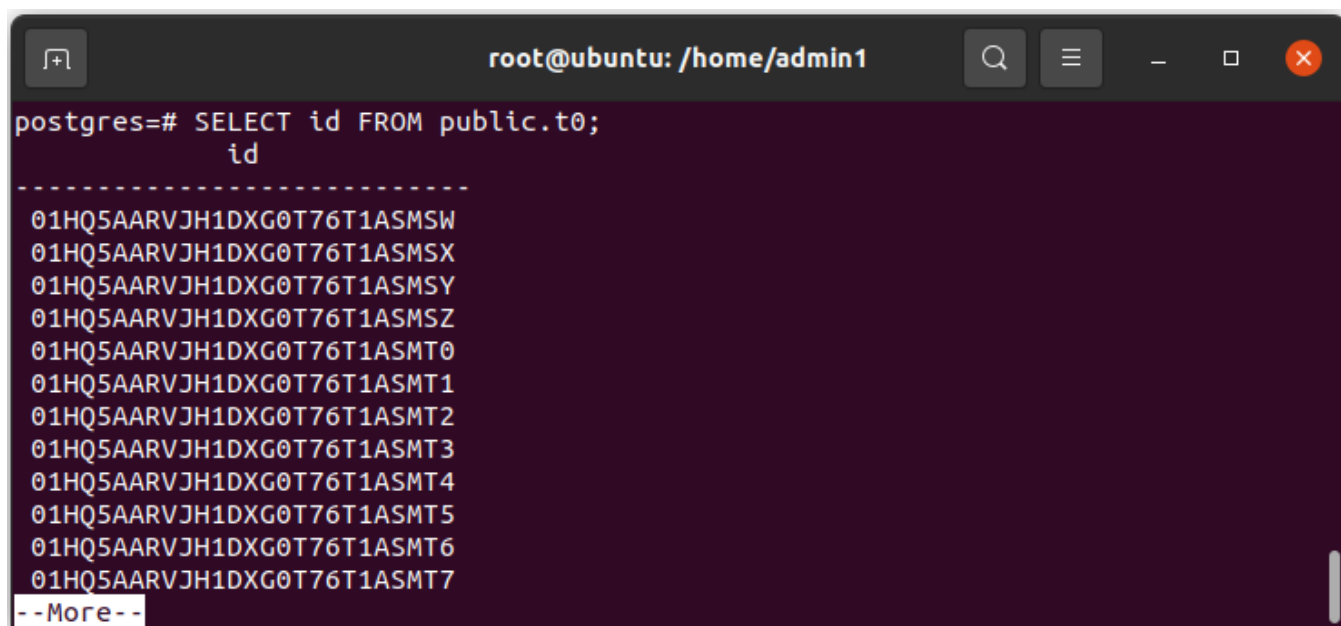
A screenshot of a terminal window with a dark background. The title bar shows 'root@ubuntu: /home/admin1'. The terminal displays a PostgreSQL prompt 'postgres=#' followed by the command 'SELECT id FROM public.t0;'. The output shows a list of 15 UUIDs, each on a new line, starting with '01HQ5AARVJH1DXG0T76T1ASMSW' and ending with '01HQ5AARVJH1DXG0T76T1ASMT7'. At the bottom of the output, there is a prompt '--More--'.

Рисунок 3.9 – Вывод содержания таблицы

Тип данных «ulid» допускает применение по отношению к нему всех типов ограничений таблицы: PRIMARY KEY, NULL/NOT NULL, UNIQUE, CHECK, REFERENCE, GENERATED

```
CREATE TABLE t ( id ulid PRIMARY KEY, ... );
```

и другие конструкции создания и изменений полей через ALTER TABLE.

### 3.1.5. Сравнение двух ulid-значений

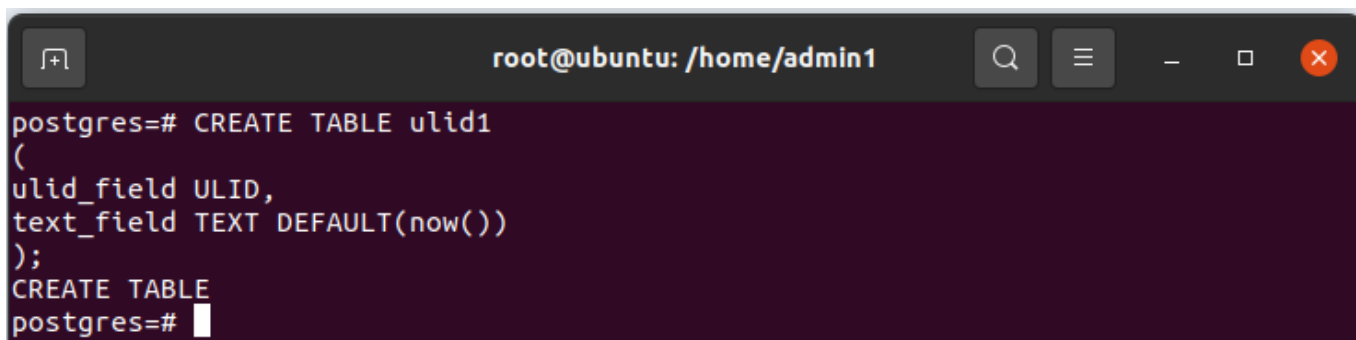
Данные выражения возвращают тип boolean и могут применяться в SQL запросах в контексте выражений:

- = оператор равенства двух ulid;
- <> оператор неравенства двух ulid;
- > больше;
- >= больше или равно;
- < меньше;
- <= меньше или равно.

## Например

Создать таблицу:

```
# CREATE TABLE ulid1
(
  ulid_field ULID,
  text_field TEXT DEFAULT(now())
);
```



The screenshot shows a terminal window with the title 'root@ubuntu: /home/admin1'. The command 'postgres=# CREATE TABLE ulid1 ( ulid\_field ULID, text\_field TEXT DEFAULT(now()) );' has been entered and executed successfully. The prompt has changed to 'postgres=#'.

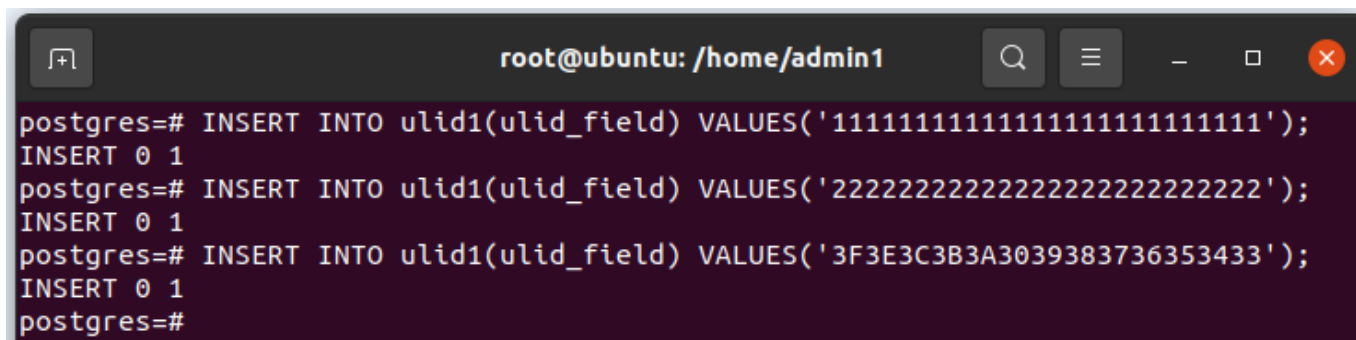
Рисунок 3.10 – Создание таблицы

Вставить данные:

```
# INSERT INTO ulid1(ulid_field)
VALUES('11111111111111111111111111111111');

# INSERT INTO ulid1(ulid_field)
VALUES('22222222222222222222222222222222');

# INSERT INTO ulid1(ulid_field)
VALUES('3F3E3C3B3A3039383736353433');
```



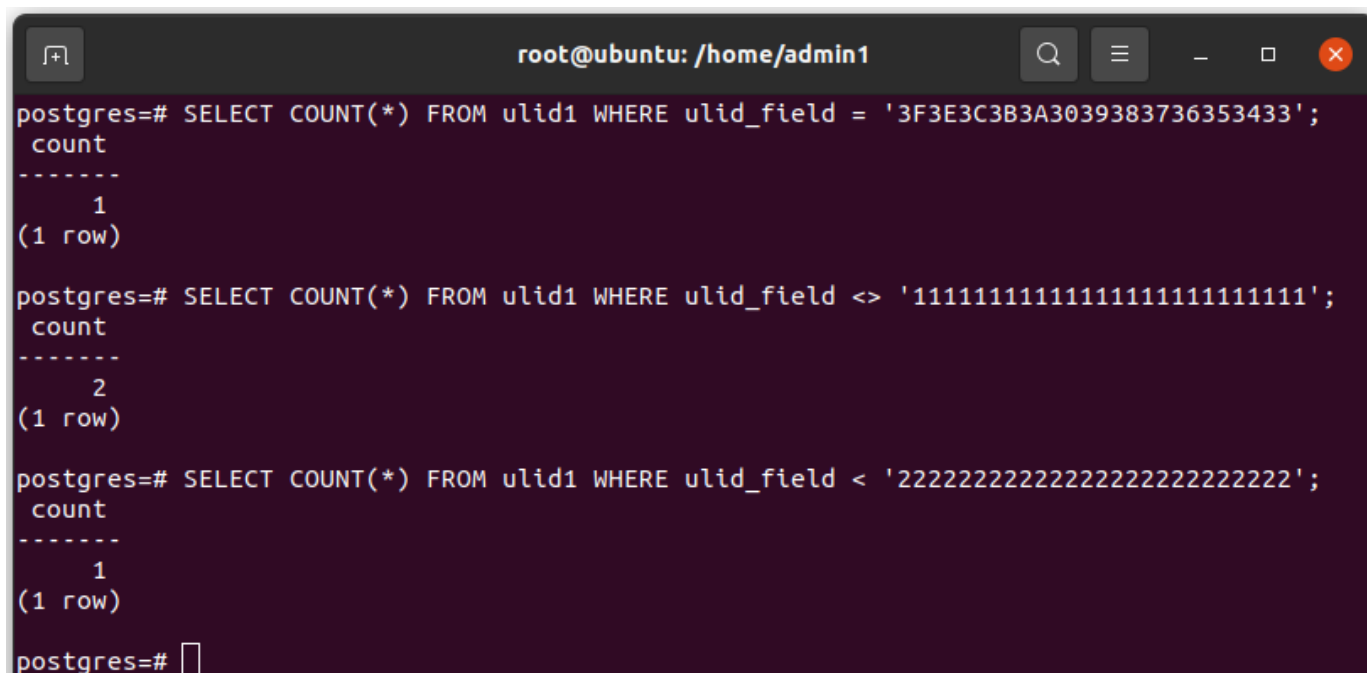
The screenshot shows a terminal window with the title 'root@ubuntu: /home/admin1'. Three INSERT commands have been entered and executed successfully. The first command is 'postgres=# INSERT INTO ulid1(ulid\_field) VALUES('11111111111111111111111111111111');', followed by 'INSERT 0 1'. The second command is 'postgres=# INSERT INTO ulid1(ulid\_field) VALUES('22222222222222222222222222222222');', followed by 'INSERT 0 1'. The third command is 'postgres=# INSERT INTO ulid1(ulid\_field) VALUES('3F3E3C3B3A3039383736353433');', followed by 'INSERT 0 1'. The prompt is now 'postgres=#'.

Рисунок 3.11 – Вставка значений в таблицу

Выполнить проверку операторов сравнения:

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

```
# SELECT COUNT(*) FROM ulid1 WHERE ulid_field =  
'3F3E3C3B3A3039383736353433';  
  
# SELECT COUNT(*) FROM ulid1 WHERE ulid_field <>  
'1111111111111111111111111111';  
  
# SELECT COUNT(*) FROM ulid1 WHERE ulid_field <  
'2222222222222222222222222222';
```



The screenshot shows a terminal window with the title 'root@ubuntu: /home/admin1'. It displays three PostgreSQL queries and their results. The first query counts rows where ulid\_field equals a specific UUID, returning 1. The second query counts rows where ulid\_field is not equal to a string of 16 ones, returning 2. The third query counts rows where ulid\_field is less than a string of 16 twos, returning 1. The terminal prompt is 'postgres=#'.

```
postgres=# SELECT COUNT(*) FROM ulid1 WHERE ulid_field = '3F3E3C3B3A3039383736353433';  
count  
-----  
1  
(1 row)  
  
postgres=# SELECT COUNT(*) FROM ulid1 WHERE ulid_field <> '1111111111111111111111111111';  
count  
-----  
2  
(1 row)  
  
postgres=# SELECT COUNT(*) FROM ulid1 WHERE ulid_field < '2222222222222222222222222222';  
count  
-----  
1  
(1 row)  
postgres=#
```

Рисунок 3.12 – Проверка операторов сравнения

### 3.1.6. Использование нового типа данных в индексах пользователя

Компонент «pg\_ulid» вводит новый тип данных «ulid» в индексах пользователя. В настоящее время поддерживаются типы индексов «BTREE» и «HASH».

Пользователь может создать индекс по полю типа данных «ulid» используя синтаксис SQL-команды:

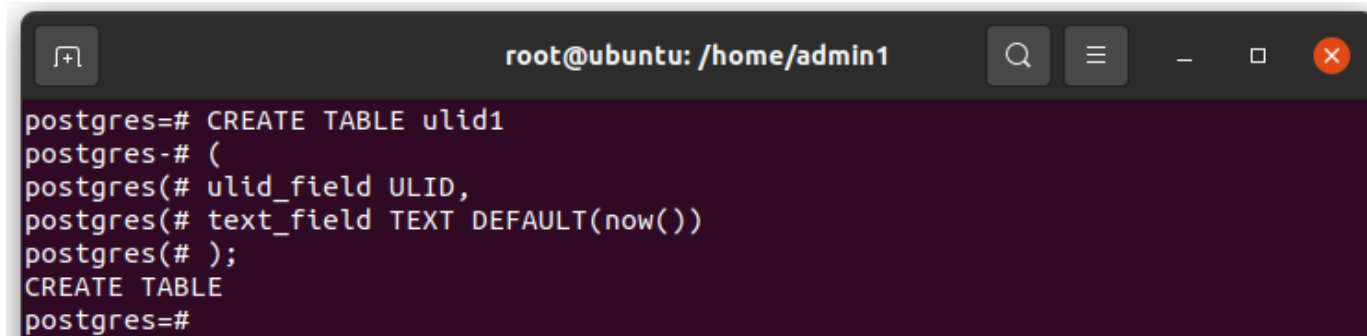
```
CREATE INDEX my_idx on my_table [ USING btree | hash ] (id);
```

#### Например

Создать таблицу:

```
# CREATE TABLE ulid1  
(
```

```
ulid_field ULID,  
text_field TEXT DEFAULT(now())  
);
```



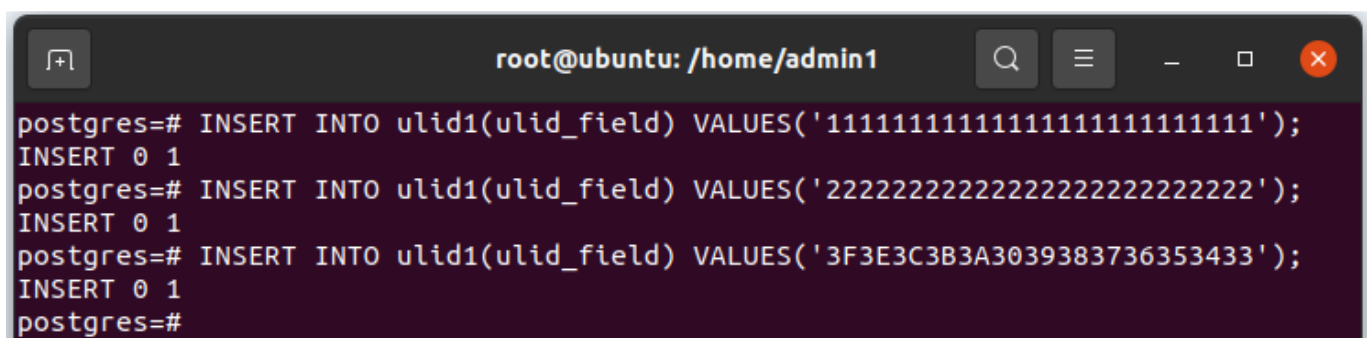
A terminal window titled 'root@ubuntu: /home/admin1' showing the execution of SQL commands to create a table. The commands are: 'CREATE TABLE ulid1', '(', 'ulid\_field ULID,', 'text\_field TEXT DEFAULT(now())', and ');'. The prompt 'postgres=#' is visible at the start of each line.

```
postgres=# CREATE TABLE ulid1  
postgres=# (  
postgres=# ulid_field ULID,  
postgres=# text_field TEXT DEFAULT(now())  
postgres=# );  
CREATE TABLE  
postgres=#
```

Рисунок 3.13 – Создание таблицы

Вставить данные:

```
# INSERT INTO ulid1(ulid_field)  
VALUES('11111111111111111111111111111111');  
  
# INSERT INTO ulid1(ulid_field)  
VALUES('22222222222222222222222222222222');  
  
# INSERT INTO ulid1(ulid_field)  
VALUES('3F3E3C3B3A3039383736353433');
```



A terminal window titled 'root@ubuntu: /home/admin1' showing the execution of three INSERT statements. Each statement is followed by the output 'INSERT 0 1'. The prompt 'postgres=#' is visible at the start of each line.

```
postgres=# INSERT INTO ulid1(ulid_field) VALUES('11111111111111111111111111111111');  
INSERT 0 1  
postgres=# INSERT INTO ulid1(ulid_field) VALUES('22222222222222222222222222222222');  
INSERT 0 1  
postgres=# INSERT INTO ulid1(ulid_field) VALUES('3F3E3C3B3A3039383736353433');  
INSERT 0 1  
postgres=#
```

Рисунок 3.14 – Вставка данных

Сгенерировать записи:

```
INSERT INTO ulid1 (ulid_field) SELECT gen_ulid() FROM  
generate_series(1,100000);
```



```
root@ubuntu: /home/admin1

postgres=# INSERT INTO ulid1 (ulid_field) SELECT gen_ulid() FROM generate_series(1,100000);
INSERT 0 100000
postgres=#
```

Рисунок 3.15 – Генерирование записей

Создать индексы «BTREE», «HASH»:

```
# CREATE INDEX ulid1_btree ON ulid1 USING BTREE (ulid_field);
# CREATE INDEX ulid1_hash ON ulid1 USING HASH (ulid_field);
```

```
root@ubuntu: /home/admin1

postgres=# CREATE INDEX ulid1_btree ON ulid1 USING BTREE (ulid_field);
CREATE INDEX
postgres=# CREATE INDEX ulid1_hash ON ulid1 USING HASH (ulid_field);
CREATE INDEX
postgres=#
```

Рисунок 3.16 – Создание индексов

Вывести план запроса:

```
EXPLAIN SELECT * FROM public.ulid1 WHERE ulid_field =
'11111111111111111111111111111111';
```

```
root@ubuntu: /home/admin1

postgres=# EXPLAIN SELECT * FROM public.ulid1 WHERE ulid_field = '11111111111111111111111111111111';
               QUERY PLAN
-----
Index Scan using ulid1_btree on ulid1  (cost=0.42..8.44 rows=1 width=45)
   Index Cond: (ulid_field = '11111111111111111111111111111111'::ulid)
(2 rows)

postgres=#
```

Рисунок 3.17 – Вывод плана запроса

Вывести содержание таблицы:

```
SELECT * FROM public.ulid1;
```

```
root@ubuntu: /home/admin1

postgres=# SELECT * FROM public.ulid1;
      ulid_field      |      text_field
-----+-----
11111111111111111111 | 2024-02-21 05:26:44.112945-08
22222222222222222222 | 2024-02-21 05:26:50.836461-08
3F3E3C3B3A3039383736353433 | 2024-02-21 05:26:57.564873-08
01HSFYXAN1HDK1MBYHDXK50TP1 | 2024-03-21 00:34:20.55237-07
01HSFYXAN1HDK1MBYHDXK50TP2 | 2024-03-21 00:34:20.55237-07
01HSFYXAN1HDK1MBYHDXK50TP3 | 2024-03-21 00:34:20.55237-07
01HSFYXAN1HDK1MBYHDXK50TP4 | 2024-03-21 00:34:20.55237-07
```

Рисунок 3.18 – Вывод содержания таблицы с индексами BTREE, HASH  
Индексы успешно созданы и используются.

## 4. УДАЛЕНИЕ КОМПОНЕНТА

Удаление компонента проводится в несколько этапов.

Удалить пакет:

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

```
apt-get remove jatoba<version>-pg-ulid
```

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

```
yum remove jatoba<version>-pg_ulid
```

После чего необходимо убрать загрузку модуля из конфигурационного файла «postgresql.conf», поставив знак #, или удалить имя расширения из списка расширений.

```
#shared_preload_libraries = 'ulid'
```

Расширение может быть удалено из базы данных пользователя SQL-командой:

```
DROP EXTENSION ulid;
```

Но при условии, что тип данных ulid нигде более в базе данных не используется.

В противном случае будет ошибка о наличии зависимостей - это нормально, пользователь сам должен разрешить эту зависимость (удалить использование типа ulid или конвертировать такие поля в text).

Расширение может быть удалено каскадным методом из базы данных пользователя SQL-командой:

```
DROP EXTENSION ulid CASCADE;
```

## ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

**ULID (Universally Unique Lexicographically Sortable Identifier)** - это уникальный, глобально уникальный, лексикографически сортируемый идентификатор. ULID представляет собой строку, состоящую из двух частей: даты и времени создания объекта и уникального случайного числа. Обе части преобразуются в числовое значение, что позволяет сортировать объекты в лексикографическом порядке. ULID используется для уникальной идентификации объектов без использования централизованных серверов или баз данных, обеспечивая децентрализованное управление и хранение данных.

## **ПЕРЕЧЕНЬ СОКРАЩЕНИЙ**

SQL	–	Structured Query Language
БД	–	База данных
ОС	–	Операционная система
СУБД	–	Система управления базами данных

[illegible]

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