

ЗАЩИЩЕННАЯ СИСТЕМА УПРАВЛЕНИЯ
БАЗАМИ ДАННЫХ «ЈАТОВА»

Руководство по настройке. Часть 10.
Поддержка лексографического идентификатора.
Компонент «pg_ulid»

643.72410666.00067-07 98 01-10

Листов 29

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 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» в ОС GNU/Linux	5
2.2. Настройка конфигурационного файла «postgresql.conf»	6
2.3. Установка расширения «ulid».....	6
3. Функциональные возможности компонента.....	8
3.1. Компонент pg_ulid	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. Применение идентификатора UUIDv7.....	19
4.1. Миграция с ULID к UUIDv7	19
4.2. ULID используется в качестве первичного ключа	19
4.2.1. Создание тестовой таблицы	19
4.2.2. Изменение ограничений первичного ключа в таблице	23
4.3. ULID используется в качестве внешнего ключа	23
4.3.1. Создание тестовой таблицы	23
4.3.2. Изменение ограничений внешнего ключа	24
5. Удаление компонента	26
Термины и определения	27
Перечень сокращений.....	28

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 операционной системы 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» 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 jatoba18-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 jatoba18-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 jatoba18-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.

2.2. Настройка конфигурационного файла «postgresql.conf»

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

```
shared_preload_libraries = 'ulid'
```



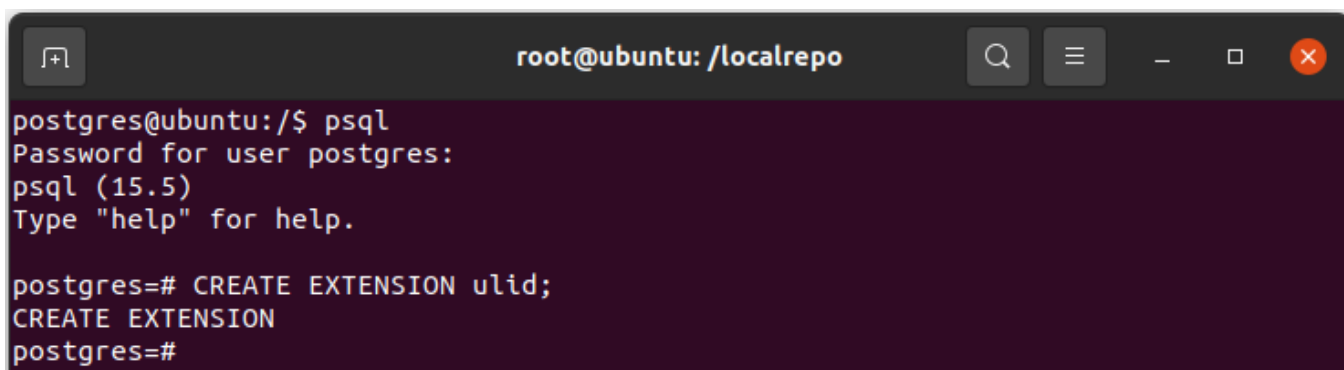
Рисунок 2.1 – Параметр загрузки расширения

Для применения параметров потребуется перезапустить СУБД.

2.3. Установка расширения «ulid»

После перезагрузки СУБД станет доступной установка расширения «ulid».

```
CREATE EXTENSION ulid;
```



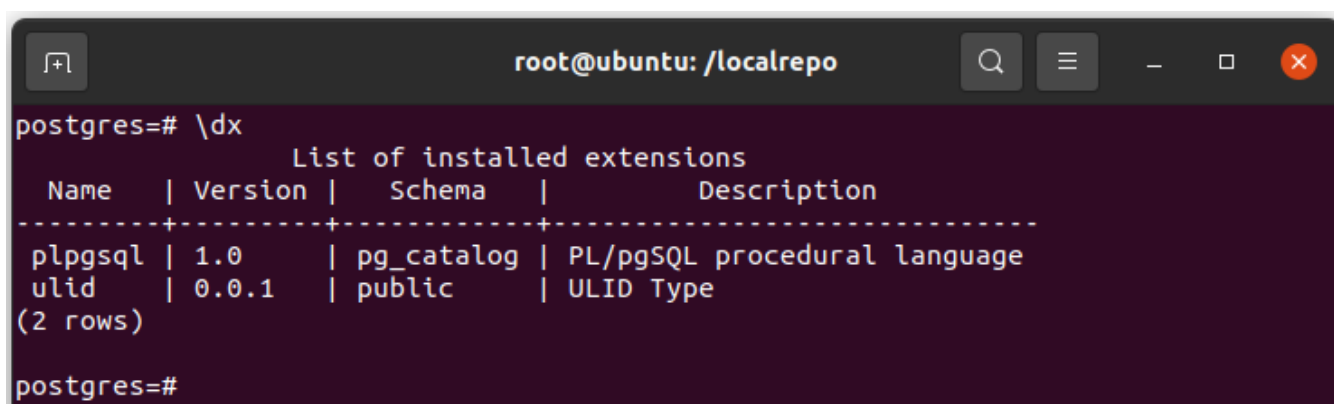
```
root@ubuntu: /localrepo

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».



```
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

3.1. Компонент pg_ulid

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.

Например

```
postgres=# SELECT gen_ulid();
          gen_ulid
-----
 01HQ5JGD1CQY4HWSR5753H61B7
(1 row)

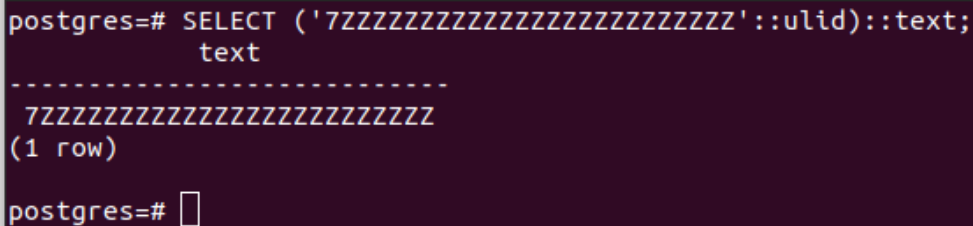
postgres=#
```

3.1.2. Конструкции приведения типа ulid к тексту и наоборот

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

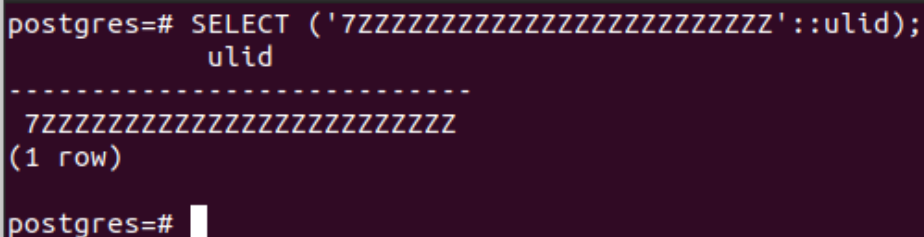
Преобразовать значение в текст:

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



Преобразовать значение в тип данных «ulid»:

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



3.1.3. Конструкции приведения типа ulid к штампу времени

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

Выполнить преобразование в timestamp:

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

```
root@ubuntu: /home/admin1
```

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

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

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

```
postgres=# 
```

Выполнить преобразование в ulid:

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

postgres=#
```

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

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

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

Например

Создать таблицу:

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

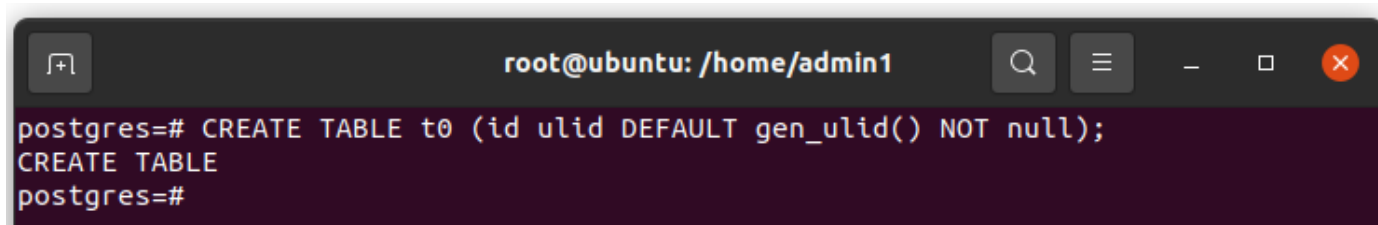


Рисунок 3.6 – Создание таблицы с типом данных «ulid»

Вставить произвольный ulid:

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

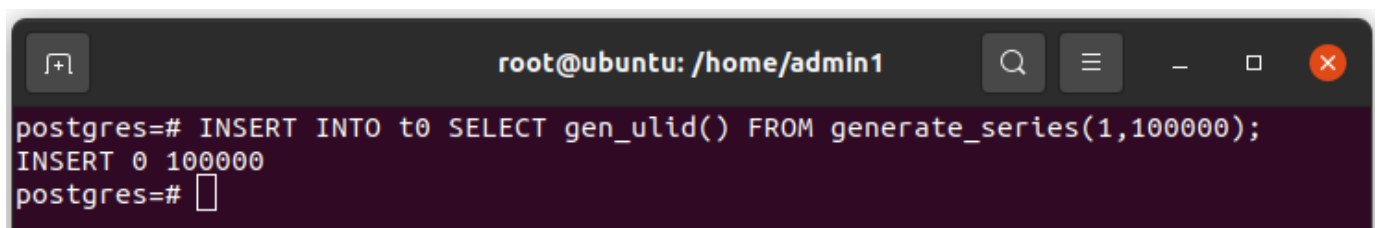


Рисунок 3.7 – Вставка произвольных значений

Указать первичный ключ:

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

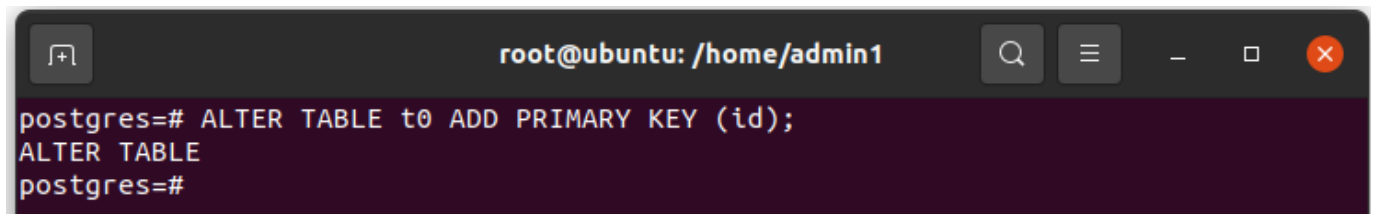
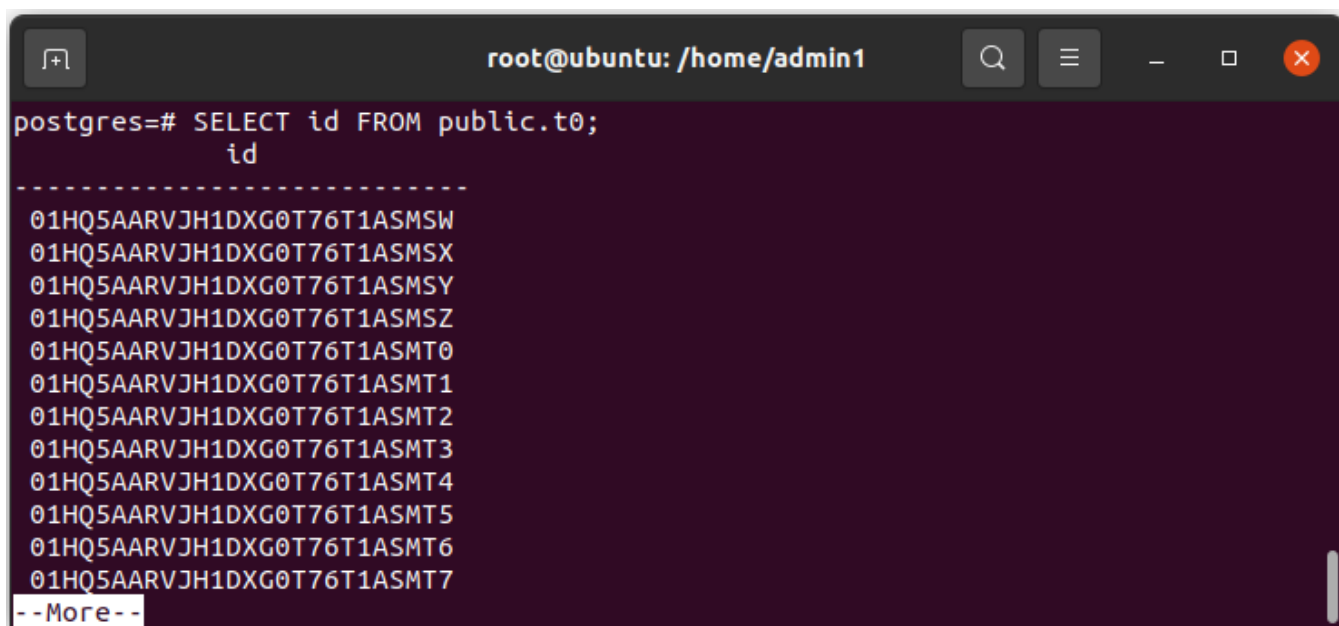


Рисунок 3.8 – Установка первичного ключа

Вывести содержание таблицы:

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



The screenshot shows a terminal window with the title bar 'root@ubuntu: /home/admin1'. The terminal content shows a PostgreSQL prompt 'postgres=#' followed by the command 'SELECT id FROM public.t0;'. The output is a list of 15 ulid values, each on a new line, starting with '01HQ5AARVJH1DXG0T76T1ASMSW' and ending with '01HQ5AARVJH1DXG0T76T1ASMT7'. Below the last line, 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

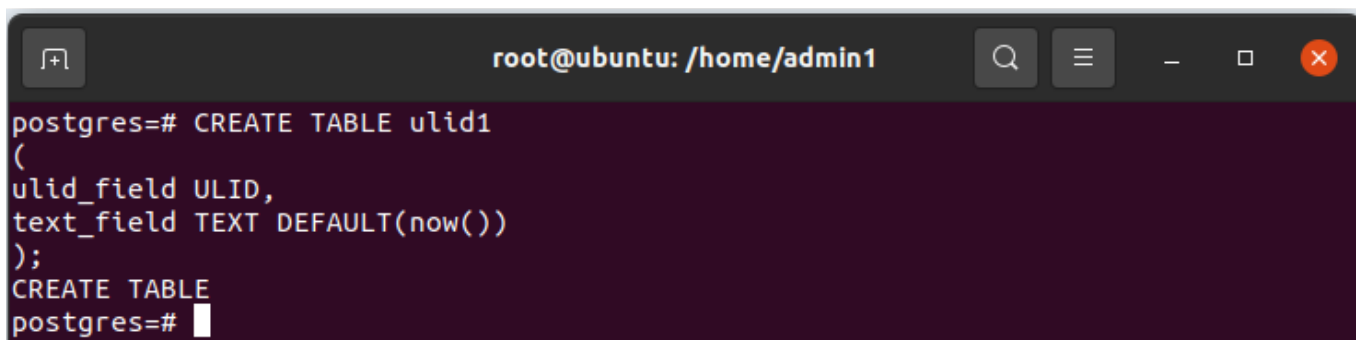
Данные выражения возвращают тип boolean и могут применяться в SQL запросах в контексте выражений:

- = 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 bar 'root@ubuntu: /home/admin1'. The command prompt is 'postgres=#'. The user has entered the command to create a table 'ulid1' with two columns: 'ulid_field' of type 'ULID' and 'text_field' of type 'TEXT' with a default value of 'now()'. The command is executed successfully, and the prompt returns to 'postgres=#'.

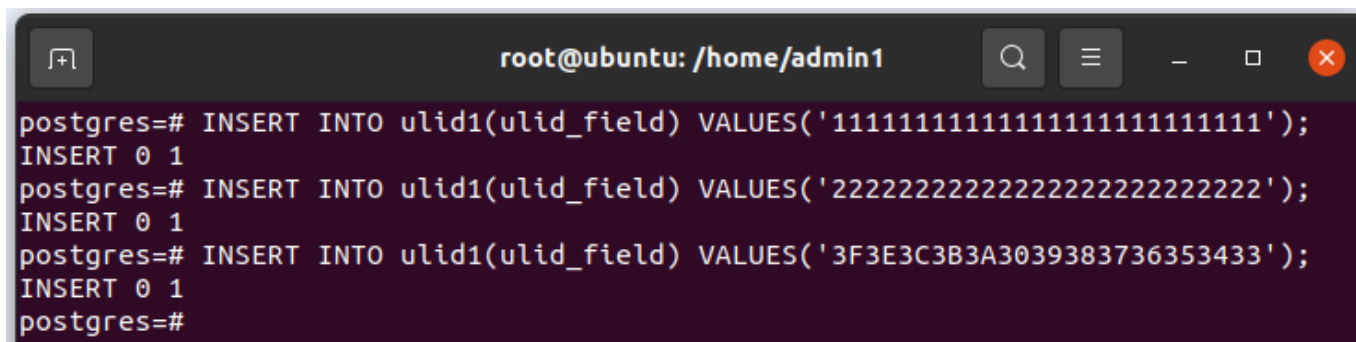
Рисунок 3.10 – Создание таблицы

Вставить данные:

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

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

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



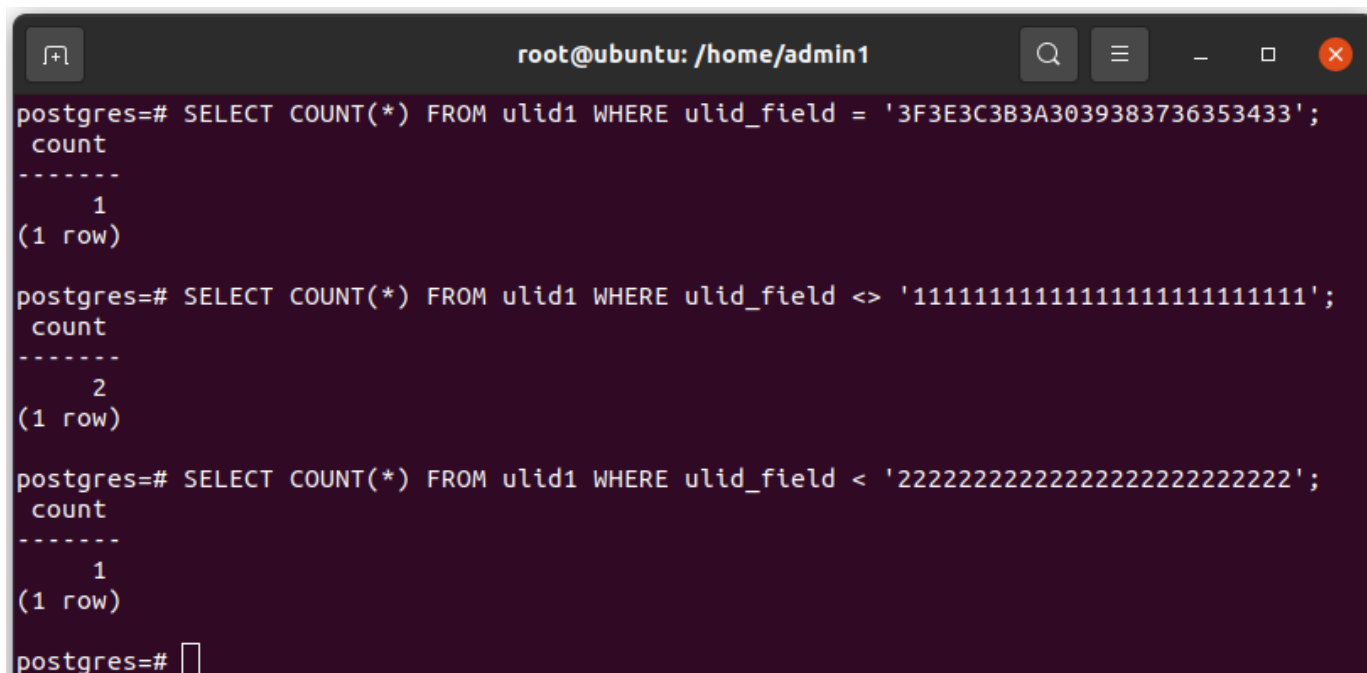
The screenshot shows a terminal window with the title bar 'root@ubuntu: /home/admin1'. The command prompt is 'postgres=#'. The user has entered three commands to insert data into the 'ulid1' table. Each command is followed by the output 'INSERT 0 1', indicating that one row was inserted successfully. The commands are: 1. 'INSERT INTO ulid1(ulid_field) VALUES('11111111111111111111111111111111');', 2. 'INSERT INTO ulid1(ulid_field) VALUES('22222222222222222222222222222222');', and 3. 'INSERT INTO ulid1(ulid_field) VALUES('3F3E3C3B3A3039383736353433');'. The prompt returns to 'postgres=#' after each command.

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 checks for a specific ulid value and returns 1 row. The second query checks for values not equal to a specific ulid and returns 1 row. The third query checks for values less than a specific ulid and returns 1 row.

```
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:

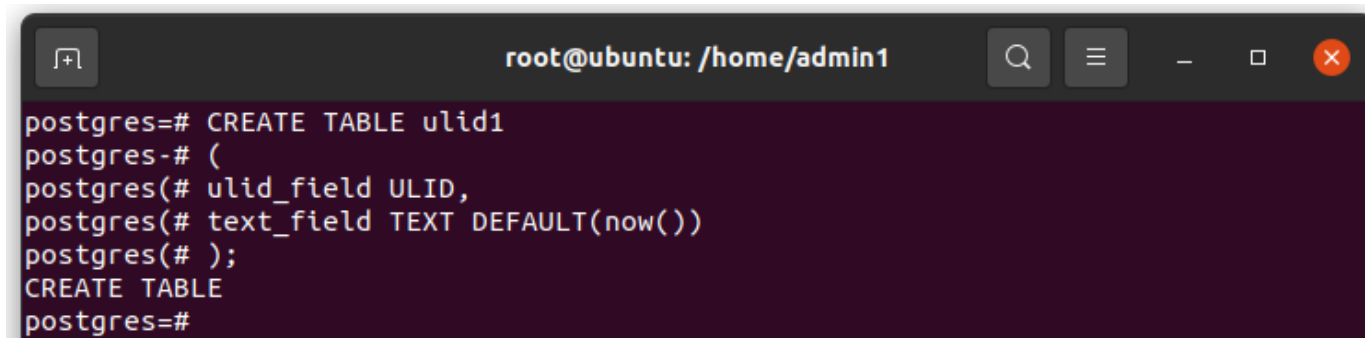
```
CREATE INDEX my_idx on my_table [ USING btree | hash ] (id);
```

Например

Создать таблицу:

```
# CREATE TABLE ulid1  
(
```

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



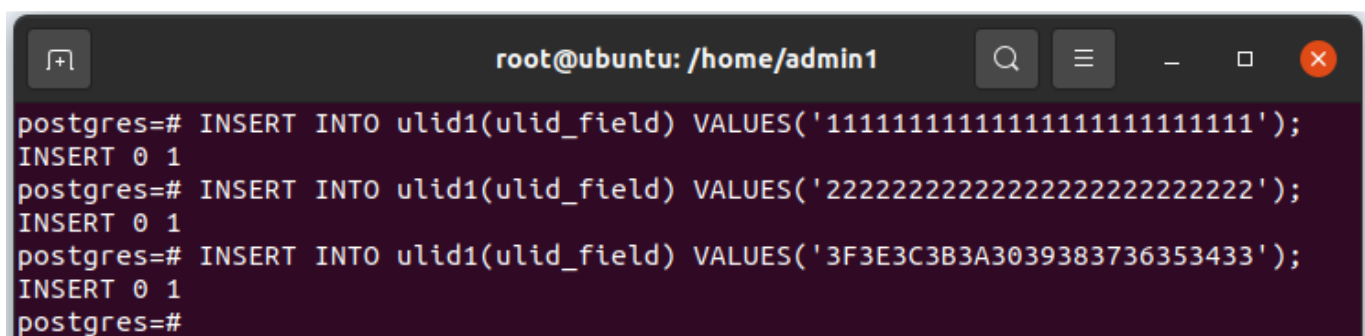
A terminal window titled 'root@ubuntu: /home/admin1' showing the execution of SQL commands to create a table. The commands are: 'CREATE TABLE ulid1', '(', 'ulid_field ULID,', 'text_field TEXT DEFAULT(now())', and ');'. The prompt 'postgres=#' is visible at the start of each line.

```
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. ПРИМЕНЕНИЕ ИДЕНТИФИКАТОРА UUIDV7

В СУБД «Jatoba» 18 встроена поддержка идентификатора UUID версии 7 – новой версии стандарта UUID (Universally Unique Identifier).

Идентификатор UUID версии 7 применяется в базах данных, распределённых системах и API.

4.1. Миграция с ULID к UUIDv7

В СУБД «Jatoba» применяются механизмы миграции с идентификаторов ULID, поддерживаемых компонентом pg_ulid, на идентификаторы UUID версии 7. Далее приводятся примеры процедур по миграции:

- Миграция от ULID к UUIDv7, когда колонка с типом данных ULID используется в качестве первичного ключа;
- Миграция от ULID к UUIDv7, когда колонка с типом данных ULID используется в качестве внешнего ключа.

4.2. ULID используется в качестве первичного ключа

4.2.1. Создание тестовой таблицы

Допустим, имеется таблица t_ulid с первичным ключом по столбцу id с типом данных ULID. Создадим таблицу t_ulid при помощи следующей команды:

```
CREATE TABLE t_ulid(  
    id          ULID PRIMARY KEY DEFAULT gen_ulid()  
, load_id    INTEGER  
, order_id   INTEGER);
```

Заполнение таблицы t_ulid случайными тестовыми данными производится при помощи команды:

```
INSERT INTO t_ulid(order_id, load_id) SELECT n, 1 FROM  
generate_series(1, 10000) AS gs(n);
```

Проверить содержимое таблицы t_ulid при помощи команды:

```
SELECT id, to_char(id::timestamp, 'dd-mm-yyyy hh24:mi:ss,ff3')
AS ts FROM t_ulid ORDER BY id FETCH FIRST 10 ROWS ONLY;
```

```
-----+-----
01KA959YWGZTV653GC7M92F2SR | 17-11-2025 15:00:34,320
01KA959YWGZTV653GC7M92F2SS | 17-11-2025 15:00:34,320
01KA959YWGZTV653GC7M92F2ST | 17-11-2025 15:00:34,320
01KA959YWGZTV653GC7M92F2SV | 17-11-2025 15:00:34,320
01KA959YWGZTV653GC7M92F2SW | 17-11-2025 15:00:34,320
01KA959YWGZTV653GC7M92F2SX | 17-11-2025 15:00:34,320
01KA959YWGZTV653GC7M92F2SY | 17-11-2025 15:00:34,320
01KA959YWGZTV653GC7M92F2SZ | 17-11-2025 15:00:34,320
01KA959YWGZTV653GC7M92F2T0 | 17-11-2025 15:00:34,320
```

Создание процедуры для формирования UUIDv7 на основе существующих значений ULID выполняется при помощи команды:

```
CREATE OR REPLACE PROCEDURE ulid2uuid_1()
LANGUAGE 'plpgsql' AS $BODY$
DECLARE
    vepoch_old    BIGINT := null;
    vepoch_new    BIGINT;
    vnano         INTEGER;
    vuuid_current TEXT;
    vuuid_new     TEXT;
    vuuid         UUID;
    c1 cursor for select id,id_uuid from t_ulid order by id for
update;
BEGIN
    for c1r in c1 loop
        vepoch_new := floor(extract(epoch from
c1r.id::timestamp)*1000)::bigint;
        if vepoch_old is null or vepoch_new<>vepoch_old then
            vepoch_old := vepoch_new; vnano := 1;
```

```
else
    vnanos := vnanos+1;
end if;

vuuid_current := uuidv7()::text; vuuid_new :=
lpad(to_hex(vepoch_new),12,'0');

vuuid_new :=
vuuid_new||substr(vuuid_current,15,1)||lpad(to_hex(vnanos),3,'0'
)||substr(vuuid_current,20,4)
||substr(vuuid_current,25,12);

vuuid := vuuid_new::uuid;

UPDATE t_ulid SET id_uuid=vuuid WHERE CURRENT OF c1;

end loop;

COMMIT;

END;

$BODY$;
```

Добавление в таблицу t_ulid нового столбца id_uuid с новым типом данных UUID при помощи команды:

```
ALTER TABLE t_ulid ADD id_uuid UUID;
```

Заполнение столбца id_uuid в таблице t_ulid значениями с использованием процедуры ulid2uuid_1 при помощи команды:

```
CALL ulid2uuid_1();
```

Пример выборки значений из таблицы t_ulid

```
SELECT id, to_char(id::timestamp,'dd-mm-yyyy hh24:mi:ss,ff3')
AS ts,id_uuid, to_char(uuid_extract_timestamp(id_uuid) AT TIME
ZONE 'UTC','dd-mm-yyyy hh24:mi:ss,ff3') AS ts_id_uuid FROM
t_ulid ORDER BY id LIMIT 10;
```

В результате будет представлено

```
01KA959YWGZTV653GC7M92F2SQ | 17-11-2025 15:00:34,320 |
019a9254-fb90-7001-ba95-779652ad9221 | 17-11-2025 15:00:34,320
```

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

```
01KA959YWGZTV653GC7M92F2SR | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7002-b777-ba487bbb4db8 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SS | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7003-a036-538e36dc2cb6 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2ST | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7004-9fa8-4ed5ac39ee44 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SV | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7005-9b20-1d61a3af869c | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SW | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7006-bc02-6afcfbbdb452 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SX | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7007-b679-46edfcfbcc79 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SY | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7008-ad68-f8c22a5e39e1 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SZ | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7009-b824-8f2c74911c8d | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2T0 | 17-11-2025 15:00:34,320 |  
019a9254-fb90-700a-a304-77b1b1480705 | 17-11-2025 15:00:34,320
```

Пример выборки значений из таблицы t_ulid

```
SELECT id, to_char(id::timestamp, 'dd-mm-yyyy hh24:mi:ss,ff3')  
AS ts, id_ulid, to_char(uuid_extract_timestamp(id_ulid) AT TIME  
ZONE 'UTC', 'dd-mm-yyyy hh24:mi:ss,ff3') AS ts_id_ulid FROM  
t_ulid ORDER BY id_ulid LIMIT 10;
```

В результате выполнения предыдущего запроса

```
01KA959YWGZTV653GC7M92F2SQ | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7001-ba95-779652ad9221 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SR | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7002-b777-ba487bbb4db8 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SS | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7003-a036-538e36dc2cb6 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2ST | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7004-9fa8-4ed5ac39ee44 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SV | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7005-9b20-1d61a3af869c | 17-11-2025 15:00:34,320
```

```
01KA959YWGZTV653GC7M92F2SW | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7006-bc02-6afcfbbdb452 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SX | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7007-b679-46edfcfbcc79 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SY | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7008-ad68-f8c22a5e39e1 | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2SZ | 17-11-2025 15:00:34,320 |  
019a9254-fb90-7009-b824-8f2c74911c8d | 17-11-2025 15:00:34,320  
  
01KA959YWGZTV653GC7M92F2T0 | 17-11-2025 15:00:34,320 |  
019a9254-fb90-700a-a304-77b1b1480705 | 17-11-2025 15:00:34,320
```

4.2.2. Изменение ограничений первичного ключа в таблице

Получение ограничений таблицы `t_ulid`:

```
SELECT conname AS constraint_name,conrelid::regclass AS  
table_name,contype AS constraint_type,pg_get_constraintdef(oid)  
AS constraint_definition FROM pg_constraint WHERE conrelid =  
't_ulid'::regclass;
```

В ответ будет выведено следующее:

```
t_ulid_pkey | t_ulid | p | PRIMARY KEY (id)
```

Удаление ограничения первичного ключа в таблице `t_ulid`:

```
ALTER TABLE t_ulid DROP CONSTRAINT t_ulid_pkey CASCADE;
```

Добавление нового ограничения первичного ключа в таблицу `t_ulid`:

```
ALTER TABLE t_ulid ADD PRIMARY KEY (id_uuid);
```

Удаление столбца `id` из таблицы `t_ulid`:

```
ALTER TABLE t_ulid DROP id CASCADE;
```

4.3. ULID используется в качестве внешнего ключа

4.3.1. Создание тестовой таблицы

Допустим, имеется таблица `t_ulid` с первичным ключом по столбцу `id` с типом данных `ULID`. Создадим таблицу `t_ulid` при помощи следующей команды:

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

```
CREATE TABLE t_ulid(  
    id          ULID PRIMARY KEY DEFAULT gen_ulid()  
    ,load_id    INTEGER  
    ,order_id   INTEGER);
```

Допустим, имеется таблица t_ulid_detail с внешним ключом по столбцу id_fk с типом данных ULID, который указывает на первичный ключ в таблице t_ulid.

После добавления и заполнения UUIDv7 в таблице t_ulid (см. п. 4.2.1), необходимо удалить старые и создать новые ограничения.

Для этого следует добавить в таблицу t_ulid_detail нового столбца id_uuid_fk с типом данных UUID:

```
ALTER TABLE t_ulid_detail ADD id_uuid_fk UUID;
```

Заполнение данными столбца id_uuid_fk в таблице t_ulid_detail:

```
UPDATE t_ulid_detail t1 SET id_uuid_fk = (SELECT t2.id_uuid  
FROM t_ulid t2 WHERE t2.id=t1.id_fk);
```

4.3.2. Изменение ограничений внешнего ключа

Получение ограничений таблицы t_ulid

```
SELECT conname AS constraint_name,conrelid::regclass AS  
table_name,contype AS constraint_type,pg_get_constraintdef(oid)  
AS constraint_definition FROM pg_constraint WHERE conrelid =  
't_ulid'::regclass;
```

В ответ будет выведено следующее:

```
t_ulid_pkey | t_ulid | p | PRIMARY KEY (id)
```

Удаление ограничения первичного ключа в таблице t_ulid:

```
ALTER TABLE t_ulid DROP CONSTRAINT t_ulid_pkey CASCADE;
```



Удаление распространяется на объект ограничение t_ulid_detail_id_fk_fkey в отношении таблицы t_ulid_detail.

Добавление нового ограничения первичного ключа в таблицу t_ulid:

```
ALTER TABLE t_ulid ADD PRIMARY KEY (id_uuid);
```

Добавление нового ограничения внешнего ключа в таблицу t_ulid_detail:

```
ALTER TABLE t_ulid_detail ADD FOREIGN KEY (id_uuid_fk)  
REFERENCES t_ulid(id_uuid);
```

5. 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: _____
--------------------	--------------------------	--------------------------