

ЗАЩИЩЕННАЯ СИСТЕМА УПРАВЛЕНИЯ
БАЗАМИ ДАННЫХ «ЈАТОВА»

Руководство по настройке. Часть 32.
Контроль выполненных планов запросов.
Компонент «pg_store_plans»

643.72410666.00067-07 98 01-32

Листов 36

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

АННОТАЦИЯ

В документе приведены сведения, необходимые для установки и эксплуатации компонента «pg_store_plans» (далее по тексту – «компонент»), предназначенного для контроля выполнения запросов.

Настоящее руководство предназначено для администраторов СУБД.

Для СУБД «Jatoba» версий ядра 5 и 6 используется версия компонента – 1.8

Для СУБД «Jatoba» версий ядра 18 используется версия компонента – 1.9

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. Параметры конфигурации.....	6
2.1.1. Параметр «pg_store_plans.max»	6
2.1.2. Параметр «pg_store_plans.track»	8
2.1.3. Параметр «pg_store_plans.plan_format»	9
2.1.4. Параметр pg_store_plans.max_plan_length	11
2.1.5. Параметр «pg_store_plans.plan_storage».....	12
2.1.6. Параметр «pg_store_plans.min_duration»	14
2.1.7. Параметр «pg_store_plans.log_analyze».....	15
2.1.8. Параметр «pg_store_plans.log_buffers»	17
2.1.9. Параметр «pg_store_plans.log_timing»	18
2.1.10. Параметр «pg_store_plans.log_triggers».....	20
2.1.11. Параметр «pg_store_plans.save»	21
3. Функциональные возможности компонента.....	24
3.1. Представление «pg_store_plans».....	24
3.2. Представление «pg_store_plans_info».....	26
3.3. Функции компонента	26
3.3.1. Функция «pg_store_plans_reset»	26
3.3.2. Функция «pg_store_plans»	27
3.3.3. Функция «pg_store_plans_info»	28
3.3.4. Функция «pg_store_hash_query».....	28
3.3.5. Функция «pg_store_plans_textplan».....	29
3.3.6. Функция «pg_store_plans_xmlplan».....	29
3.3.7. Функция «pg_store_plans_yamlplan».....	30
4. Пример вывода.....	31
4.1. Создание тестовой таблицы.....	31
4.2. Вывод оценки качества плана.....	32
5. Обновление компонента	34
5.1. Обновление компонента pg_store_plans в ОС GNU/Linux.....	34
Перечень сокращений.....	35

1. НАЗНАЧЕНИЕ КОМПОНЕНТА

Компонент «pg_store_plans» предназначен для контроля выполнения планов запросов статистическими методами всех операторов SQL, выполняемых сервером СУБД.

1.1. Условия применения

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



В текущей реализации компонента не поддерживается управление через компонент 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).

Для установки компонента в разделе «Shared Library Preloading» конфигурационного файла «postgresql.conf» прописать следующую строку:

```
shared_preload_libraries = 'pg_store_plans'
```

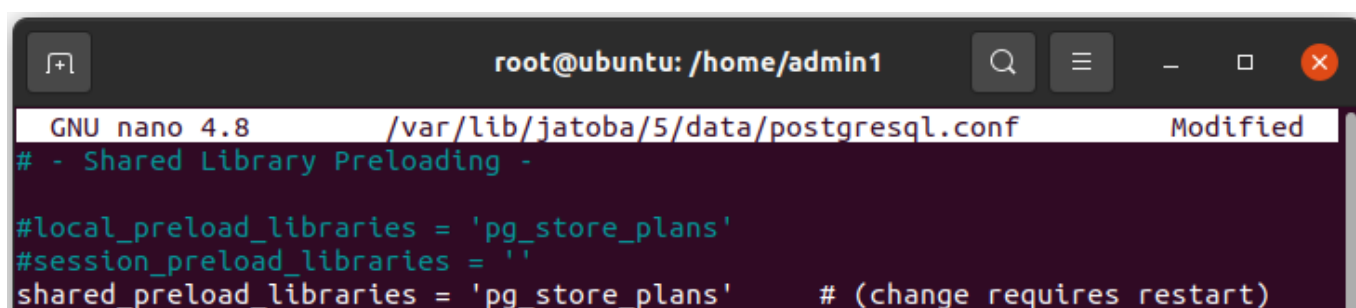


Рисунок 2.1 – Параметры конфигурационного файла «postgresql.conf»

Сохранив изменения и перезагрузив СУБД станет доступной установка расширения выполнением SQL-команды:

```
CREATE EXTENSION pg_store_plans;
```

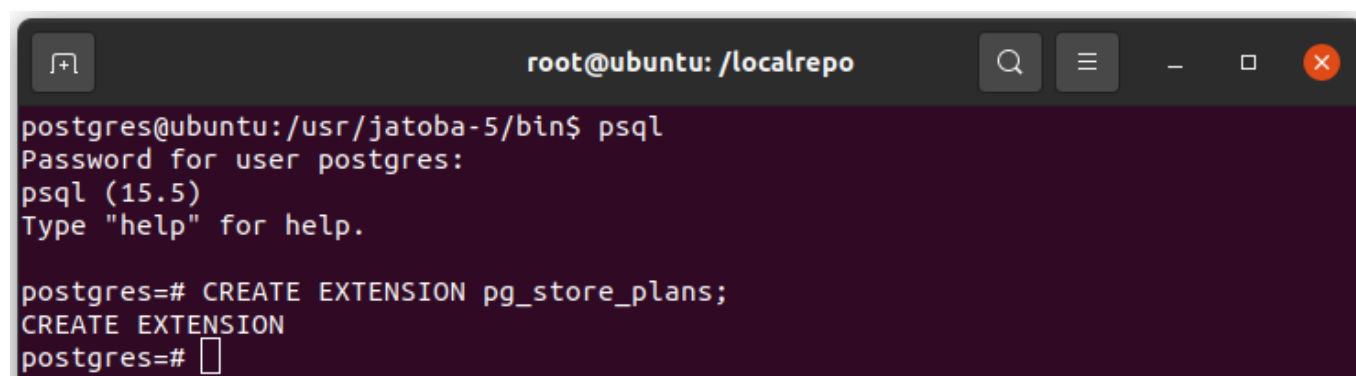


Рисунок 2.2 – Установка расширения «pg_store_plans»

Корректность установки расширения выполняется SQL-командой

```
\dx
```

```

postgres=# \dx
              List of installed extensions
  Name          | Version | Schema  | Description
-----+-----+-----+-----
 pg_store_plans | 1.7     | public  | track plan statistics of all SQL statements executed
 plpgsql        | 1.0     | pg_catalog | PL/pgSQL procedural language
(2 rows)

postgres=#

```

Рисунок 2.3 – Проверка установленных расширений

2.1. Параметры конфигурации

Параметры конфигурации компонента имеют два способа установки:

- через конфигурационный файл «postgresql.conf», как представлено на рисунке 2.4.

```

GNU nano 4.8 /var/lib/jatoba/5/data/postgresql.conf Modified
# - Shared Library Preloading -

#local_preload_libraries = 'pg_store_plans'
#session_preload_libraries = ''
shared_preload_libraries = 'pg_store_plans'      # (change requires restart)
pg_store_plans.max = 10000
pg_store_plans.track = all

```

Рисунок 2.4 – Параметры конфигурации в конфигурационном файле «postgresql.conf»

- через SQL-команды:

```

ALTER SYSTEM SET [имя параметра конфигурации];
SELECT pg_reload_conf();

```

Параметры компонента устанавливаются от имени и с правами привилегированного пользователя postgres или с пользователя с атрибутом «SUPERUSER».

2.1.1. Параметр «pg_store_plans.max»

Параметр «pg_store_plans.max» имеет синтаксис:

```
pg_store_plans.max (целое число)
```

Значение по умолчанию — 1000

Параметр «pg_store_plans.max» — это максимальное количество планов, отслеживаемых компонентом, а также параметр ограничивает максимальное количество строк в представлении «pg_store_plans», описание которого приводится в п. 3.1.

Параметр задается только при запуске сервера.

Например

Посмотрим установленное значение «pg_store_plans.max» через SQL-команду

```
SHOW pg_store_plans.max;
```

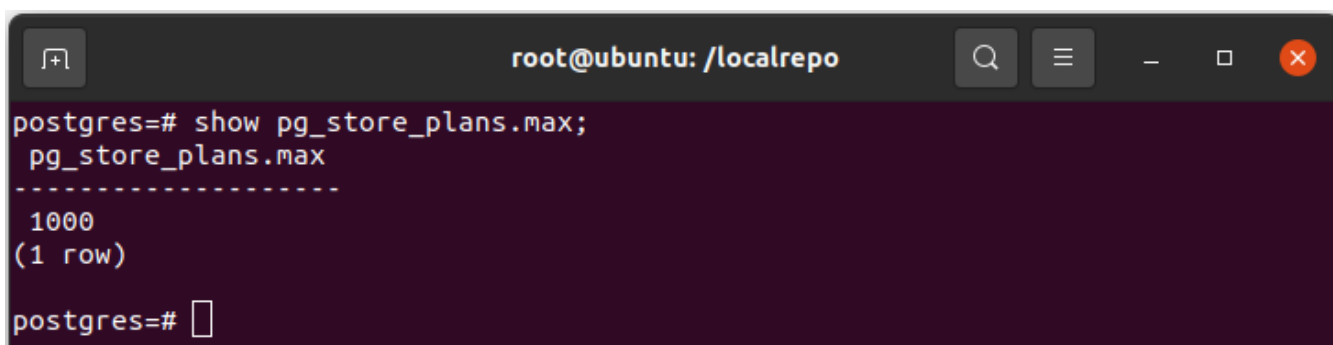


Рисунок 2.5 – Просмотр параметра по умолчанию

Значение параметра соответствует значению по умолчанию.

Установим новое значение равным 10000, перезагрузим конфигурацию СУБД и проверим применение установленного параметра SQL-командами:

```
ALTER SYSTEM SET pg_store_plans.max = 10000;  
SELECT pg_reload_conf();  
SHOW pg_store_plans.max;
```

```
root@ubuntu: /localrepo
postgres=# alter system set pg_store_plans.max = 10000;
ALTER SYSTEM
postgres=# select pg_reload_conf();
pg_reload_conf
-----
t
(1 row)

postgres=# show pg_store_plans.max;
pg_store_plans.max
-----
10000
(1 row)

postgres=#
```

Рисунок 2.6 – Установка параметра «pg_store_plans.max»

2.1.2. Параметр «pg_store_plans.track»

Параметр «pg_store_plans.track» имеет синтаксис:

```
pg_store_plans.track (перечисление)
```

Значение по умолчанию — «top».

Параметр «pg_store_plans.track» указывает, какие операторы учитываются компонентом.

Устанавливаются следующие значения:

- «top» - отслеживание операторов верхнего уровня (выданные непосредственно клиентами);
- «all» - отслеживание вложенных операторов (например, операторы, вызываемые внутри функций);



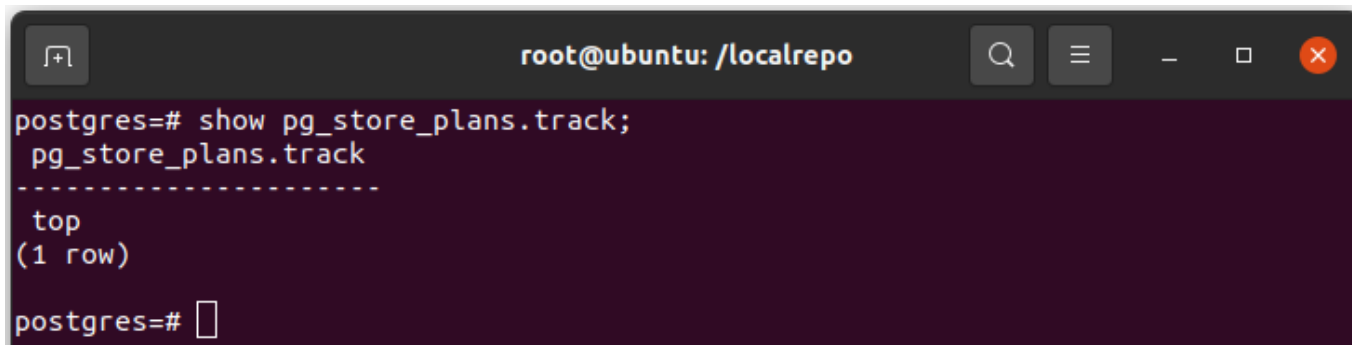
При использовании значения «all» команды «CREATE EXTENSION» и «ALTER EXTENSION» игнорируются

- «none» - отключение сбора статистики операторов.

Например

Посмотрим установленное значение «pg_store_plans.max» через SQL-команду:


```
SHOW pg_store_plans.track;
```



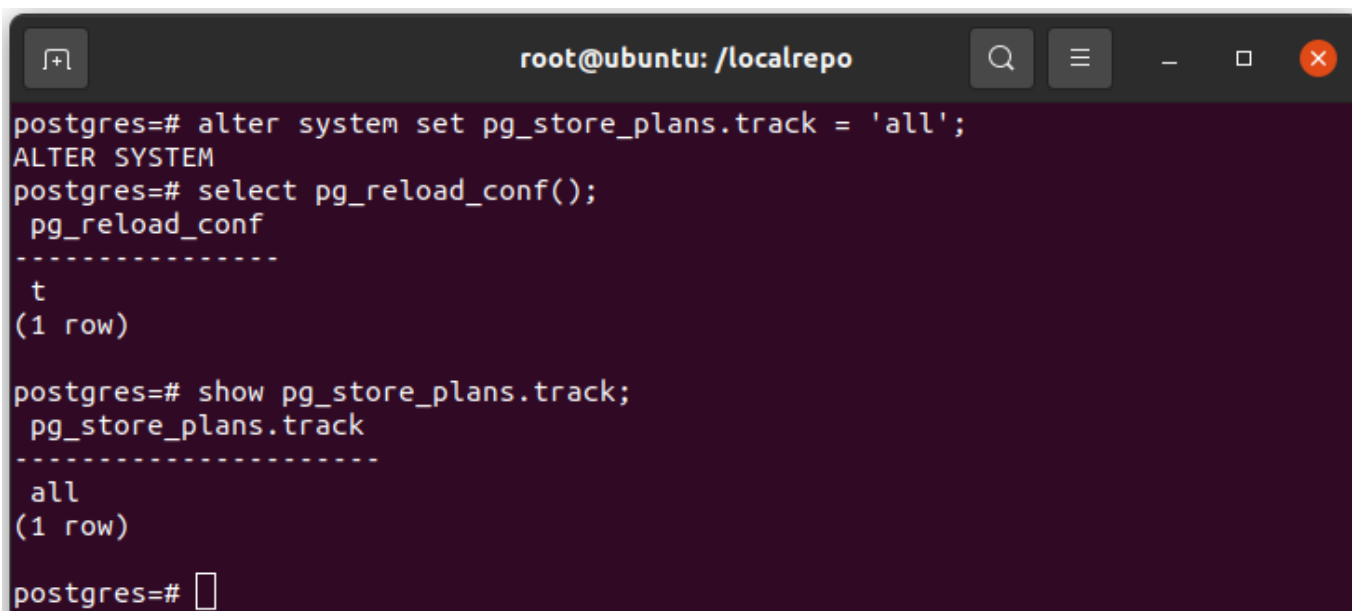
```
root@ubuntu: /localrepo
postgres=# show pg_store_plans.track;
pg_store_plans.track
-----
top
(1 row)
postgres=#
```

Рисунок 2.7 - Просмотр установленного значения параметра «pg_store_plans.max»

Значение параметра соответствует значению по умолчанию.

Установим новое значение равным «all», перезагрузим конфигурацию СУБД и проверим применение установленного параметра SQL-командами:

```
ALTER SYSTEM SET pg_store_plans.track = 'all';
SELECT pg_reload_conf();
SHOW pg_store_plans.track;
```



```
root@ubuntu: /localrepo
postgres=# alter system set pg_store_plans.track = 'all';
ALTER SYSTEM
postgres=# select pg_reload_conf();
pg_reload_conf
-----
t
(1 row)

postgres=# show pg_store_plans.track;
pg_store_plans.track
-----
all
(1 row)
postgres=#
```

Рисунок 2.8 – Установка нового значения параметра «pg_store_plans.max»

2.1.3. Параметр «pg_store_plans.plan_format»

Параметр «pg_store_plans.plan_format» имеет синтаксис:

```
pg_store_plans.plan_format (перечисление)
```

Значение по умолчанию — text.

Параметр «pg_store_plans.plan_format» устанавливает форматы планов.

Устанавливаются следующие значения:

- «text»;
- «json»;
- «xml»;
- «yaml»;
- «raw».



Значение «raw» используется, когда требуется получить внутреннее представление, которое можно передать функция компонента

Например

Посмотрим установленное значение параметра «pg_store_plans.max» через SQL-команду:

```
SHOW pg_store_plans.plan_format;
```

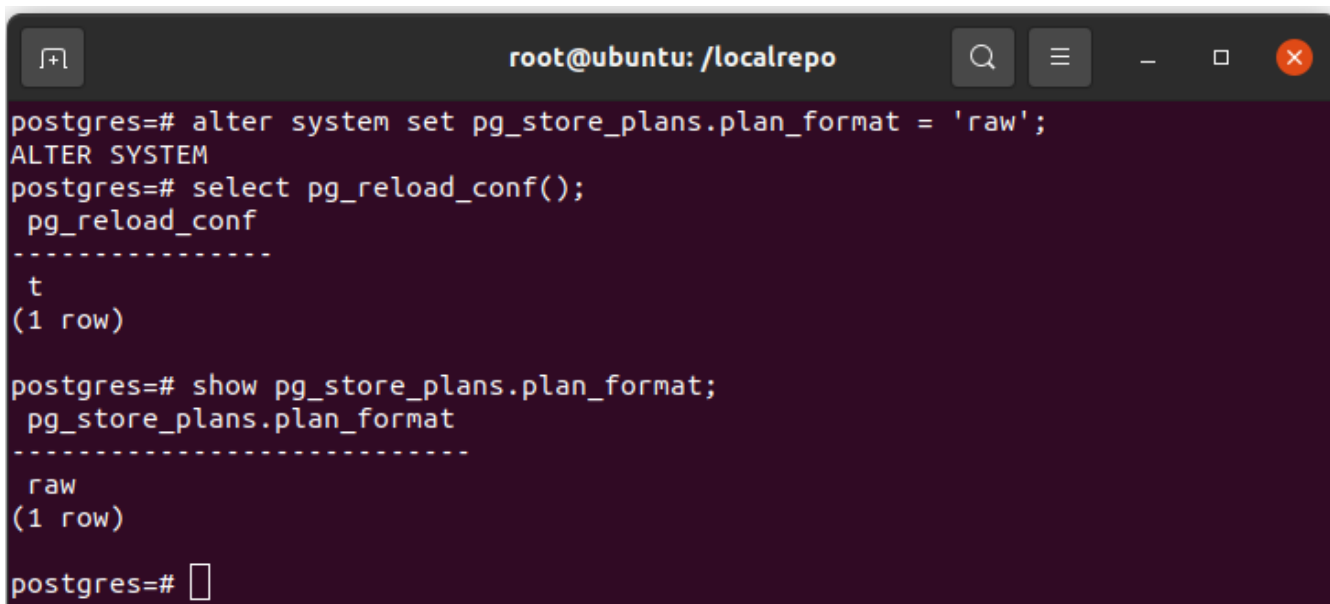
```
root@ubuntu: /localrepo
postgres=# show pg_store_plans.plan_format;
pg_store_plans.plan_format
-----
text
(1 row)
postgres=#
```

Рисунок 2.9 – Просмотр значения параметра «pg_store_plans.max»

Значение параметра соответствует значению по умолчанию.

Установим новое значение равным «raw», перезагрузим конфигурацию СУБД и проверим применение установленного параметра SQL-командами:

```
ALTER SYSTEM SET pg_store_plans.plan_format = 'raw';  
SELECT pg_reload_conf();  
SHOW pg_store_plans.plan_format;
```



The screenshot shows a terminal window with the title 'root@ubuntu: /localrepo'. The user is connected to the PostgreSQL command prompt 'postgres=#'. The following commands and their outputs are shown:

```
postgres=# alter system set pg_store_plans.plan_format = 'raw';  
ALTER SYSTEM  
postgres=# select pg_reload_conf();  
pg_reload_conf  
-----  
t  
(1 row)  
  
postgres=# show pg_store_plans.plan_format;  
pg_store_plans.plan_format  
-----  
raw  
(1 row)  
  
postgres=#
```

Рисунок 2.10 – Установка нового значения параметра «pg_store_plans.max»

2.1.4. Параметр pg_store_plans.max_plan_length

Параметр «pg_store_plans.max_plan_length» имеет синтаксис:

```
pg_store_plans.max_plan_length (целое число)
```

Значение по умолчанию — 5000



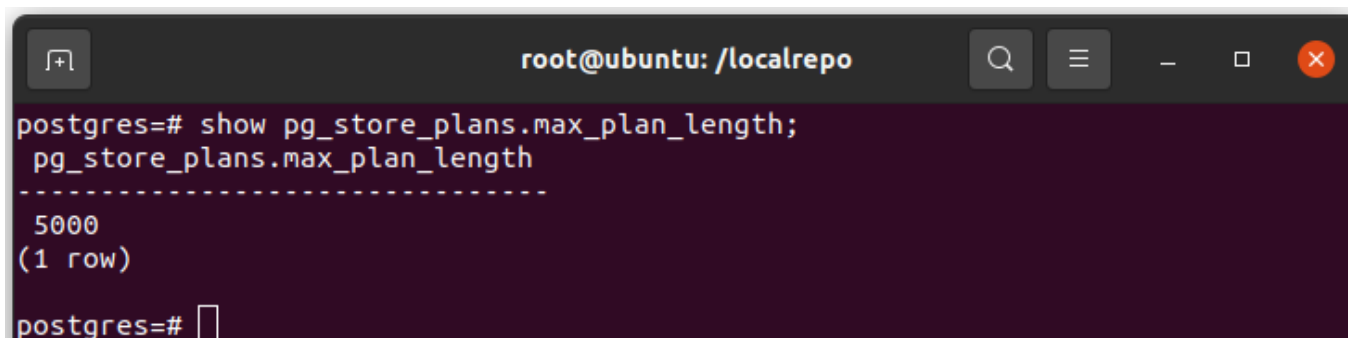
Устанавливается только после значения «pg_store_plans.plan_format» (см. п. 2.1.3)

Параметр «pg_store_plans.max_plan_length» устанавливает максимальную длину планов в необработанном (сокращенном формате JSON) для хранения в байтах. Этот параметр можно задать только при запуске сервера.

Например

Посмотрим установленное значение «pg_store_plans.max_plan_length» через SQL-команду:

```
SHOW pg_store_plans.max_plan_length;
```

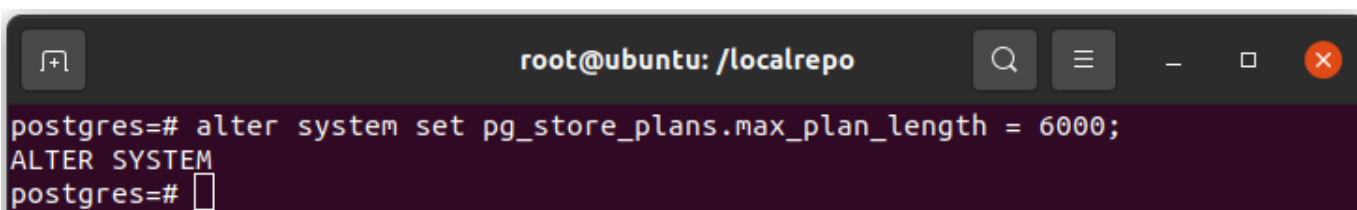


```
root@ubuntu: /localrepo
postgres=# show pg_store_plans.max_plan_length;
pg_store_plans.max_plan_length
-----
5000
(1 row)
postgres=#
```

Рисунок 2.11 – SQL-команда просмотра параметра «pg_store_plans.max_plan_length»
Значение параметра соответствует значению по умолчанию.

Установим новое значение равным 6000 SQL-командой:

```
ALTER SYSTEM SET pg_store_plans.max_plan_length = 6000;
```



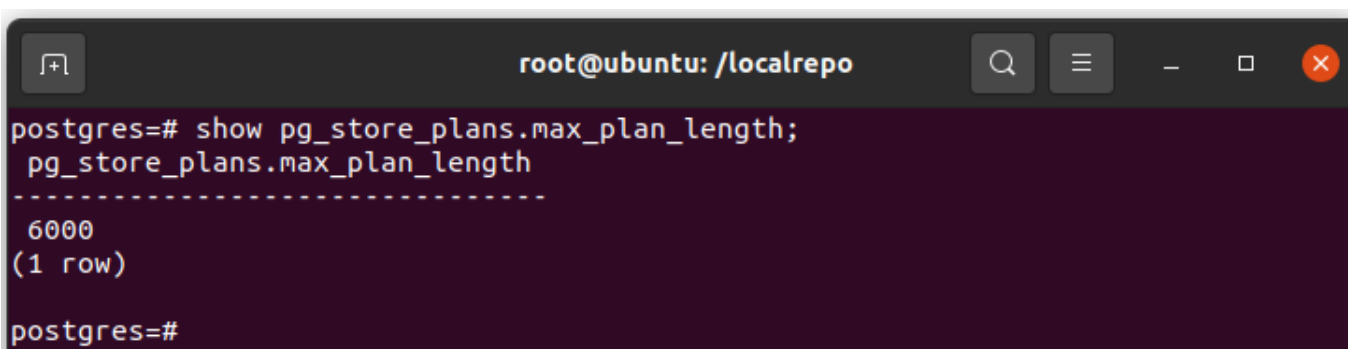
```
root@ubuntu: /localrepo
postgres=# alter system set pg_store_plans.max_plan_length = 6000;
ALTER SYSTEM
postgres=#
```

Рисунок 2.12 – SQL-команда установки нового значения параметра
«pg_store_plans.max_plan_length»

Перезагрузим СУБД.

Проверим применение установленного параметра SQL-командой:

```
SHOW pg_store_plans.max_plan_length;
```



```
root@ubuntu: /localrepo
postgres=# show pg_store_plans.max_plan_length;
pg_store_plans.max_plan_length
-----
6000
(1 row)
postgres=#
```

Рисунок 2.13 - SQL-команда просмотра нового значения параметра
«pg_store_plans.max_plan_length»

2.1.5. Параметр «pg_store_plans.plan_storage»

Параметр «pg_store_plans.plan_storage» имеет синтаксис:

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

```
pg_store_plans.plan_storage (перечисление)
```

Значение по умолчанию — «файл».

Параметр «pg_store_plans.plan_storage» устанавливает метод хранения текстов планов во время работы сервера.

Устанавливаются следующие значения:

- «file» - хранение текстов планов сохраняются во временном файле;
- «shmem» - хранение текстов планов в памяти.

Например

Посмотрим установленное значение «pg_store_plans.plan_storage» через SQL-команду:

```
SHOW pg_store_plans.plan_storage;
```

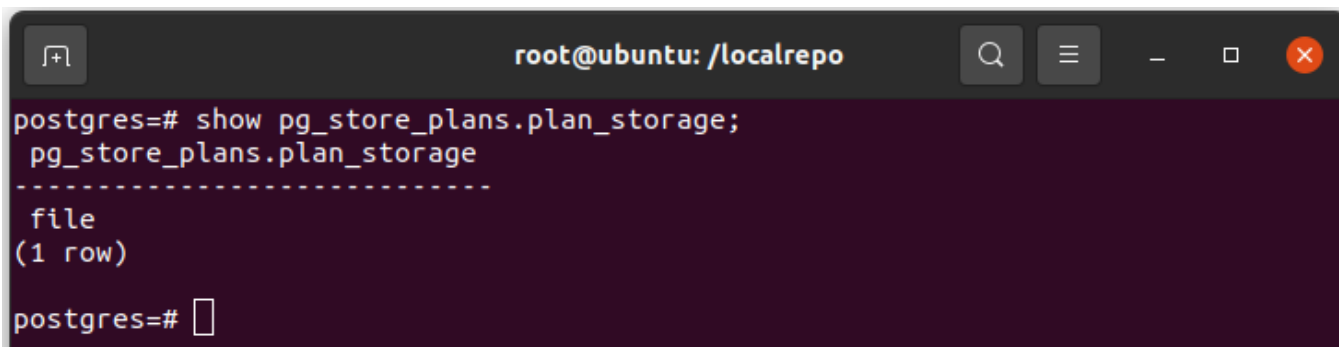
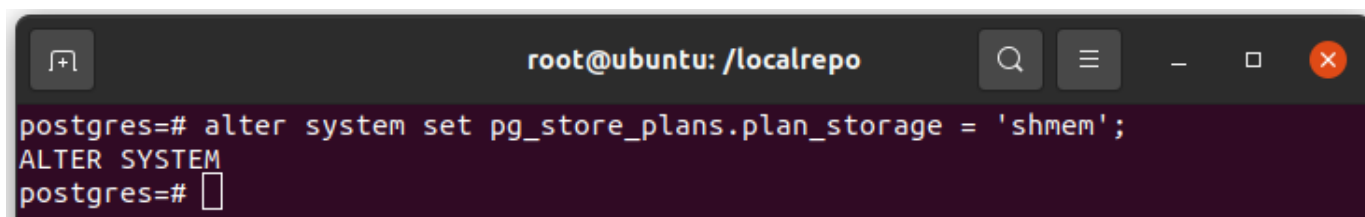


Рисунок 2.14 – SQL-команда установленного значения по умолчанию параметра «pg_store_plans.plan_storage»

Значение параметра соответствует значению по умолчанию.

Установим новое значение равным «shmem» SQL-командой:

```
ALTER SYSTEM SET pg_store_plans.plan_storage = 'shmem';
```



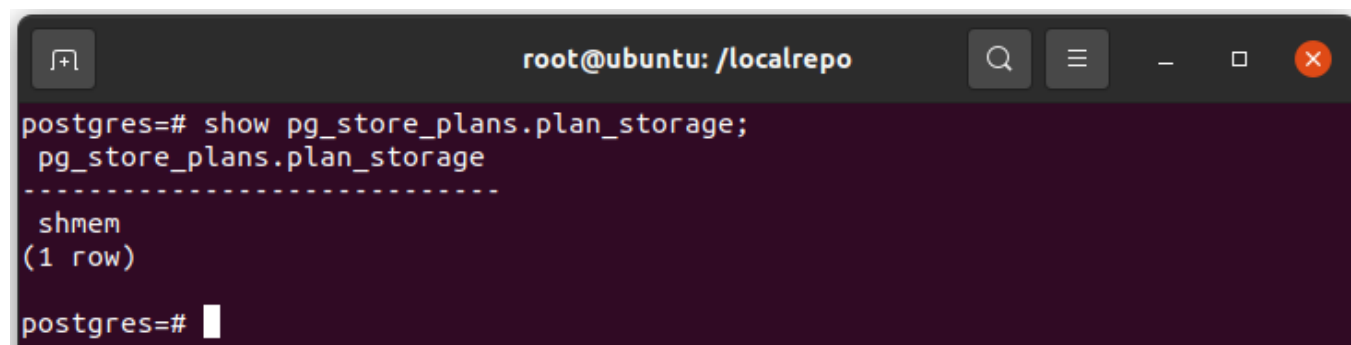
```
root@ubuntu: /localrepo
postgres=# alter system set pg_store_plans.plan_storage = 'shmem';
ALTER SYSTEM
postgres=#
```

Рисунок 2.15 - SQL-команда установки нового значения параметра «pg_store_plans.plan_storage»

Перезагрузим СУБД.

Проверим применение установленного параметра SQL-командой:

```
SHOW pg_store_plans.plan_storage;
```



```
root@ubuntu: /localrepo
postgres=# show pg_store_plans.plan_storage;
pg_store_plans.plan_storage
-----
shmem
(1 row)
postgres=#
```

Рисунок 2.16 – Просмотр установленного параметра «pg_store_plans.plan_storage»

2.1.6. Параметр «pg_store_plans.min_duration»

Параметр «pg_store_plans.min_duration» имеет синтаксис:

```
pg_store_plans.min_duration (целое число)
```

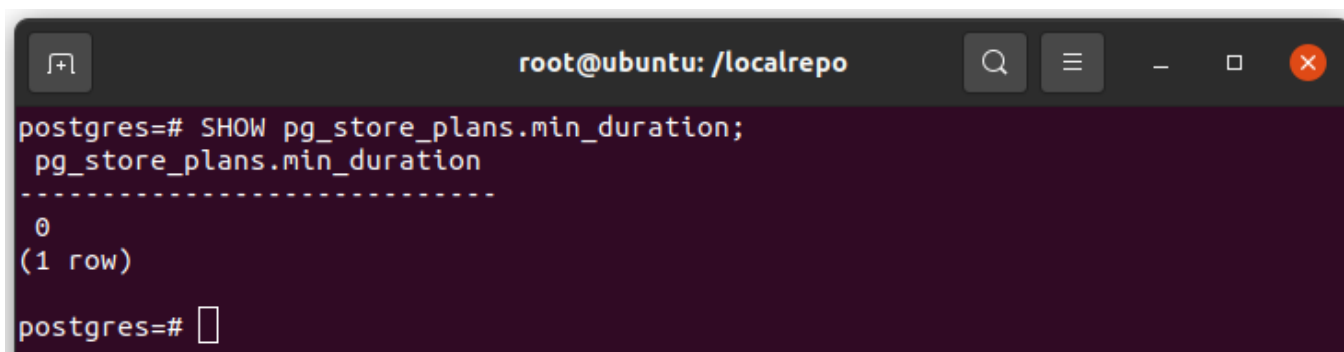
Значение по умолчанию - «0».

Параметр «pg_store_plans.min_duration» устанавливает минимальное время выполнения оператора в миллисекундах. Установленное значение по умолчанию регистрирует все планы.

Например

Просмотрим установленное значение параметра «pg_store_plans.min_duration» через SQL-команду:

```
SHOW pg_store_plans.min_duration;
```



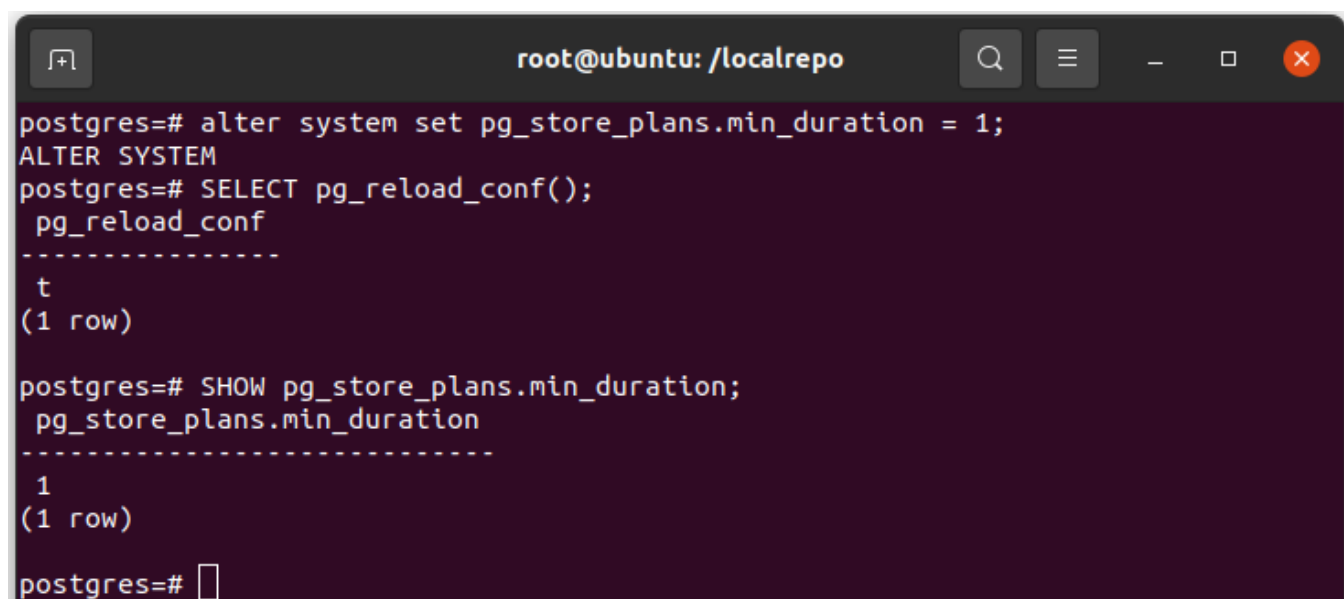
```
root@ubuntu: /localrepo
postgres=# SHOW pg_store_plans.min_duration;
pg_store_plans.min_duration
-----
0
(1 row)

postgres=#
```

Рисунок 2.17 – Просмотр текущего значения параметра «pg_store_plans.min_duration»

Установим новое значение равным «1», перезагрузим конфигурацию СУБД и проверим применение установленного параметра SQL-командами:

```
ALTER SYSTEM SET pg_store_plans.min_duration = 1;
SELECT pg_reload_conf();
SHOW pg_store_plans.min_duration;
```



```
root@ubuntu: /localrepo
postgres=# alter system set pg_store_plans.min_duration = 1;
ALTER SYSTEM
postgres=# SELECT pg_reload_conf();
pg_reload_conf
-----
t
(1 row)

postgres=# SHOW pg_store_plans.min_duration;
pg_store_plans.min_duration
-----
1
(1 row)

postgres=#
```

Рисунок 2.18 – Установка нового значения параметра «pg_store_plans.min_duration»

2.1.7. Параметр «pg_store_plans.log_analyze»

Параметр «pg_store_plans.log_analyze» имеет синтаксис:

```
pg_store_plans.log_analyze (логическое значение)
```

Устанавливаются значения:

- «on», «true», «yes», «1» – включен;

- «off», «false», «no», «0»- отключен.

По умолчанию параметр отключен.

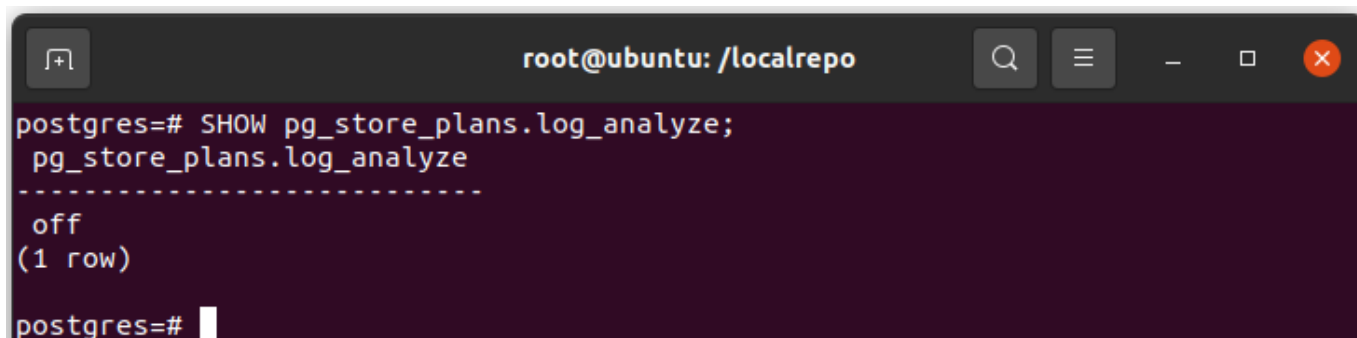
При выводе показываются значения «on»/«off».

Параметр «pg_store_plans.log_analyze» устанавливает включение в план вывода команды «EXPLAIN» с параметром «ANALYZE», а не только команду «EXPLAIN».

Например

Посмотрим установленное значение параметра «pg_store_plans.max» через SQL-команду:

```
SHOW pg_store_plans.log_analyze;
```

A screenshot of a terminal window with a dark background. The window title is 'root@ubuntu: /localrepo'. The terminal shows the command 'postgres=# SHOW pg_store_plans.log_analyze;' followed by the output 'pg_store_plans.log_analyze' and a dashed line separator. Below the separator, the value 'off' is displayed, followed by '(1 row)'. The prompt 'postgres=#' is visible at the bottom with a cursor.

```
postgres=# SHOW pg_store_plans.log_analyze;
pg_store_plans.log_analyze
-----
off
(1 row)

postgres=#
```

Рисунок 2.19 – Просмотр значения по умолчанию параметра «pg_store_plans.log_analyze»
Значение параметра соответствует значению по умолчанию.

Установим новое значение равным «on», перезагрузим конфигурацию СУБД и проверим применение установленного параметра SQL-командами:

```
ALTER SYSTEM SET pg_store_plans.log_analyze = on;
SELECT pg_reload_conf();
SHOW pg_store_plans.log_analyze;
```



```
root@ubuntu: /localrepo
postgres=# ALTER SYSTEM SET pg_store_plans.log_analyze = on;
ALTER SYSTEM
postgres=# SELECT pg_reload_conf();
pg_reload_conf
-----
t
(1 row)

postgres=# SHOW pg_store_plans.log_analyze;
pg_store_plans.log_analyze
-----
on
(1 row)

postgres=#
```

Рисунок 2.20 – Установка нового значения параметра «pg_store_plans.log_analyze»

2.1.8. Параметр «pg_store_plans.log_buffers»

Параметр «pg_store_plans.log_buffers» имеет синтаксис:

```
pg_store_plans.log_buffers (логическое значение)
```

Устанавливаются значения:

- «on», «true», «yes», «1» – включен;
- «off», «false», «no», «0» – отключен.

По умолчанию параметр отключен.

При выводе показываются значения «on»/«off».

Параметр «pg_store_plans.log_buffers» устанавливает включение в план вывода команду «EXPLAIN» с параметром «ANALYZE, BUFFERS», а не только команду «EXPLAIN». Значения, выводимые с параметром «BUFFERS», детализируют информацию на какие части запроса приходится большинство операций ввода-вывода.

Например

Посмотрим установленное значение параметра «pg_store_plans.log_buffers» через SQL-команду:

```
SHOW pg_store_plans.log_buffers;
```

```

root@ubuntu: /localrepo
postgres=# SHOW pg_store_plans.log_buffers;
pg_store_plans.log_buffers
-----
off
(1 row)

postgres=#

```

Рисунок 2.21 – Вывод значения параметра «pg_store_plans.log_buffers» по умолчанию
Значение параметра соответствует значению по умолчанию.

Установим новое значение равным «on», перезагрузим конфигурацию СУБД и проверим применение установленного параметра SQL-командами:

```

ALTER SYSTEM SET pg_store_plans.log_buffers = on;
SELECT pg_reload_conf();
SHOW pg_store_plans.log_buffers;

```

```

root@ubuntu: /localrepo
postgres=# ALTER SYSTEM SET pg_store_plans.log_buffers = on;
ALTER SYSTEM
postgres=# SELECT pg_reload_conf();
pg_reload_conf
-----
t
(1 row)

postgres=# SHOW pg_store_plans.log_buffers;
pg_store_plans.log_buffers
-----
on
(1 row)

postgres=#

```

Рисунок 2.22 – Изменение параметра «pg_store_plans.log_buffers»

2.1.9. Параметр «pg_store_plans.log_timing»

Параметр «pg_store_plans.log_timing» имеет синтаксис:

```
pg_store_plans.log_timing (логическое значение)
```

Устанавливаются значения:

- «on», «true», «yes», «1» – включен;

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

- «off», «false», «no», «0»- отключен.

По умолчанию параметр включен.

При выводе показываются значения «on»/«off».

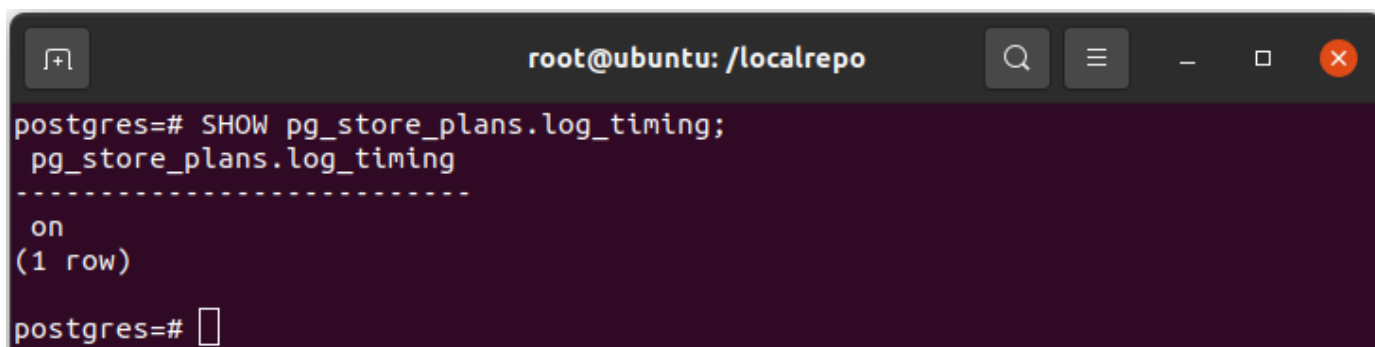
Параметр «pg_store_plans.log_timing» управляет включением/отключением записи точного времени выполнения для каждого узла выполнения. Его целесообразно использовать при включенном параметре «pg_store_plans.log_analyze» (см. п. 2.1.7).

Запись временных показателей может значительно увеличить время выполнение запроса. Для отображения фактического отображения строк запроса, параметр отключается.

Например

Просмотрим установленное значение параметра «pg_store_plans.log_timing» через SQL-команду:

```
SHOW pg_store_plans.log_timing;
```



The screenshot shows a terminal window with the title 'root@ubuntu: /localrepo'. The command 'postgres=# SHOW pg_store_plans.log_timing;' is entered, and the output is 'pg_store_plans.log_timing' followed by a dashed line and 'on'. Below this, it says '(1 row)' and then 'postgres=#' with a cursor. The terminal window has standard Ubuntu window controls at the top.

Рисунок 2.23 – Отображение установленного значения параметра «pg_store_plans.log_timing» по умолчанию

Значение параметра соответствует значению по умолчанию.

Установим новое значение равным «off», перезагрузим конфигурацию СУБД и проверим применение установленного параметра SQL-командами:

```
ALTER SYSTEM SET pg_store_plans.log_timing = off;  
SELECT pg_reload_conf();  
SHOW pg_store_plans.log_timing;
```

```
root@ubuntu: /localrepo
postgres=# ALTER SYSTEM SET pg_store_plans.log_timing = off;
ALTER SYSTEM
postgres=# SELECT pg_reload_conf();
pg_reload_conf
-----
t
(1 row)

postgres=# SHOW pg_store_plans.log_timing;
pg_store_plans.log_timing
-----
off
(1 row)

postgres=#
```

Рисунок 2.24 - Установка нового значения параметра «pg_store_plans.log_timing»

2.1.10. Параметр «pg_store_plans.log_triggers»

Параметр «pg_store_plans.log_triggers» имеет синтаксис:

```
pg_store_plans.log_triggers (логическое значение)
```

Устанавливаются значения:

- «on», «true», «yes», «1» – включен;
- «off», «false», «no», «0» – отключен.

По умолчанию параметр включен.

При выводе показываются значения «on»/«off».

Параметр «pg_store_plans.log_triggers» управляет включением/отключением записи выполнения триггера в записываемые планы.



Параметр не функционирует при отключённом параметре «pg_store_plans.log_analyze» (см. п. 2.1.7)

Например

Просмотрим установленное значение параметра «pg_store_plans.log_triggers» через SQL-команду:

```
SHOW pg_store_plans.log_buffers;
```

```

root@ubuntu: /localrepo
postgres=# SHOW pg_store_plans.log_buffers;
pg_store_plans.log_buffers
-----
on
(1 row)

postgres=#

```

Рисунок 2.25 – Просмотр значения параметра «pg_store_plans.log_triggers» по умолчанию
Установим новое значение равным «off», перезагрузим конфигурацию СУБД и проверим применение установленного параметра SQL-командами:

```

ALTER SYSTEM SET pg_store_plans.log_triggers = off;
SELECT pg_reload_conf();
SHOW pg_store_plans.log_triggers;

```

```

root@ubuntu: /localrepo
postgres=# ALTER SYSTEM SET pg_store_plans.log_triggers = off;
ALTER SYSTEM
postgres=# SELECT pg_reload_conf();
pg_reload_conf
-----
t
(1 row)

postgres=# SHOW pg_store_plans.log_triggers;
pg_store_plans.log_triggers
-----
off
(1 row)

postgres=#

```

Рисунок 2.26 - Изменение параметра «pg_store_plans.log_triggers»

2.1.11. Параметр «pg_store_plans.save»

Параметр «pg_store_plans.save» имеет синтаксис:

```
pg_store_plans.save (логическое значение)
```

Устанавливаются значения:

- «on», «true», «yes», «1» – включен;

- «off», «false», «no», «0»- отключен.

По умолчанию параметр включен.

При выводе показываются значения «on»/«off».

Параметр «pg_store_plans.save» управляет включением/отключением записи статистики плана при выключении сервера СУБД.

Например

Посмотрим установленное значение параметра «pg_store_plans.save» через SQL-команду:

```
SHOW pg_store_plans.save;
```

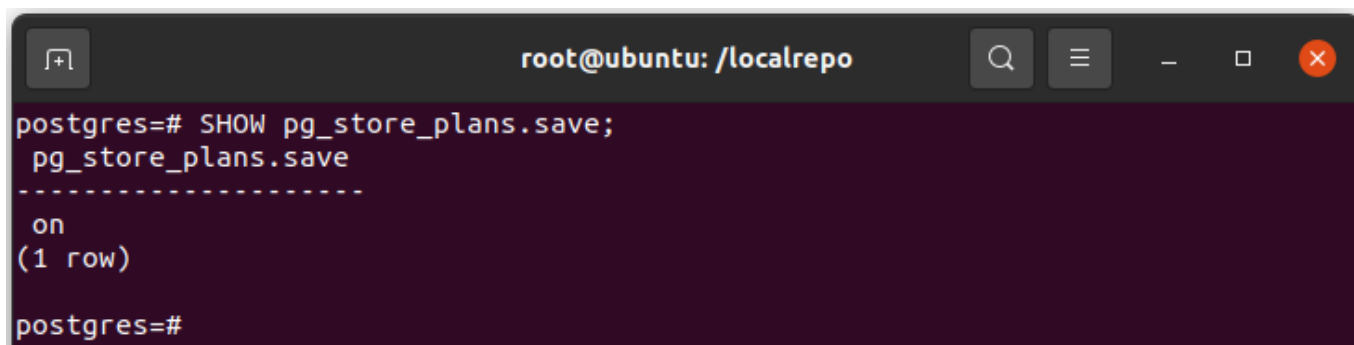
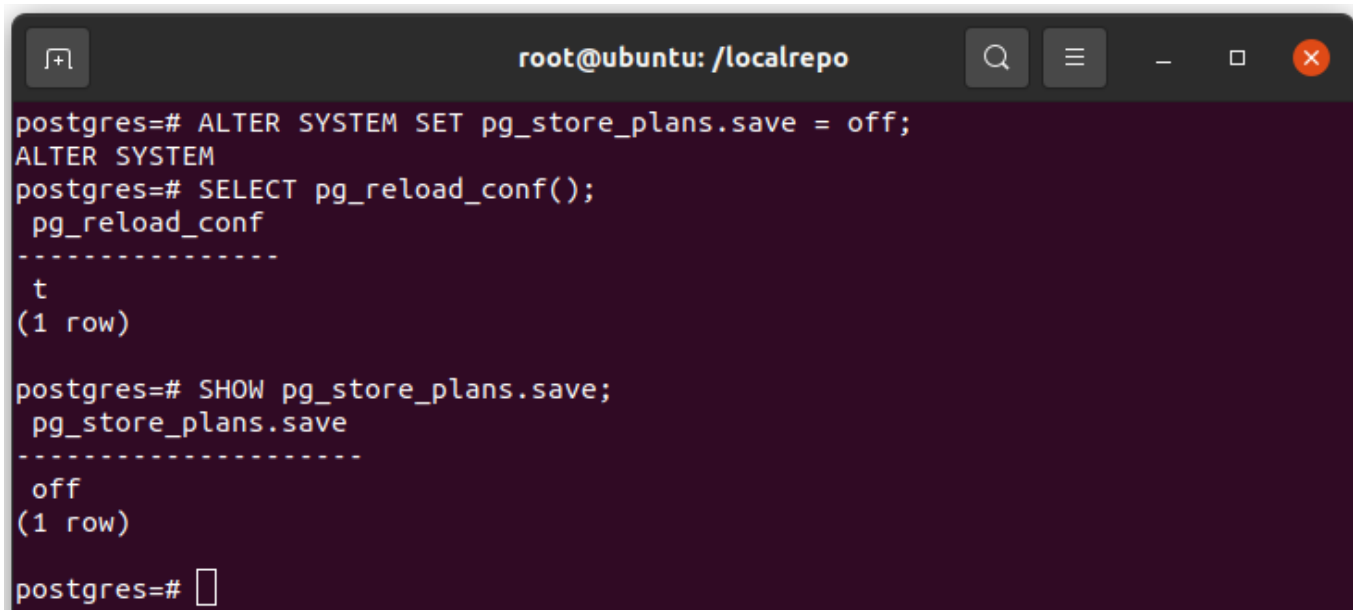


Рисунок 2.27 – Просмотр значения по умолчанию параметра «pg_store_plans.save»

Значение параметра соответствует значению по умолчанию.

Установим новое значение равным «off», перезагрузим конфигурацию СУБД и проверим применение установленного параметра SQL-командами:

```
ALTER SYSTEM SET pg_store_plans.save = off;  
SELECT pg_reload_conf();  
SHOW pg_store_plans.save;
```



```
root@ubuntu: /localrepo
postgres=# ALTER SYSTEM SET pg_store_plans.save = off;
ALTER SYSTEM
postgres=# SELECT pg_reload_conf();
 pg_reload_conf 
-----
 t
(1 row)

postgres=# SHOW pg_store_plans.save;
 pg_store_plans.save 
-----
 off
(1 row)

postgres=#
```

Рисунок 2.28 – Установка нового значения параметра «pg_store_plans.save»

3. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ КОМПОНЕНТА

В результате установки расширения сформируются два представления «pg_store_plans» (см. п. 3.1) и «pg_store_plans_info» (см п. 3.2).

В представление «pg_store_plans» аккумулирует в себе основную информацию, а представление «pg_store_plans_info» содержит в себе статистику расширения «pg_store_plans».

3.1. Представление «pg_store_plans»

В представление «pg_store_plans» аккумулирует в себе строки для:

- каждого набора идентификаторов БД;
- идентификаторов пользователей;
- идентификаторов запросов.

Столбцы представления описаны в таблице 3.1.

Таблица 3.1 - Столбцы «pg_store_plans»

Название	Тип	Описание
userid	oid	OID пользователя, выполнившего оператор, получается из системного каталога «pg_authid.oid»
dbid	oid	OID базы данных, в которой был выполнен оператор получается из системного каталога «pg_database.oid»
queryid	int8	Идентификатор запроса, сгенерированный ядром. Если для параметра calculate_query_id установлено значение "нет", pg_store_plan автоматически отключается
planid	int8	Хэш-код плана, вычисленный из нормализованного представления плана
plan	text	Текст репрезентативного плана. Формат задается параметром конфигурации pg_store_plans.plan_format.
calls	int8	Количество выполненных раз
total_time	float8	Общее время нахождения в выписке по плану, в миллисекундах
min_time	float8	Минимальное время
max_time	float8	Максимальное время
mean_time	float8	Среднее время
stddev_time	float8	Стандартное время
rows	int8	Общее количество строк, извлеченных или затронутых оператором с использованием плана
№ изменения: _____		Подпись отв. лица: _____ Дата внесения изм: _____

Название	Тип	Описание
shared_blks_hit	int8	Общее количество попаданий в кэш общих блоков оператором, использующим план
shared_blks_read	int8	Общее количество общих блоков, прочитанных оператором с использованием плана
shared_blks_dirtied	int8	Общее количество общих блоков, загаженных оператором с использованием плана
shared_blks_written	int8	Общее количество общих блоков, записанных оператором с использованием плана
local_blks_hit	int8	Общее количество попаданий в локальный кэш блоков оператором, использующим план
local_blks_read	int8	Общее количество локальных блоков, прочитанных оператором с использованием плана
local_blks_dirtied	int8	Общее количество локальных блоков, загаженных оператором с использованием плана
local_blks_written	int8	Общее количество локальных блоков, записанных оператором с использованием плана
temp_blks_read	int8	Общее количество временных блоков, прочитанных оператором с использованием плана
temp_blks_written	int8	Общее количество временных блоков, записанных оператором с использованием плана
blk_read_time	float8	Общее время, затраченное оператором, использующим план, на чтение блоков, в миллисекундах (если включена функция track_io_timing, иначе ноль) До версии компонента 1.8 включительно
blk_write_time	float8	Общее время, затраченное оператором с использованием плана на запись блоков, в миллисекундах (если включена функция track_io_timing, иначе ноль) До версии компонента 1.8 включительно
shared_blk_read_time	float8	Общее время, затраченное оператором на чтение разделяемых блоков, в миллисекундах (если включён track_io_timing, иначе ноль) Начиная с версии компонента 1.9
shared_blk_write_time	float8	Общее время, затраченное оператором на запись разделяемых блоков, в миллисекундах (если включён track_io_timing, иначе ноль) Начиная с версии компонента 1.9
local_blk_read_time	float8	Общее время, затраченное оператором на чтение локальных блоков, в миллисекундах (если включён track_io_timing, или ноль в противном случае). В версиях Postgres Pro ниже 17 всегда содержит ноль. Начиная с версии компонента 1.9

Название	Тип	Описание
local_blk_write_time	float8	Общее время, затраченное оператором на запись локальных блоков, в миллисекундах (если включён track_io_timing, или ноль в противном случае). В версиях Postgres Pro ниже 17 всегда содержит ноль. Начиная с версии компонента 1.9
temp_blk_read_time	float8	Общее время, затраченное оператором, использующим план, на чтение блоков временного файла, в миллисекундах (если включена функция track_io_timing, в противном случае ноль)
temp_blk_write_time	float8	Общее время, затраченное оператором с использованием плана на запись блоков временного файла, в миллисекундах (если включена функция track_io_timing, иначе ноль)
first_call	timestamp with time zone	Временная метка для последнего вызова запроса с использованием этого плана
last_call	timestamp with time zone	Отметка времени для последнего вызова запроса с использованием этого плана

3.2. Представление «pg_store_plans_info»

Статистика самого модуля «pg_store_plans» отслеживается и становится доступной через представление с именем «pg_store_plans_info». Это представление содержит только одну строку. Столбцы представления показаны в таблице 3.2.

Таблица 3.2 - Столбцы «pg_store_plans_info»

Имя	Тип	Описание
dealloc	bigint	Общее количество раз, когда записи pg_store_plans о наименее выполняемых инструкциях были освобождены из-за того, что наблюдалось больше отдельных инструкций, чем в pg_store_plans.max
stats_reset	timestamp with time zone	Время последнего сброса всей статистики в представлении pg_store_plans

3.3. Функции компонента

3.3.1. Функция «pg_store_plans_reset»

Функция «pg_store_plans_reset» имеет синтаксис:

```
pg_store_plans_reset() returns void
```

Функция сбрасывает всю статистику, собранную компонентом и выполняется от имени и с правами привилегированного пользователя postgres или с пользователя с атрибутом «SUPERUSER».

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

Например

Сброс статистики выполняется SQL-командой:

```
SELECT pg_store_plans_reset();
```

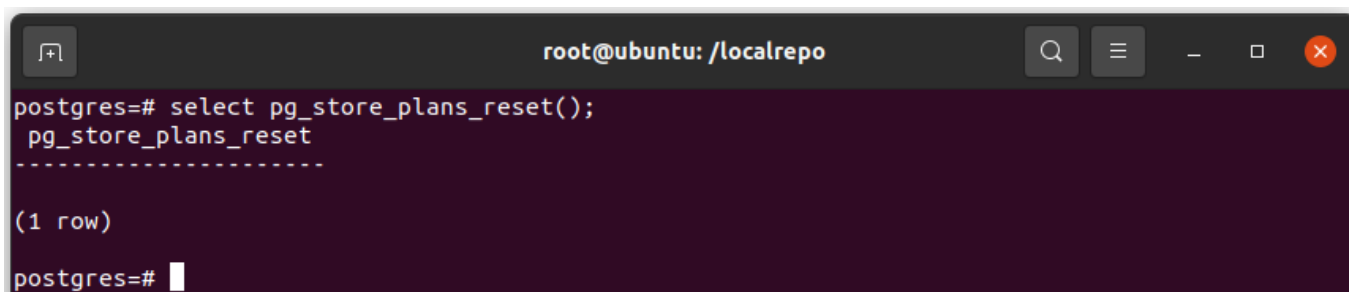


Рисунок 3.1 – SQL-команда выполнения сброса статистики

3.3.2. Функция «pg_store_plans»

Функция «pg_store_plans» служит для вывода одноименного представления «pg_store_plans» и имеет синтаксис вызова:

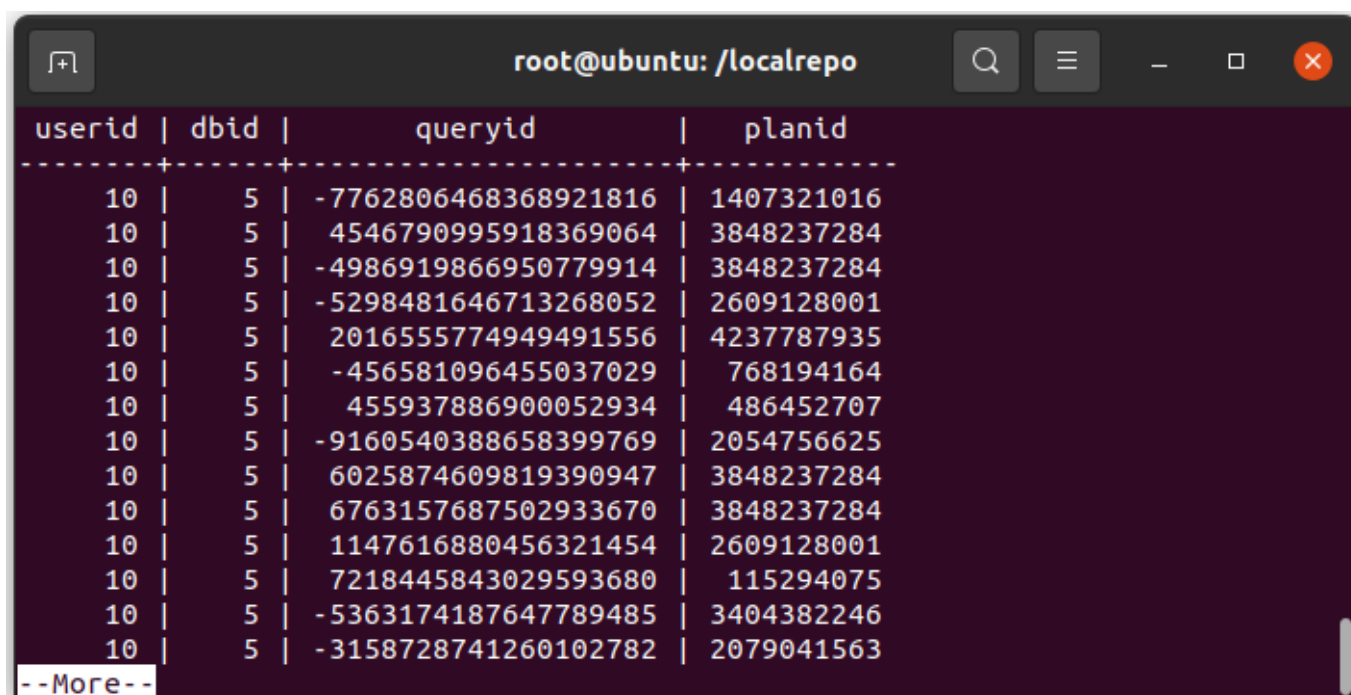
```
SELECT pg_store_plans();
```

При этом выведенная информация не будет структурирована и целесообразно вывести требуемые столбы.

Например

Выведем содержимое представления «pg_store_plans» с столбцами userid, dbid, queryid, planid. SQL-команда будет следующей:

```
SELECT userid, dbid, queryid, planid from pg_store_plans;
```



userid	dbid	queryid	planid
10	5	-7762806468368921816	1407321016
10	5	4546790995918369064	3848237284
10	5	-4986919866950779914	3848237284
10	5	-5298481646713268052	2609128001
10	5	2016555774949491556	4237787935
10	5	-456581096455037029	768194164
10	5	455937886900052934	486452707
10	5	-9160540388658399769	2054756625
10	5	6025874609819390947	3848237284
10	5	6763157687502933670	3848237284
10	5	1147616880456321454	2609128001
10	5	7218445843029593680	115294075
10	5	-5363174187647789485	3404382246
10	5	-3158728741260102782	2079041563

--More--

Рисунок 3.2 – Вывод представления «pg_store_plans»

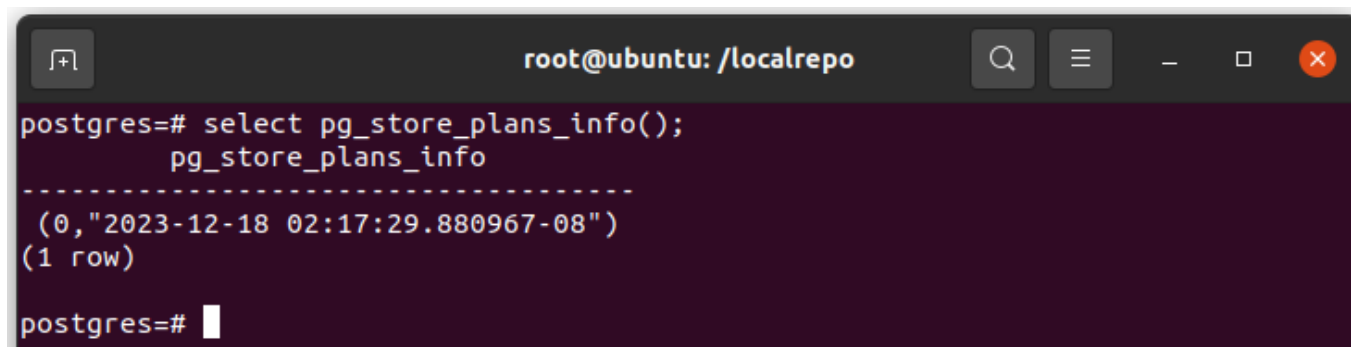
3.3.3. Функция «pg_store_plans_info»

Функция «pg_store_plans_info» предназначена для просмотра одноимённого представления.

Например

Просмотр содержания представления выполняется SQL-командой:

```
SELECT pg_store_plans_info();
```



```
postgres=# select pg_store_plans_info();
pg_store_plans_info
-----
(0, "2023-12-18 02:17:29.880967-08")
(1 row)

postgres=#
```

Рисунок 3.3 - Просмотр содержания представления «pg_store_plans_info»

3.3.4. Функция «pg_store_hash_query»

Функция «pg_store_hash_query» вычисляет хеш-значение текста SQL-запроса и имеет синтаксис:

```
pg_store_hash_query (query text)
```

Например

Вычислим хеш-значение текста SQL-запроса в созданную таблицу «orders». Для этого в теле SQL запроса указывается вызываемая функция (допускается указать схему данных в которой она расположена) и текст SQL-запроса к таблице:

```
SELECT public.pg_store_plans_hash_query('select * FROM
orders');
```

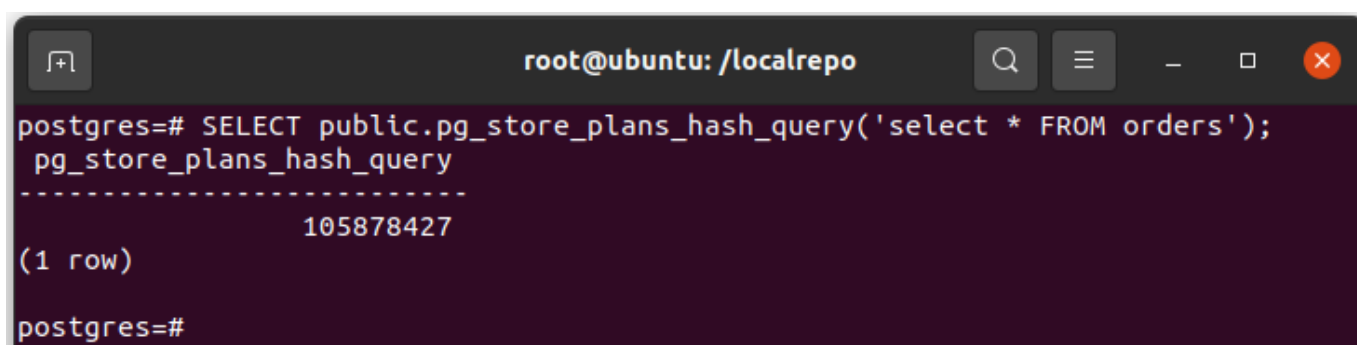


Рисунок 3.4 – Вызов функции «pg_store_hash_query»

3.3.5. Функция «pg_store_plans_textplan»

Функция «pg_store_plans_textplan» генерирует обычное текстовое представление из необработанного представления плана в «pg_store_plans», при установленном значении 'raw' параметра «pg_store_plans.plan_formats» (см. п. 2.1.3).

Поскольку текст плана результатов генерируется из представления json, он может немного отличаться от того, что вы получите непосредственно из команды «EXPLAIN».

Функция имеет синтаксис:

```
pg_store_plans_textplan(query text) returns text
```

3.3.6. Функция «pg_store_plans_xmlplan»

Функция «pg_store_plans_xmlplan» создает XML-представление из необработанного представления плана в «pg_store_plans», при установленном значении 'raw' параметра «pg_store_plans.plan_formats» (см. п. 2.1.3).

Функция имеет синтаксис:

```
pg_store_plans_xmlplan(query text) returns text
```

3.3.7. Функция «pg_store_plans_yamlplan»

Функция «pg_store_plans_yamlplan» генерирует представление YAML из необработанного представления плана в «pg_store_plans», при установленном значении 'raw' параметра «pg_store_plans.plan_formats» (см. п. 2.1.3).

Функция имеет синтаксис:

```
pg_store_plans_yamlplan(query text) returns text
```

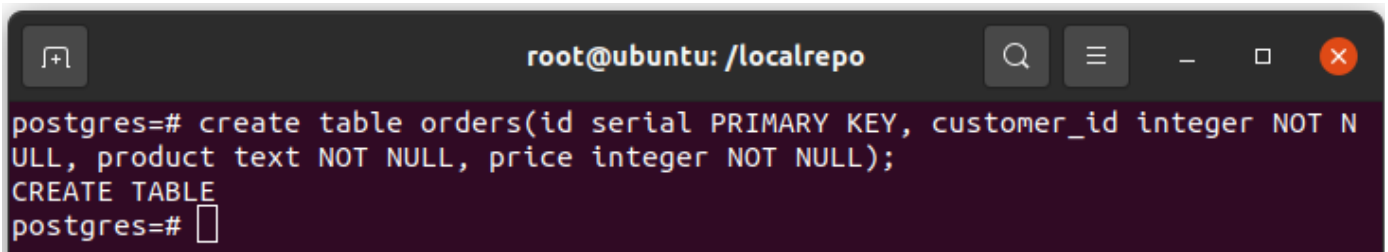
4. ПРИМЕР ВЫВОДА

В качестве примера использования компонента будет использована тестовая таблица, к которой будет создан простейший план запроса. Данному плану запроса будет дана качественная оценка компонентом «pg_store_plans».

4.1. Создание тестовой таблицы

В БД создать таблицу «orders»:

```
CREATE TABLE orders(id serial PRIMARY KEY, customer_id integer NOT NULL, product text NOT NULL, price integer NOT NULL);
```

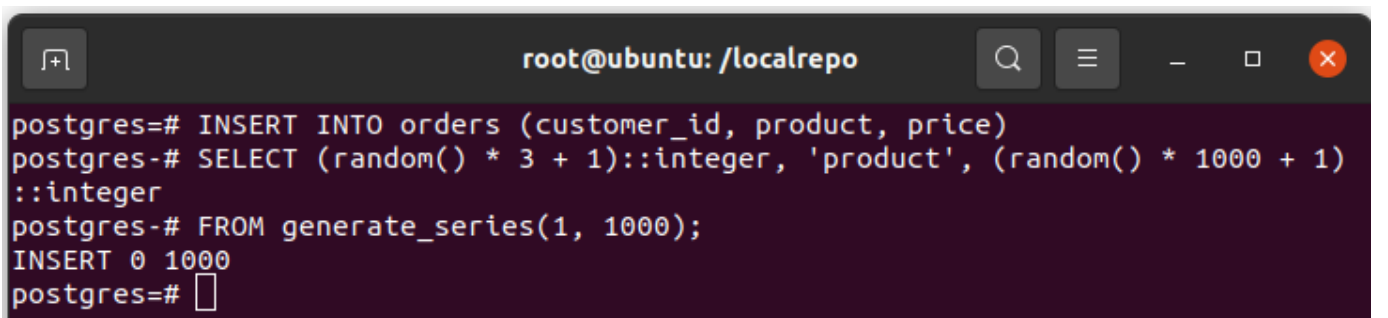


```
root@ubuntu: /localrepo
postgres=# create table orders(id serial PRIMARY KEY, customer_id integer NOT NULL, product text NOT NULL, price integer NOT NULL);
CREATE TABLE
postgres=#
```

Рисунок 4.1 – Создание таблицы «orders»

Добавить 1000 записей в таблицу «orders»:

```
INSERT INTO orders (customer_id, product, price)
SELECT (random() * 3 + 1)::integer, 'product', (random() * 1000 + 1)::integer
FROM generate_series(1, 1000);
```



```
root@ubuntu: /localrepo
postgres=# INSERT INTO orders (customer_id, product, price)
postgres-# SELECT (random() * 3 + 1)::integer, 'product', (random() * 1000 + 1)::integer
postgres-# FROM generate_series(1, 1000);
INSERT 0 1000
postgres=#
```

Рисунок 4.2 – Команда добавления записей в таблицу

4.2. Вывод оценки качества плана

Для вывода оценки качества плана потребуется:

- установить формат вывода (см. п. 2.1.3);
- перезагрузить конфигурацию СУБД;
- просмотреть установленные изменения;
- очистить имеющуюся статистику (см. п. 3.3.1);

SQL – командами:

```
ALTER SYSTEM SET pg_store_plans.plan_format = 'text';  
SELECT pg_reload_conf();  
SHOW pg_store_plans.plan_format;  
SELECT pg_store_plans_reset();
```

В качестве запроса для анализа будет использоваться SQL-запрос к таблице «orders»:

```
SELECT * FROM orders;
```

Создать план запроса с опциями:

- «ANALYZE», которая выполняет фактический запрос в реальном времени для сбора и подготовки плана выполнения;
- «SUMMARY» - добавляет итоговую информацию в план выполнения запроса;

SQL – командой:

```
EXPLAIN (ANALYZE, SUMMARY FALSE) SELECT * FROM orders;
```

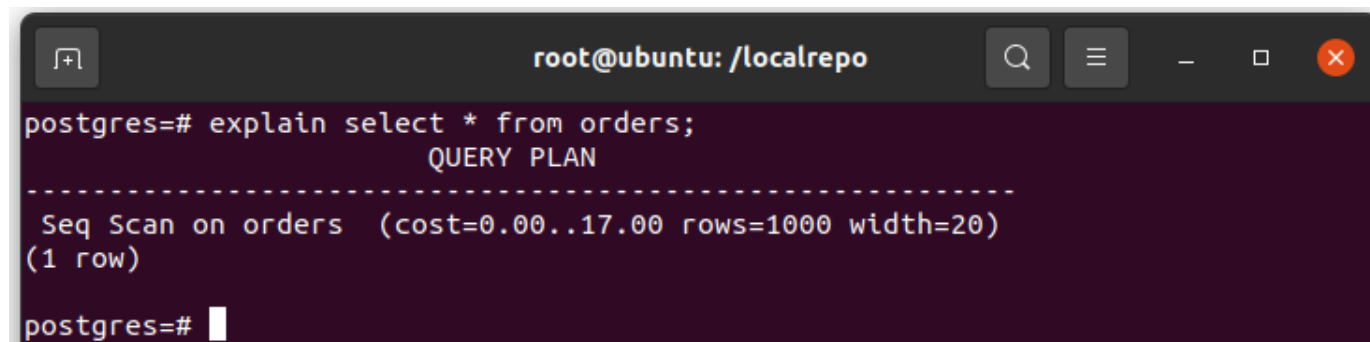


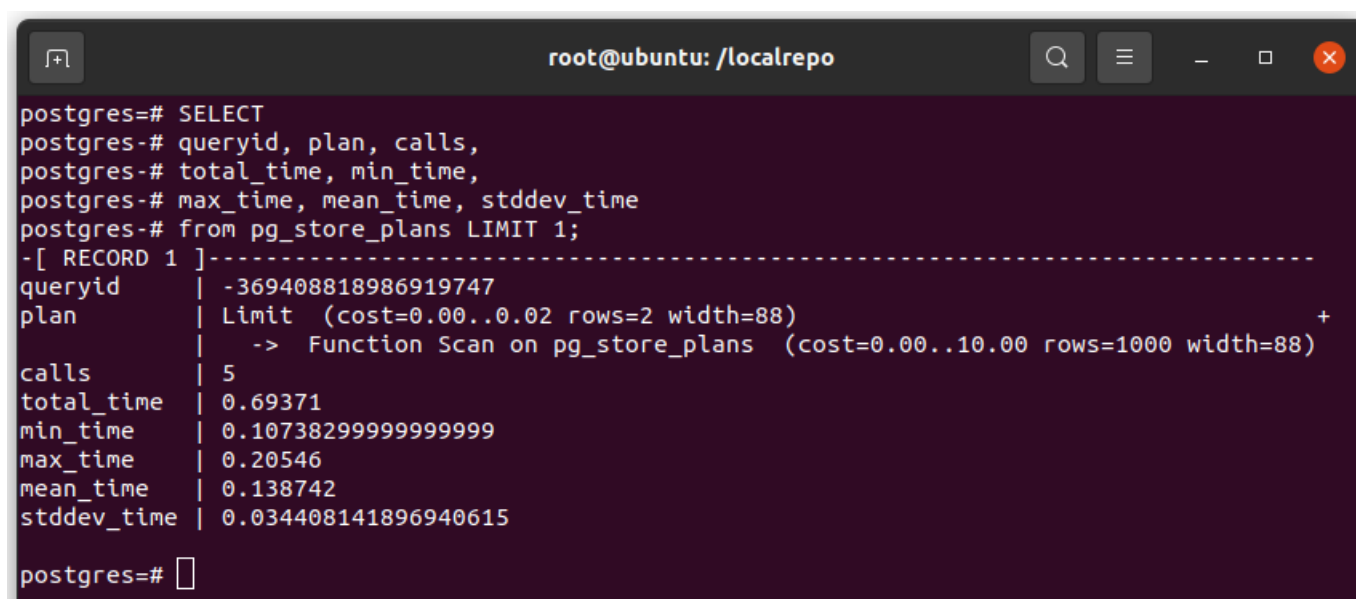
Рисунок 4.3 – Создание плана выполнения запроса

Установить расширенный вывод в СУБД:

```
\x
```

Вывести характеристики плана выполнения запроса:

```
SELECT
queryid, plan, calls,
total_time, min_time,
max_time, mean_time, stddev_time
from pg_store_plans LIMIT 1;
```



```
root@ubuntu: /localrepo
postgres=# SELECT
postgres=# queryid, plan, calls,
postgres=# total_time, min_time,
postgres=# max_time, mean_time, stddev_time
postgres=# from pg_store_plans LIMIT 1;
-[ RECORD 1 ]-----+
queryid          | -369408818986919747
plan              | Limit (cost=0.00..0.02 rows=2 width=88)
                  | -> Function Scan on pg_store_plans (cost=0.00..10.00 rows=1000 width=88)
calls            | 5
total_time       | 0.69371
min_time         | 0.10738299999999999
max_time         | 0.20546
mean_time        | 0.138742
stddev_time      | 0.034408141896940615
postgres=#
```

Рисунок 4.4 – Вывод оценки выполнения плана запроса

Вывод оценочной информации в SQL-запросе может дополняться строками из представления «pg_store_plans», приведенными в таблице 3.1.

5. ОБНОВЛЕНИЕ КОМПОНЕНТА

5.1. Обновление компонента `pg_store_plans` в ОС GNU/Linux

Предварительные условия: выполнено обновление СУБД «Jatoba» до версии 18 согласно документу «Руководство по обновлению СУБД Jatoba» 643.72410666.00067-07 93 01.



При обновлении СУБД «Jatoba» до версии 18 необходимо отключить функцию подсчета контрольных сумм при инициализации каталога данных. Сведения о процедурах обновления до СУБД «Jatoba» 18 изложены в документе «Руководство по обновлению» 643.72410666.00067-07 93 01

Для обновления компонента `pg_store_plans` с версии 1.8 включительно до версии 1.9 необходимо выполнить следующие шаги:

- 1) Остановить службу СУБД «Jatoba»:

```
# systemctl stop jatoba-18
```

- 2) Установить новую версию компонента командой:

```
# apt install jatoba18-pg-store-plans
```

- 3) В конфигурационный файл `/var/lib/jatoba/18/data/postgresql.conf` добавить параметры:

```
shared_preload_libraries = 'pg_store_plans, pg_stat_statements'  
pg_store_plans.max = 10000  
pg_store_plans.track = all
```

- 4) Выполнить запуск обновлённой СУБД «Jatoba» 18:

```
# systemctl start jatoba-18 && systemctl status jatoba-18
```

- 5) Обновить версию расширения в БД:

```
ALTER EXTENSION pg_store_plans UPDATE;
```

На данном шаге обновление компонента `pg_store_plans` завершено.

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