

ЗАЩИЩЕННАЯ СИСТЕМА УПРАВЛЕНИЯ
БАЗАМИ ДАННЫХ «ЈАТОВА»

Руководство по настройке. Часть 21.
Управление планами запросов.
Компонент "ja_Plan_Manager"

643.72410666.00067-07 98 01-21

Листов 27

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

АННОТАЦИЯ

В документе приведены сведения, необходимые для установки и эксплуатации компонента «ja_Plan_Manager» (далее по тексту – компонент или ja_Plan_Manager), предназначенного для сохранения планов запросов, их дальнейшего использования, экспорта и импорта в базах данных (БД) и оптимизации.

Настоящее руководство предназначено для администраторов СУБД.



Все примеры в данном документе приведены для СУБД «Jatoba» версии ядра 6.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 – 1.2

Степени важности примечаний, применяемые в документе:



Важная информация – указания, требующие особого внимания



Дополнительная информация – указания, позволяющие упростить работу с изделием



Важная информация

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

СОДЕРЖАНИЕ

| | |
|---|----|
| 1. Назначение компонента..... | 4 |
| 1.1. Условия применения..... | 4 |
| 2. Установка компонента..... | 5 |
| 2.1. Установка компонента «ja_Plan_Manager» в ОС GNU/Linux..... | 5 |
| 3. Настройка компонента..... | 7 |
| 3.1. Механизм работы компонента..... | 7 |
| 3.2. Настройка конфигурационного файла postgresql.conf..... | 8 |
| 3.3. Установка расширения «ja_Plan_Manager»..... | 9 |
| 4. Функциональные возможности компонента..... | 11 |
| 4.1. Включение/отключение режима сохранения плана запросов..... | 11 |
| 4.2. Включение/отключение режима использования сохраненных планов запросов..... | 12 |
| 4.3. Экспорт/импорт плана запросов..... | 12 |
| 4.3.1. Экспорт..... | 12 |
| 4.3.2. Импорт..... | 13 |
| 4.4. Просмотр сохраненных планов запросов..... | 15 |
| 4.5. Анализ планов запросов..... | 15 |
| 4.6. Журналирование отработанных планов запросов..... | 15 |
| 5. Пример реализации функциональных возможностей компонента..... | 17 |
| 5.1. Подготовка БД «test_db_a»..... | 17 |
| 5.2. Подготовка БД «test_db_b»..... | 19 |
| 5.3. Создание плана запроса на test_db_a..... | 20 |
| 5.4. Экспорт плана в каталог пользователя..... | 22 |
| 5.5. Импорт плана в БД «test_db_b»..... | 23 |
| 6. Удаление компонента..... | 25 |
| 6.1. Отключение режима использования плана запросов..... | 25 |
| Перечень сокращений..... | 26 |

1. НАЗНАЧЕНИЕ КОМПОНЕНТА

Компонент «ja_Plan_Manager» предназначен для сохранения, экспорта/импорта и подмены планов запросов в БД.

1.1. Условия применения

Компонент «ja_Plan_Manager» может использоваться с СУБД «Jatoba» версий 5.x и выше, под управлением операционной системы GNU/Linux.



В текущей реализации компонента не поддерживается управление через компонент пользовательского веб-интерфейса для администраторов «Jatoba data safe», но поддерживается установка расширения.



Ввиду различий в планировщике на разных мажорных версиях СУБД - импорт планов запросов необходимо выполнять в рамках той же версии, из которой выполнялся экспорт.

Ограничений по совместимости с другими компонентами нет.

2. УСТАНОВКА КОМПОНЕНТА

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 «ja_Plan_Manager» 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 jatoba<ver>-ja-plan-manager
```

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

```
yum install jatoba<ver>-ja-plan-manager
```

Отдельного уточнения требуют операционные системы ALT Linux и openSUSE.

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

```
apt-get install jatoba<ver>-ja-plan-manager
```

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

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

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

3. НАСТРОЙКА КОМПОНЕНТА

3.1. Механизм работы компонента

Компонент работает, используя принцип импорта/экспорта. Для экспорта используется домашний каталог пользователя postgres, либо указанный администратором СУБД.

Сначала формируется план запросов в одной БД (test_db_a), он в виде файла экспортируется в папку обмена.

Затем из папки обмена или через соединение dblink загружается в другую БД (test_db_b). Для этого необходимо:

- настроить конфигурационный файл для каждой БД;
- установить расширение для каждой БД;
- выполнить действия по экспорту/импорту плана запросов.

Схема работы компонента представлена на рисунке 3.1.

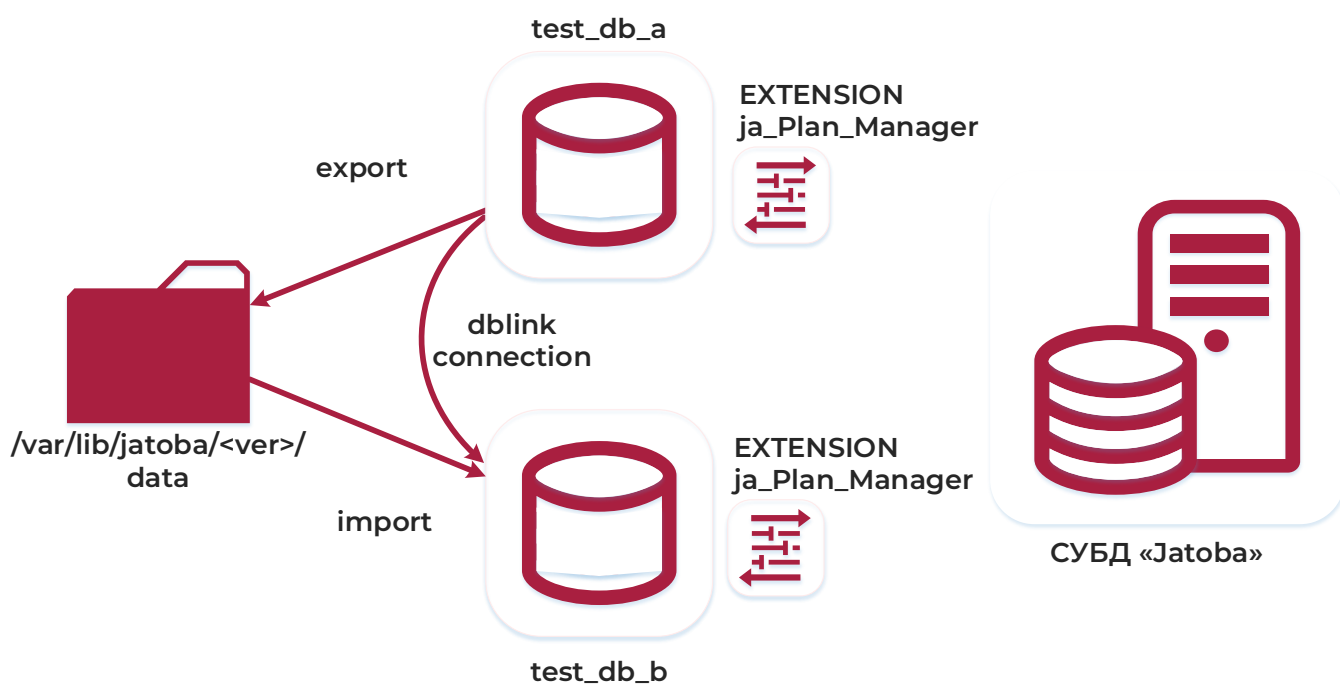


Рисунок 3.1 – Схема работы компонента

Компонент хранит планы запросов в таблице `ja_plan_manager.jpm_plans`. Столбцы таблицы описаны в таблице 3.1.

Таблица 3.1 – Описание столбцов таблицы jpm_plans

| Название столбца | Описание |
|------------------|---|
| query | запрос (записывается в параметризованном виде) |
| query_hash | хеш от запроса |
| query_id | идентификатор запроса (вычисляется только при включении системной функции compute_query_id) |
| enable | включение (true) / отключение (false) режима использования плана запроса |
| plan | план запроса |
| plan_hash | хеш от плана запроса |
| description | описание (заполняется пользователем при необходимости) |
| reloads | системный столбец, необходимый для работы компонента |
| index_reloads | системный столбец, необходимый для работы компонента |

3.2. Настройка конфигурационного файла postgresql.conf

Установка расширения «ja_plan_manager» требует, чтобы в конфигурационном файле postgresql.conf были заданы параметры загрузки разделяемых библиотек, которые будут загружаться при запуске сервера СУБД.

Файл расположен в каталоге:

```
/var/lib/jatoba/<ver>/data/
```

Откройте его в режиме редактирования и в разделе «Shared Library Preloading», для последующей загрузки расширения установите параметр:

```
shared_preload_libraries = 'ja_plan_manager'
```



Рисунок 3.2 – Параметры конфигурационного файла postgresql.conf

Для применения параметров потребуется перезапустить СУБД и проверить статус демона «jatoba-5».

```
systemctl restart jatoba-<ver>  
systemctl status jatoba-<ver>
```

В случае, когда базы данных находятся в разных инсталляциях СУБД, вышеописанные действия проводятся в каждой из СУБД.

3.3. Установка расширения «ja_Plan_Manager»

После перезагрузки СУБД и загрузки расширения станет доступной установка расширения «ja_plan_manager». Расширение должно быть установлено в каждой базе данных, в которых планируются проводить манипуляции по экспорту/импорту планов запросов. В рассматриваемом примере расширение должно быть установлено в тестовых базах данных:

- test_db_a;
- test_db_b.

Расширение устанавливается SQL-командой:

```
CREATE EXTENSION ja_plan_manager;
```

Просмотреть расширения БД можно SQL-командой:

```
\dx
```

```

root@ubuntu: /home/admin1
test_db_b=# \connect test_db_a
You are now connected to database "test_db_a" as user "postgres".
test_db_a=# CREATE EXTENSION ja_plan_manager;
CREATE EXTENSION
test_db_a=# \dx

              List of installed extensions
  Name          | Version | Schema  | Description
-----+-----+-----+-----
ja_plan_manager | 1.2     | public  | functions to save and read plan
plpgsql         | 1.0     | pg_catalog | PL/pgSQL procedural language
(2 rows)

test_db_a=# 

```

Рисунок 3.4 – Команда установки расширения в «test_db_a»

Аналогичные действия выполняются для второй тестовой БД «test_db_b»

```

# \connect test_db_b
# CREATE EXTENSION ja_plan_manager;
# \dx

```

```

root@ubuntu: /home/admin1
postgres=# \connect test_db_b
You are now connected to database "test_db_b" as user "postgres".
test_db_b=# CREATE EXTENSION ja_plan_manager;
CREATE EXTENSION
test_db_b=# \dx

              List of installed extensions
  Name          | Version | Schema  | Description
-----+-----+-----+-----
ja_plan_manager | 1.2     | public  | functions to save and read plan
plpgsql         | 1.0     | pg_catalog | PL/pgSQL procedural language
(2 rows)

test_db_b=# 

```

Рисунок 3.5 – Команда установки расширения в «test_db_b»

4. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ КОМПОНЕНТА

Функциональные возможности компонента позволяют:

- включать/отключать режим сохранения запросов;
- включать/отключать режим использования сохраненных запросов;
- экспортировать планы запросов в формате:
 - json;
 - text;
 - xml;
 - yaml;
- экспортировать планы запросов через:
 - домашний каталог пользователя;
 - строку соединения с другой БД (dblink connection);
- импортировать планы запросов.

При использовании компонента есть возможность использовать кириллицу в именах объектов СУБД «Jatoba».

4.1. Включение/отключение режима сохранения плана запросов

Для включения режима сохранения плана запросов устанавливается переменная SQL-командой:

```
SET ja_plan_manager.write_mode = true;
```

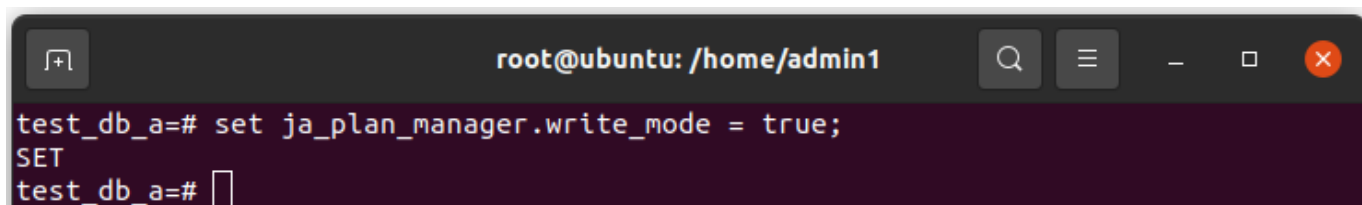


Рисунок 4.1 – SQL-команда включения режима сохранения плана запросов

В режиме сохранения плана запросов данные будут сохраняться в таблицу «ja_plan_manager.jpm_plans». В таблицу будут сохраняться только разные планы, даже если они были сгенерированы для одинакового запроса.

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

Отключается режим сохранения планов запросов SQL-командой:

```
SET ja_plan_manager.write_mode = false;
```

4.2. Включение/отключение режима использования сохраненных планов запросов

Для установления режима использования сохраненных планов запросов устанавливается переменная SQL-командой:

```
UPDATE ja_plan_manager.jpm_plans SET enable=true;
```

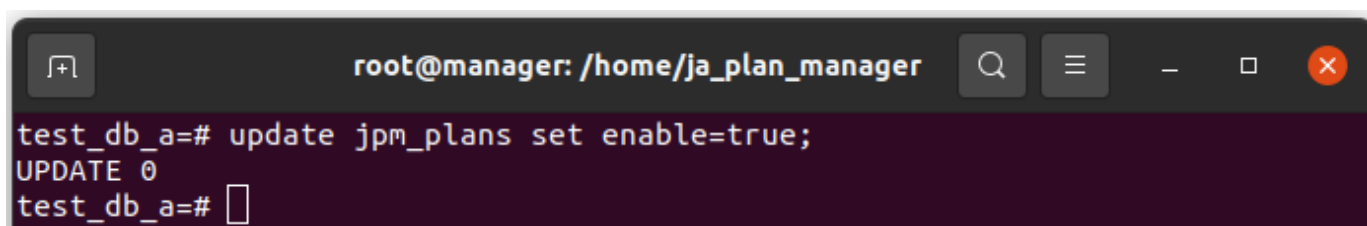


Рисунок 4.2 – Команда установления режима использования сохраненных запросов

После этого шага план для запроса будет взят из таблицы «ja_plan_manager.jpm_plans».

Одному и тому же запросу могут соответствовать разные планы. Если включен режим использования одного плана запроса, второй план к этому запросу невозможно будет включить: сработает триггер.

4.3. Экспорт/импорт плана запросов

4.3.1. Экспорт

Чтобы экспортировать план запроса через файл json, необходимо выполнить следующий SQL-запрос:

```
SELECT ja_plan_manager.ja_export_plan (<query_hash>,  
<file_path>);
```

Формат вывода указывается в расширении имени файла экспорта. Поддерживаются следующие форматы:

- json;
- text;
- xml;

- yaml.

4.3.1.1 Экспорт через «dblink connection»

Чтобы экспортировать план запроса непосредственно в другую базу данных с которой есть соединение, необходимо выполнить следующий SQL-запрос:

```
SELECT ja_plan_manager.ja_copy_plan(<connection_string>,  
<query_hash>);
```

План будет автоматически импортирован принимающей базой данных без необходимости вызова там каких-либо функций.

Формат «connection_string» совпадает с обычной строкой подключения «dblink». Он должен содержать параметры, необходимые для идентификации базы данных и авторизации импорта (имя базы данных, хост, порт, суперпользователь и пароль).

После импорта плана необходимо активировать его, чтобы он был выполнен соответствующим SQL-запросом:

```
UPDATE ja_plan_manager.jpm_plans SET enable = true;
```

4.3.2. Импорт

Для импорта плана запроса необходимо выполнить следующую SQL-команду:

```
SELECT ja_plan_manager.ja_import_plan(<query_hash>,  
<file_path>);
```

4.3.2.1 Импорт в другую версию СУБД

В случае если план запросов создан в другой версии СУБД «Jatoba», при выполнении его импорта в новую версию может отличаться результат query_hash. В этом случае импортированный план запросов не будет задействован в СУБД.

Для того чтобы задействовать импортированный план запросов администратору СУБД необходимо:

- 1) Установить новую версию СУБД.
- 2) Установить (см. раздел 2) и настроить (см. раздел 3) расширение ja_Plan_Manager так, как это указано в данном руководстве.

- 3) Создать таблицы и заполнить их теми же данными как при использовании на предыдущей версии СУБД.
- 4) Включить режим сохранения планов запросов (см. п. 4.1).
- 5) Выполнить запросы так же как на предыдущей версии СУБД.
- 6) Выключить режим сохранения планов запросов (см. п. 4.1).
- 7) Получить данные query_hash из таблицы ja_plan_manager.jpmp_plans:

```
SELECT * FROM ja_plan_manager.jpmp_plans;
```

- 8) Обновить значения query_hash планов запросов в JSON-файлах (которые использовались для экспорта в предыдущей версии СУБД) новыми, полученными в пункте 6).

Важно обращать внимание на наименование запросов и соответствующее значение query_hash.

- 9) Удалить из БД сохраненные планы запросов.
- 10) Выполнить импорт планов запросов из предыдущей версии СУБД с исправленными значениями query_hash (см. п. 4.3.2).

4.3.2.2 Импорт через «dblink connection»

Для прямой передачи плана запроса на другую базу внутри сети можно использовать «dblink». Для этого необходимо вызвать следующий SQL-запрос:

```
SELECT ja_plan_manager.ja_copy_plan(<connection_string>,  
<query_hash>);
```

План запроса будет автоматически импортирован принимающей базой данных без необходимости вызывать какие-либо функции в ней.

Формат «connection_string» аналогичен формату «connstr» в функциях «dblink». В ней должна содержаться информация, достаточная для подключения к базе и авторизации в ней:

- dbname;
- host;
- port;

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

- user;
- password – пароль суперпользователя.

После импорта плана необходимо активировать его, чтобы при вызове соответствующего ему запроса выполнялся именно импортированный план:

```
UPDATE ja_plan_manager.jpm_plans SET enable = true;
```

4.4. Просмотр сохраненных планов запросов

С помощью функции «show_plan» можно просмотреть сохраненные планы. Для этого требуется знать хеш запроса, который может быть извлечен из таблицы «ja_plan_manager.jpm_plans», и выполнить SQL-запрос:

```
SELECT ja_plan_manager.show_plan(<query_hash>);
```

4.5. Анализ планов запросов

Анализ плана запроса выполняется SQL-командой:

```
EXPLAIN ANALYZE <query>;
```

4.6. Журналирование отработанных планов запросов

При использовании плана запроса возникнет уведомление (notice), сообщающее, план с каким query_hash был отработан:

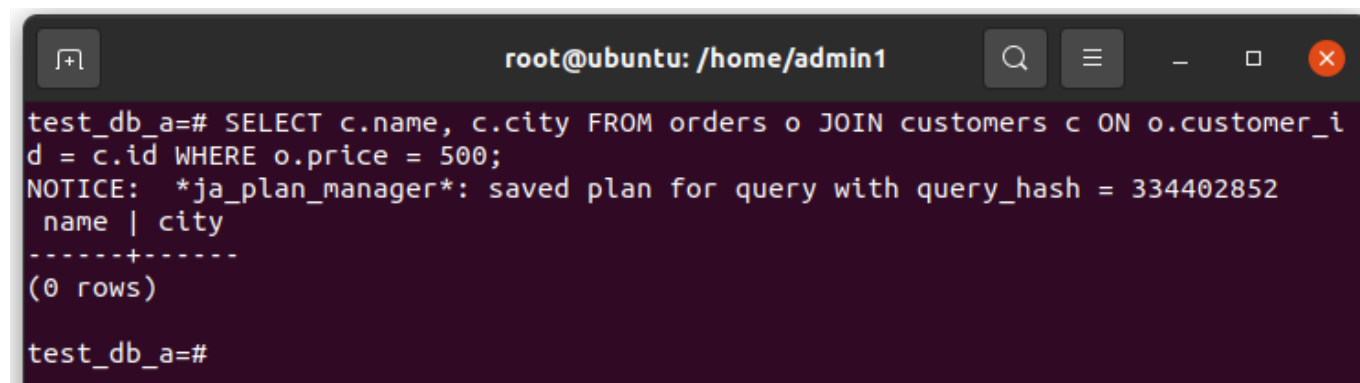


Рисунок 4.3 – Уведомление о срабатывании плана запроса

В файле журнала событий СУБД «Jatoba» также появится запись о срабатывании плана:

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

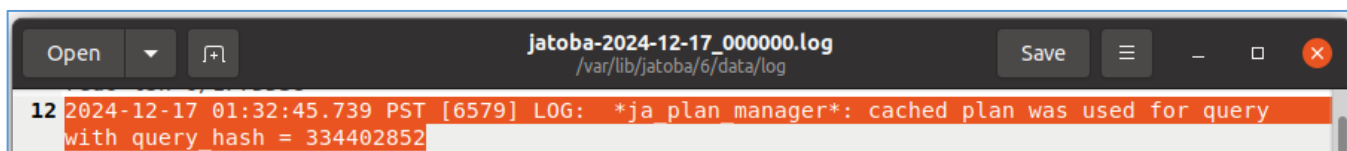


Рисунок 4.4 – Файл журнала событий с записью о срабатывании плана запроса

Файлы журнала событий находятся в директории:

```
/var/lib/jatoba/<ver>/data/log/
```

Запись в журнале событий появится также если отработан запрос, план которого есть в таблице «ja_plan_manager.jpmm_plans», но он не включен (enable=false).

5. ПРИМЕР РЕАЛИЗАЦИИ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ КОМПОНЕНТА

В качестве примера использования компонента будут рассмотрены две тестовые БД, находящиеся в одной СУБД, у которых уже установлено расширение «ja_plan_manager».

- test_db_a;
- test_db_b.

В базах данных создаются таблицы «customers» и «orders». Таблица «customers» содержит три записи, а таблица «orders» будет содержать тысячу записей.

5.1. Подготовка БД «test_db_a»

В БД «test_db_a» создать таблицу «customers»:

```
CREATE table customers(id serial PRIMARY KEY, name text not null, city text not null);
```

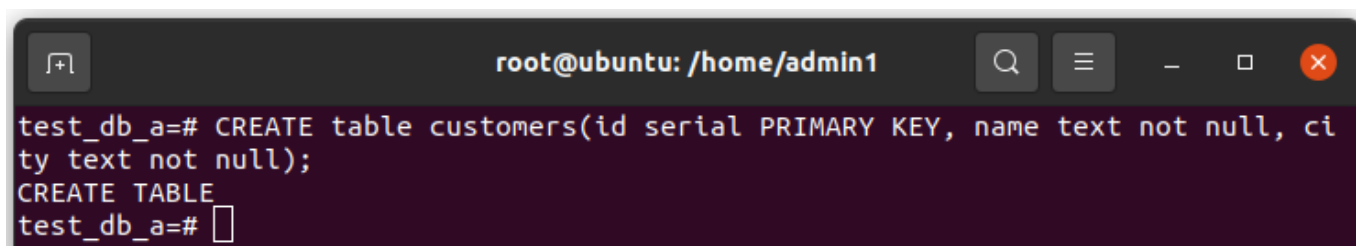
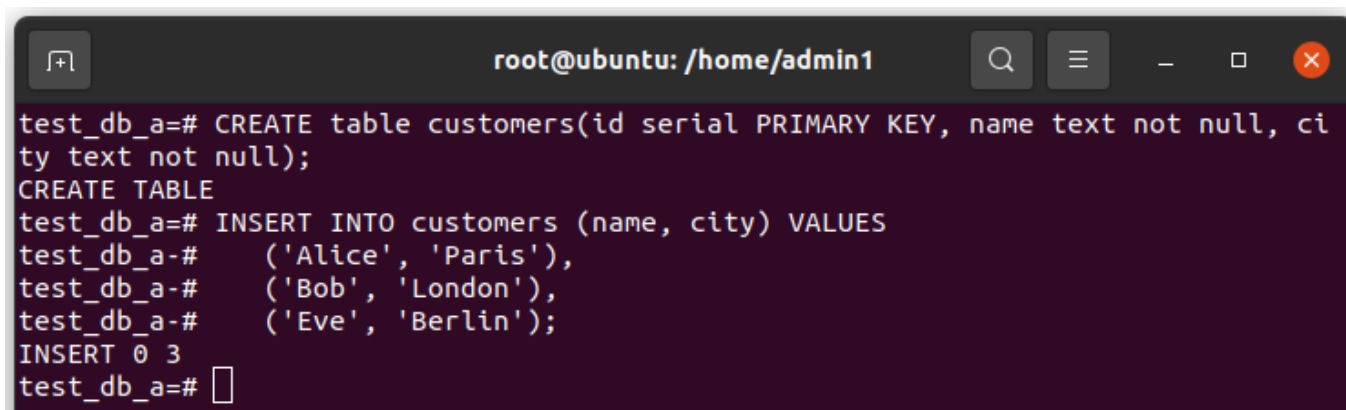


Рисунок 5.1 – Создание таблицы «customers» в БД «test_db_a»

Добавить три записи в таблицу «customers»:

```
INSERT INTO customers (name, city) VALUES  
('Alice', 'Paris'),  
('Bob', 'London'),  
('Eve', 'Berlin');
```

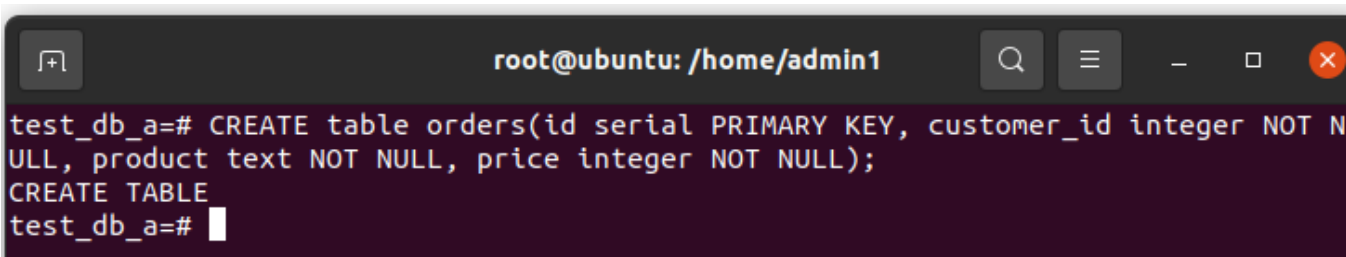


```
root@ubuntu: /home/admin1
test_db_a=# CREATE table customers(id serial PRIMARY KEY, name text not null, ci
ty text not null);
CREATE TABLE
test_db_a=# INSERT INTO customers (name, city) VALUES
test_db_a-#      ('Alice', 'Paris'),
test_db_a-#      ('Bob', 'London'),
test_db_a-#      ('Eve', 'Berlin');
INSERT 0 3
test_db_a=#
```

Рисунок 5.2 – Добавление значений в таблицу «customers» в БД «test_db_a»

В БД «test_db_a» создать таблицу «orders»:

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

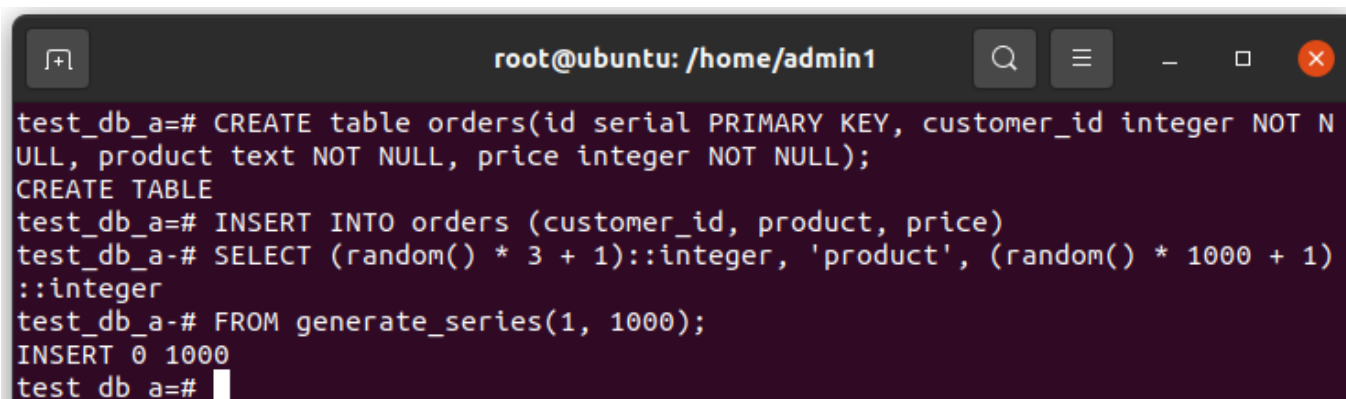


```
root@ubuntu: /home/admin1
test_db_a=# CREATE table orders(id serial PRIMARY KEY, customer_id integer NOT N
ULL, product text NOT NULL, price integer NOT NULL);
CREATE TABLE
test_db_a=#
```

Рисунок 5.3 – Создание таблицы «orders» в БД «test_db_a»

Добавьте 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: /home/admin1
test_db_a=# CREATE table orders(id serial PRIMARY KEY, customer_id integer NOT N
ULL, product text NOT NULL, price integer NOT NULL);
CREATE TABLE
test_db_a=# INSERT INTO orders (customer_id, product, price)
test_db_a-# SELECT (random() * 3 + 1)::integer, 'product', (random() * 1000 + 1)
::integer
test_db_a-# FROM generate_series(1, 1000);
INSERT 0 1000
test_db_a=#
```

Рисунок 5.4 – Добавление значений в таблицу «orders» в БД «test_db_a»

5.2. Подготовка БД «test_db_b»

В БД «test_db_b» создать таблицу «customers»:

```
CREATE TABLE customers(id serial PRIMARY KEY, name text not null, city text not null);
```

Добавить три записи в таблицу «customers»:

```
INSERT INTO customers (name, city) VALUES  
('Alice', 'Paris'),  
('Bob', 'London'),  
('Eve', 'Berlin');
```

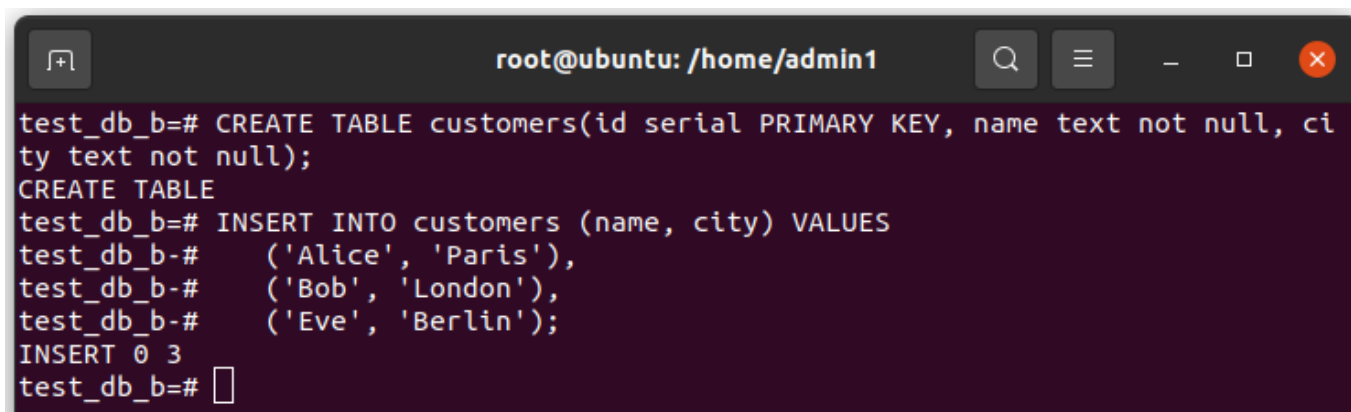


Рисунок 5.5 – Создание и добавление значений в таблицу «customers» в БД «test_db_b»

В БД «test_db_b» создать таблицу «orders»:

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

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: /home/admin1
test_db_b=# CREATE table orders(id serial PRIMARY KEY, customer_id integer NOT NULL, product text NOT NULL, price integer NOT NULL);
CREATE TABLE
test_db_b=# INSERT INTO orders (customer_id, product, price)
test_db_b=# SELECT (random() * 3 + 1)::integer, 'product', (random() * 1000 + 1)::integer
test_db_b=# FROM generate_series(1, 1000);
INSERT 0 1000
test_db_b=#
```

Рисунок 5.6 – Создание и добавление значений в таблицу «orders» в БД «test_db_b»

Для последующего сравнения планов запроса необходимо создать индекс для таблицы «orders» в БД «test_db_b»:

```
CREATE index idx_orders_price on orders(price);
```

```
root@ubuntu: /home/admin1
test_db_b=# CREATE index idx_orders_price on orders(price);
CREATE INDEX
test_db_b=#
```

Рисунок 5.7 – Создание индекса

5.3. Сознание плана запроса на test_db_a

Установить соединение с БД «test_db_a» и выполнить анализ плана запроса:

```
EXPLAIN ANALYZE SELECT c.name, c.city FROM orders o JOIN
customers c ON o.customer_id = c.id WHERE o.price = 500;
```

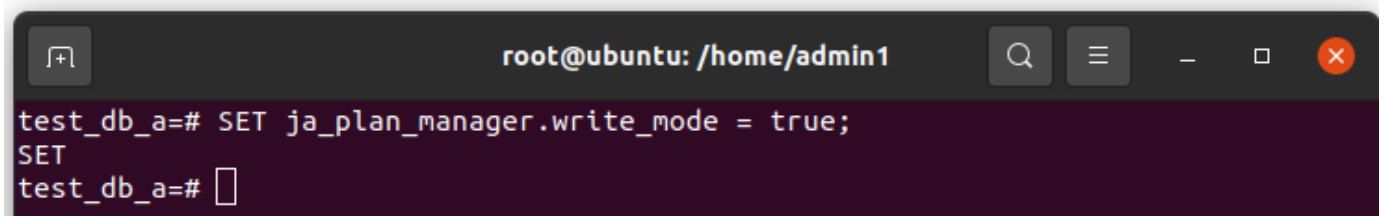
СУБД выведет параметры плана запроса.

```
root@ubuntu: /home/admin1
test_db_a=# EXPLAIN ANALYZE SELECT c.name, c.city FROM orders o JOIN customers c ON o.customer_id = c.id WHERE o.price = 500;
QUERY PLAN
-----
Nested Loop  (cost=0.15..27.69 rows=1 width=64) (actual time=0.066..0.066 rows=0 loops=1)
-> Seq Scan on orders o  (cost=0.00..19.50 rows=1 width=4) (actual time=0.066..0.066 rows=0 loops=1)
    Filter: (price = 500)
    Rows Removed by Filter: 1000
-> Index Scan using customers_pkey on customers c  (cost=0.15..8.17 rows=1 width=68) (never executed)
    Index Cond: (id = o.customer_id)
Planning Time: 0.121 ms
Execution Time: 0.078 ms
(8 rows)
test_db_a=#
```

Рисунок 5.8 – Вывод плана запроса

Включить режим сохранения плана запросов, установив переменную «write_mode» в режим «true» SQL-командой:

```
SET ja_plan_manager.write_mode = true;
```

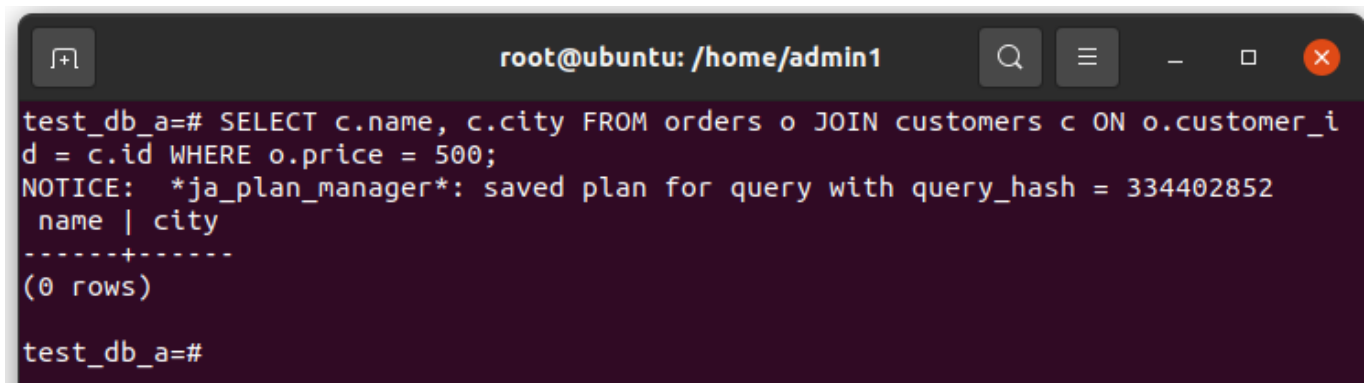


```
root@ubuntu: /home/admin1
test_db_a=# SET ja_plan_manager.write_mode = true;
SET
test_db_a=#
```

Рисунок 5.9 – Режим сохранения плана запросов

Затем выполнить SQL-запрос, который запишется в таблицу «jpm_plans»:

```
SELECT c.name, c.city FROM orders o JOIN customers c ON
o.customer_id = c.id WHERE o.price = 500;
```

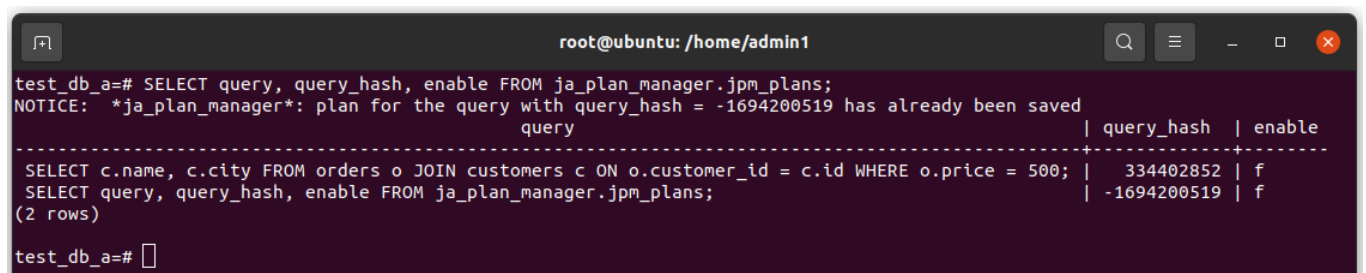


```
root@ubuntu: /home/admin1
test_db_a=# SELECT c.name, c.city FROM orders o JOIN customers c ON o.customer_i
d = c.id WHERE o.price = 500;
NOTICE: *ja_plan_manager*: saved plan for query with query_hash = 334402852
 name | city
-----+-----
(0 rows)
test_db_a=#
```

Рисунок 5.10 – Выполнение SQL-запроса

Из записи таблицы «ja_plan_manager.jpm_plans» вывести присвоенное значение «query_hash»:

```
SELECT query, query_hash, enable FROM
ja_plan_manager.jpm_plans;
```



```
root@ubuntu: /home/admin1
test_db_a=# SELECT query, query_hash, enable FROM ja_plan_manager.jpm_plans;
NOTICE: *ja_plan_manager*: plan for the query with query_hash = -1694200519 has already been saved
      query      | query_hash | enable
-----+-----+-----
SELECT c.name, c.city FROM orders o JOIN customers c ON o.customer_id = c.id WHERE o.price = 500; | 334402852 | f
SELECT query, query_hash, enable FROM ja_plan_manager.jpm_plans; | -1694200519 | f
(2 rows)
test_db_a=#
```

Рисунок 5.11 – Вывод присвоенного значения

Полученный хеш запроса равен «334402852». Полученное значение потребуется для экспорта.

5.4. Экспорт плана в каталог пользователя

Выполнить экспорт плана запроса, указав формат вывода JSON в расширении файла, значение «query_hash» равное «334402852» и путь к каталогу пользователя:

```
SELECT ja_plan_manager.ja_export_plan(334402852,
'/var/lib/jatoba/6/data/plan.json');
```



Экспорт плана запроса должен выполняться с тем же присвоенным значением хеш (query_hash).

При вводе не верного значения рационального числа хеш в SQL-команде экспорта будет создан пустой файл плана запроса.

```
root@ubuntu: /home/admin1
test_db_a=# SELECT ja_plan_manager.ja_export_plan(334402852, '/var/lib/jatoba/6/
data/plan.json');
ja_export_plan
-----
Successful export of plans to /var/lib/jatoba/6/data/plan.json
(1 row)
test_db_a=#
```

Рисунок 5.12 – SQL-команда экспорта плана запроса

В результате по указанному пути будет создан файл плана запроса в формате JSON.

```
plan.json
/var/lib/jatoba/6/data
1 {
2   "query_hash": 334402852,
3   "query_id": 0,
4   "plan_hash": 656919638,
5   "enable": false,
6   "query": "SELECT c.name, c.city FROM orders o JOIN customers c ON o.customer_id = c.id
WHERE o.price = 500;",
7   "relocks": [
8     {"relname": "orders", "relid": "16496", "schema": "public"},
9     {"relname": "customers", "relid": "16487", "schema": "public"}
10  ],
11  "index_relocks": [
12    {"relname": "customers_pkey", "relid": "16493", "schema": "public"}
13  ],
14  "plan": "{PLANNEDSTMT :commandType 1 :queryId 0 :hasReturning false :hasModifyingCTE
false :canSetTag true :transientPlan false :dependsOnRole false :parallelModeNeeded
false :jitFlags 0 :planTree {NESTLOOP :join.plan.startup_cost 0.15 :join.plan.total_cost
38.35384253819036 :join.plan.plan_rows 2 :join.plan.plan_width 64 :join.plan.parallel_aware
```

Рисунок 5.13 – Экспортированный план запроса

5.5. Импорт плана в БД «test_db_b»

Установить соединение с БД «test_db_b» и выполнить импорт плана запроса из каталога пользователя SQL-командой:

```
SELECT ja_plan_manager.ja_import_plan(334402852,  
'/var/lib/jatoba/6/data/plan.json');
```

Импортированный план запроса запишется в таблицу «jpm_plans».

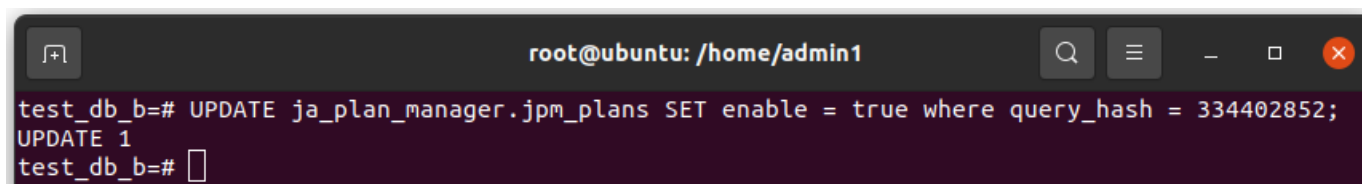


```
root@ubuntu: /home/admin1  
test_db_a=# \connect test_db_b  
You are now connected to database "test_db_b" as user "postgres".  
test_db_b=# SELECT ja_plan_manager.ja_import_plan(334402852, '/var/lib/jatoba/6/data/plan.json');  
ja_import_plan  
-----  
-----  
Successful import of plans from /var/lib/jatoba/6/data/plan.json  
+  
For the plan to work, the enable flag must be set to true:  
+  
      'UPDATE ja_plan_manager.jpm_plans SET enable = true WHERE query_hash =  
334402852;'  
(1 row)
```

Рисунок 5.14 – Импорт плана запроса

Чтобы использовать импортированный план запроса, необходимо включить режим использования сохраненных планов запросов SQL-командой:

```
UPDATE ja_plan_manager.jpm_plans SET enable = true where  
query_hash = 334402852;
```



```
root@ubuntu: /home/admin1  
test_db_b=# UPDATE ja_plan_manager.jpm_plans SET enable = true where query_hash = 334402852;  
UPDATE 1  
test_db_b=#
```

Рисунок 5.15 – Режим использования сохраненных планов запросов

Затем необходимо выполнить анализ плана запроса, чтобы убедиться в применении плана запроса:

```
EXPLAIN ANALYZE SELECT c.name, c.city FROM orders o JOIN  
customers c ON o.customer_id = c.id WHERE o.price = 500;
```

Выведенные параметры плана запроса в БД «test_db_b» идентичны параметрам плана запроса в БД «test_db_a».

```

root@ubuntu: /home/admin1
test_db_b=# EXPLAIN ANALYZE SELECT c.name, c.city FROM orders o JOIN customers c ON o.customer_id = c.id WH
ERE o.price = 500;
NOTICE:  *ja_plan_manager*: cached plan was used for query with query_hash = 334402852
          QUERY PLAN
-----
Nested Loop  (cost=0.15..38.35 rows=2 width=64) (actual time=0.081..0.082 rows=0 loops=1)
-> Seq Scan on orders o  (cost=0.00..22.00 rows=2 width=4) (actual time=0.081..0.081 rows=0 loops=1)
    Filter: (price = ja_plan_manager._p(500))
    Rows Removed by Filter: 1000
-> Index Scan using customers_pkey on customers c  (cost=0.15..8.17 rows=1 width=68) (never executed)
    Index Cond: (id = o.customer_id)
Planning Time: 0.125 ms
Execution Time: 0.098 ms
(8 rows)

test_db_b=# 

```

Рисунок 5.16 – Вывод плана запроса в БД «test_db_b»

6. УДАЛЕНИЕ КОМПОНЕНТА

Удаление компонента производится SQL-командой:

```
DROP EXTENSION ja_plan_manager;
```

После чего необходимо убрать загрузку модуля из postgresql.conf, поставив знак # или удалив имя расширения из списка расширений.

```
#shared_preload_libraries = 'ja_plan_manager'
```

6.1. Отключение режима использования плана запросов

Отключить режим использования планов запросов:

```
UPDATE ja_plan_manager.jpm_plans SET enable = false where  
query_hash = 334402852;
```

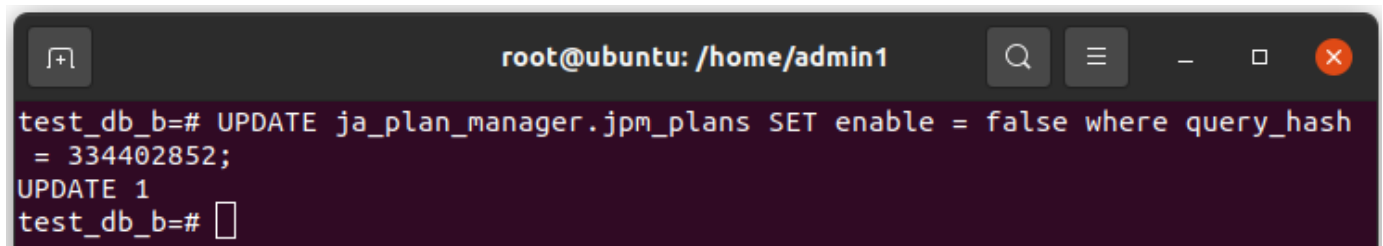


Рисунок 6.1 – Отключение режима использования планов запросов

Затем выполнить анализ плана запроса:

```
EXPLAIN ANALYZE SELECT c.name, c.city FROM orders o JOIN  
customers c ON o.customer_id = c.id WHERE o.price = 500;
```

Убедиться, что изменились параметры плана запроса. В частности, в плане запроса используется сканирование индекса таблицы.

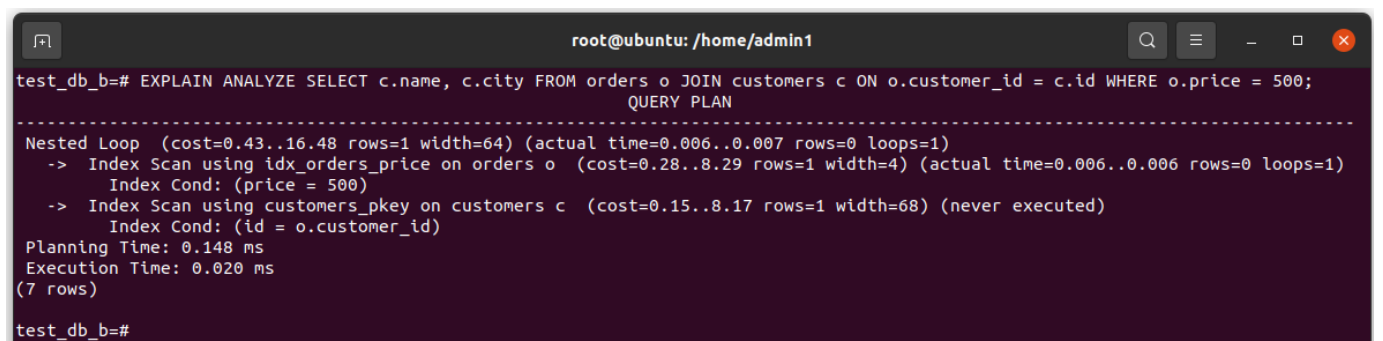


Рисунок 6.2 – Новый план запроса

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

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

| | | |
|------|---|---|
| SQL | – | Structured Query Language – язык структурированных запросов |
| БД | – | База данных |
| ОС | – | Операционная система |
| СУБД | – | Система управления базами данных |

